Trabajo UT2 2020

El siguiente trabajo de aplicación de la UT2 del curso de Programación Funcional en 2020 se trata de implementar en Haskell una versión del juego Tatetí. Se trata del tradicional juego de tablero para dos jugadores.

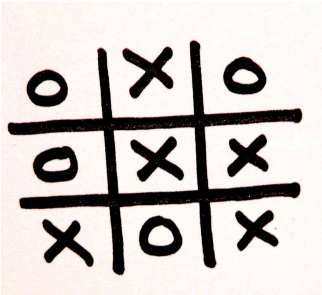

Figure 1: Juego de Tatetí terminado en empate

Tatetí

El juego se juega marcando una de las casillas vacías de un tablero de 3x3 casillas. El primer jugador usa la marca *X* mientras que el segundo usa la marca *O*. Los jugadores se alternan en turnos, marcando una casilla vacía a la vez.

Figure 2: Partida de Tatetí

Ambos jugadores pueden ganar consiguiendo una línea de 3 marcas en cualquier sentido: horizontal, vertical o diagonal (8 líneas posibles). Si el tablero queda sin casillas vacías, el juego termina en un empate.

Planteo

Primero se debe definir una estructura de datos para almacenar la información del estado del juego en cualquier momento de la partida. Dicha estructura de datos se implementará como un tipo de Haskell llamado `TicTacToeGame`. También se debe definir un tipo de Haskell llamado `TicTacToeAction` para representar las acciones posibles para el jugador que mueve (llamado *jugador activo*). Los jugadores se definen de la siguiente forma:

```
type TicTacToePlayer = String
playerX = "X"
playerO = "O"
```

Las funciones a implementar son las siguientes:

- `beginning :: TicTacToeGame`: El estado inicial del juego Tatetí con un tablero vacío de 3x3 casillas.
- `activePlayer :: TicTacToeGame -> TicTacToePlayer`: Esta función determina a cuál jugador le toca mover, dado un estado de juego.
- `actions :: TicTacToeGame -> [(TicTacToePlayer, [TicTacToeAction])]`: La lista debe incluir una y solo una tupla para cada jugador. Si el jugador está activo, la lista asociada debe incluir todos sus posibles movimientos para el estado de juego dado. Sino la lista debe estar vacía.

- `next :: TicTacToeGame -> (TicTacToePlayer, TicTacToeAction) -> TicTacToeGame`: Esta función aplica una acción sobre un estado de juego dado, y retorna el estado resultante. Se debe dar un error si el jugador dado no es el jugador activo, si el juego está terminado, o si la acción no es válida o no es realizable en el estado de juego dado.
- `result :: TicTacToeGame -> [(TicTacToePlayer, Int)]`: Si el juego está terminado retorna el resultado de juego para cada jugador. Este valor es `1` si el jugador ganó, `-1` si perdió y `0` si se empató. Si el juego no está terminado, se debe retornar una lista vacía.
- `showGame :: TicTacToeGame -> String`: Convierte el estado de juego a un texto que puede ser impreso en la consola para mostrar el tablero y demás información de la partida. Por ejemplo:

```
TicTacToe_eq00> showGame (testGames !! 7)
X.O
OXX
..O
TicTacToe_eq00>
```

- `showAction :: TicTacToeAction -> String`: Convierte una acción a un texto que puede ser impreso en la consola para mostrarla.
- `readAction :: String -> TicTacToeAction`: Obtiene una acción a partir de un texto que puede haber sido introducido por el usuario en la consola.
- `testGames :: [TicTacToeGame]`: Una lista de 10 o más estados de juego para pruebas.

El módulo principal del programa debe llamarse `TicTacToe_eqXX` siendo `XX` el número del equipo (siempre de dos dígitos). La cátedra entregará código de referencia para facilitar las pruebas del código solicitado. Éste contiene: definiciones de algunos tipos, esqueletos de funciones a implementar, y una función `main` para poder probar la implementación.

El trabajo debe realizarse en equipo durante la clase del 8 de mayo. Se entregará vía Webasignatura hasta las 23:00 horas del mismo día 8 de mayo.

Extras

Las siguientes tareas extra pueden ser realizadas por los equipos si sobra tiempo.

- *Parser de estados de juegos.* La función `readGame :: String -> TicTacToeGame` implementaría la función inversa a `showGame`, tomando una representación de texto de un estado de juego y retornan la estructura de datos usada en las demás funciones.
- *Juego m,n,k.* En un módulo separado llamado `MNKGame`, implementar una versión generalizada del Tatetí. La función `beginning` tomaría tres argumentos: las cantidades de filas (`m`) y columnas (`n`) del tablero, y el largo de la línea que los jugadores tienen que formar para ganar (`k`). Esos valores pueden ser cualquier número natural mayor que cero. Las demás funciones deberían soportar este cambio.

Referencias

- *Tres en línea* @ Wikipedia.
- *Tic-Tac-Toe* @ Board Game Geek.
