Proyecto de Haskell

Ciencias de la Computación Universidad de La Habana

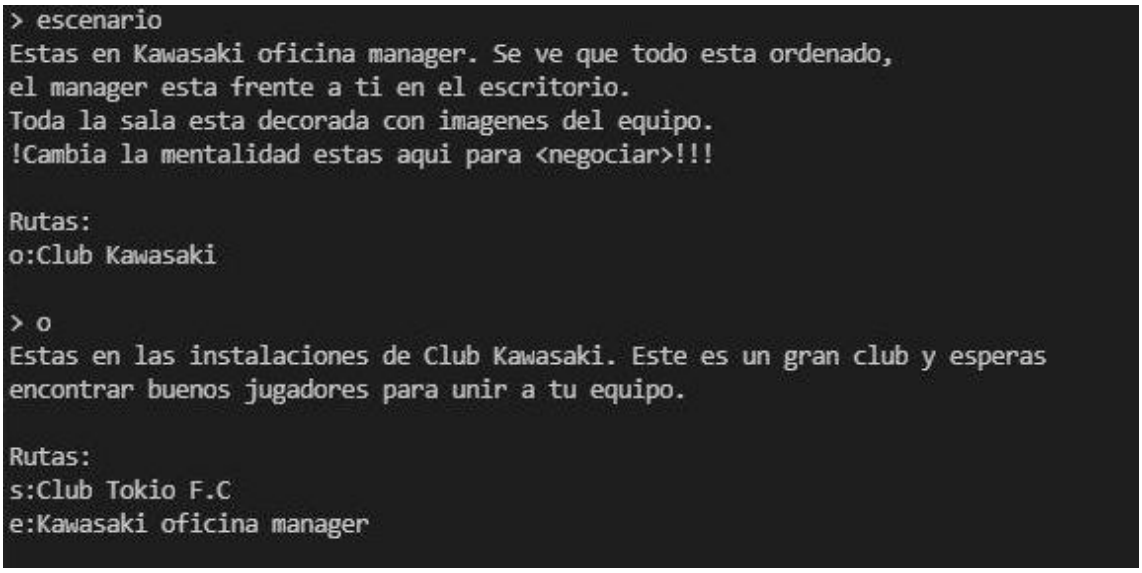
Integrantes: David Cabrera García C-312

Abel Molina Sánchez C-311

Nombre del Proyecto: *Manager*

Descripción del juego:

Manager es un juego de la modalidad Aventura de Texto, implementado en Haskell. El juego intenta simular el trabajo de un director deportivo de un club de futbol. El jugador se adentra en el momento en el que inicia su trabajo en un nuevo club, y debe cumplir sus objetivos de formar un equipo competitivo para la próxima temporada del equipo. La aventura se centra en un club ubicado en Argentina. El jugador tiene la posibilidad de desplazarse por un extenso mapa, que le permitirá visitar clubes de América, de Europa y hasta de Asia. Para lo mismo se constará de dos formas de movimiento para desplazarse por el mapa. Lo podrá hacer por rutas que unen lugares adyacentes en el mapa, ejemplo : al norte de la oficina se encuentra la plaza, por tanto al sur de la plaza se encuetra la oficina. La otra forma de desplazamiento que incluye el juego es para unir destinos lejanos, y se realiza a través de vuelos. Los vuelos se realizan entre aeropuertos, mientras que los aeropuertos son a su vez lugares y poseen rutas que lo unen otros destinos, de esta forma todo el mapa es un grafo conexo. En cambio, no todos los aeropuertos poseen vuelos a todos los destinos, con lo cual por momentos será necesario buscar conexiones para llegar a los objetivos.



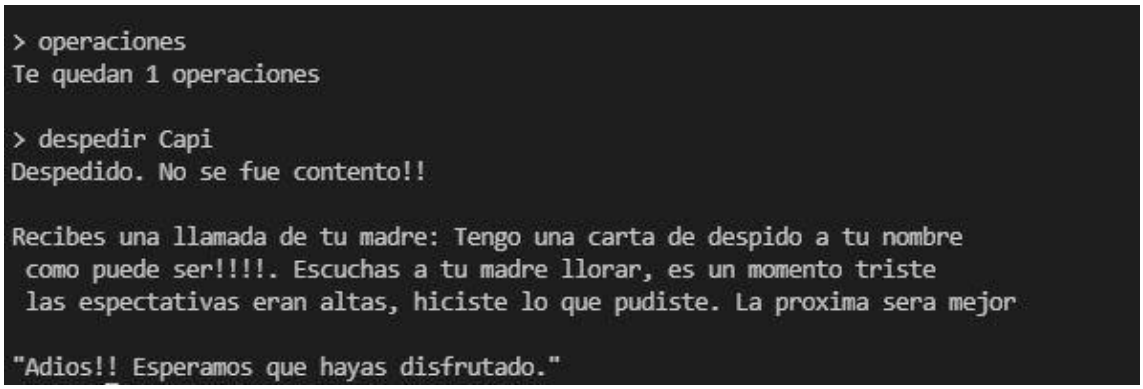
Ejemplo de interacción

Como el objetivo central de un manager es formar un equipo competitivo, en el juego se podrán realizar fichajes de jugadores. Para ello será necesario que el jugador viaje a las oficinas de los managers de otros equipos para poder conocer o contratar a sus futbulistas en lista de venta. Los jugadores tienen cada uno un precio asociado a su nivel(2-7), y el nivel del equipo se ve determinado por la media de los niveles de los jugadores. El jugador tiene como meta formar un grupo de al menos 7 futbolistas, con un nivel general superior a 4.2 puntos. Para ello comenzará con un presupuesto inicial, que disminuirá en el valor de un jugador cuando se contrate y que aumentará en el valor de un jugador cuando se venda. También existe la opción de despedir a un jugador del equipo, esto no afectará de ninguna manera el presupuesto. Las ventas solo pueden realizarse en los Mercados. El manager debe tener en cuenta que el mercado no es infinito, por tanto tiene un límite de 15 operaciones de mercado para alcanzar el objetivo, cada compra, venta, despido disminuye el número de operaciones restantes.

En el juego, como en la vida real, existe otra forma de aumentar el presupuesto del manager, y es a través de contratos con sponsors. Estos patrocinadores se le irán apareciendo al manager en su recorrido por el mundo. Los sponsors no son solo ventajas en el juego, ya que pudiera

ocurrir que los mejores jugadores subscribieran sus fichajes a contar con determinado número de sponsors.

La misma naturaleza del juego garantiza que hayan distintos finales, ya que de por si las combinaciones victoriosas pueden ser varias, y los casos de derrota pueden darse de distintas formas, no hay un único camino que lleve al fracaso o a la victoria en el juego.



Ejemplo de final con derrota

Ejemplo de final con victoria

Características de la implementación:

El juego se desarrolla utilizando comandos de instrucciones que se reconocen, los comandos están cosntituidos por una o dos palabras y tienen una sitaxis fija. El programa reconoce los comandos y ejecuta en consecuencia si la entrada es correcta.

Estos son los *comandos* que se incluyen en el juego:

.ayuda -- mostrar los comandos disponibles.

.n .s .e .o -- moverse en las direcciones(norte, sur,…)

.*volar <destino>* -- viajar al aeropuerto de destino

.*vuelos* -- ver los destinos del aeropuerto actual.

.*firmar <sponsor>* -- firmar con el sponsor seleccionado.

.*despedir <jugador>* -- despedir a tu jugador seleccionado.

.*vender <jugador>* -- vender a tu jugador seleccionado.

.*negociar* -- ver los jugadores en venta de otro club.

.*contratar <jugador>* -- contratar al jugador seleccionado.

.*escenario* -- observar donde te encuentras.

.*equipo* -- ver tu equipo

.*nivel* -- saber el nivel actual del equipo.

.*presupuesto* -- ver tu presupuesto actual.

.*operaciones* -- saber tus operaciones restantes.

.*sponsors* -- ver tus patrocinadores.

. *completar* -- presentar tu equipo al director

.*salir* -- finalizar el juego.

Los comandos abarcan todas las situaciones del juego.

Para ejecutar el juego se puede hacer :

>ghci Manager.hs

y luego llamar main para inicializar.

El proyecto se desarrolló en Windows, intalamos Haskell Plataform versión 8.6.5.

El juego corre en una pc con las siguientes características:

.SO -- Windows 10 64bits

.Procesador --Inter(R)CoreTm i3-6006U@2.00GHz 1.99Ghz

.RAM -- 4.00 GB

Detalles de la implementación:

Para la implementación del juego se utilizó el lenguaje Haskell. Por tanto se hizo uso de las bondades y características del mismo, y ahora explicaremos parte del flujo del programa.

Se definieron dos clases de tipos principales para representar los datos:

```
data Lugar = Plaza {nombre::String, rutas::[(Direccion, PlazaDestino)]}|
             Aeropuerto {nombre::String, rutas::[(Direccion, PlazaDestino
)],vuelos::[AeropuertoDestino]} deriving (Show)
 data Objeto = Persona {nombre_o::String}|

              Jugador {nombre_o::String, valor::Int, nivel::Int}|
              Sponsor {nombre_o::String, ingreso::Int} deriving (Show)
```

Con Lugar abarcábamos los distintos puntos de estancia que puede tener la historia, que lo constituyen los lugares comunes o Plazas y los Aeropuertos. De esta forma, cada lugar tiene su nombre y una lista de rutas accesibles que lo conectan y en el caso de los Aeropuertos se incluyen también los destinos hacia los cuales se pude volar.

A su vez con los objetos definimos el resto de actores del juego, que no son más que los Jugadores y los Sponsors. El objeto Persona se mantuvo por consistencia con la idea inicial ya que debía complementar algunas interacciones que finalmente no se llegaron a implementar.

También con el fin de dar mayor legibilidad e intuitividad al código se utilizaron los sinónimos de tipo como podrían ser, Dirección, Mensaje, PlazaDestino, que representan un String.

Para representar los objetos sobre el terreno, utilizamos listas de duplas de String, que podrían verse como un diccionario clave – valor, donde la clave sería el nombre del objeto, y el valor el nombre de su ubicación. Para su representación utilizamos el sinónimo Ubicaciones.

El flujo de acción del programa se desarrolla en Manager.hs, que es donde está incluida la función main. Controlamos el flujo del programa dentro del método recursivo jugar:: IO(Partida) -> IO(Partida) donde Partida representa un cuarteto que contiene Ubicaciones, Presupuesto, Operaciones, Mensaje. A través de esta construcción se van llevando los cambios de estados dentro del juego. En cada "iteración", el usuario va a recibir el Mensaje contenido en la entrada y va a poder ejecutar una entrada, en caso de ser un comando válido, este será ejecutado por efectivamente por la función ejecutar_comando, que efectivamente devuelve una Patida como resultado de la ejecución.

```
ejecutar_comando:: String -> Ubicaciones -> Presupuesto -> Operaciones -
> Partida
```

Presupuesto y Operaciones constituyen sinónimos de Int

En caso de matchar alguno de los comandos permitidos con la entrada, dentro de ejecutar_comando se llamará a la función específica de ese comando para producir la salida esperada por el usuario.

El juego en su conjunto esta contituido por otros 3 módulos a parte de *Manager.hs.*

Está el módulo *Datos.hs* donde se declaran los principales tipos que se especificaron más arriba, así como consituye el módulo donde se representa el mapa del juego.

Está el módulo *Descripciones.hs* donde se describen los escenarios del juego, el texto que aparece al visitar el Mercado Brasil, el que aparece al ingresar al AeropuertoArgentina, etc.

Y está el módulo *Manager_Aux.hs* donde se definen algunas funciones complementarias para el trabajo con los datos. Es un módulo que como su nombre indica sirve de complemento a las funciones de *Manager.hs* y permite tener el código más organizado.

No fue necesaria la instalación de ningún módulo complementario de Haskell, solo se utilizaron los por nosotros implementados así como *Data.List* y *Data.Char.*

Conclusión del proyecto:

La fase de desarrollo del proyecto nos permitió consolidar y ampliar los conocimientos del lenguaje Haskell así como de la programación funcional. Estamos satisfechos con el resultado obtenido con *Manager*, el tema de la aventura fue motivante, aunque somos concientes de que el proyecto es perfectible e incluso se nos quedaron en el debe algunas características que hubiéramos querido incluir como los comandos más dinámicos y flexibles en cuanto a la entrada. Igual, consideramos que la destreza con la que terminamos programando no tiene comparación con el principio del proyecto.

Bibliografía:

Para el desarrollo del proyecto nos apoyamos principalmente en el libro:

- *Aprende Haskell por el bien de todos*

Y como material auxiliar también utilizamos un pdf obtenido de la web en esta url:

https://loshijosdelagrange.files.wordpress.com/2012/11/guia-rc3a1pida-haskell.pdf