Proyecto del curso de Programación Funcional 2021

Onitama

El proyecto de curso de Programación Funcional en 2021 será implementar en Haskell una versión del juego *Onitama*. Se trata de un juego de tablero de estrategia abstracta para dos jugadores, creado por Shimpei Sato, y publicado por Pegasus Games en 2017.

Figure 1: Partida de Onitama

El juego

Los dos jugadores cuentan con 5 piezas en un tablero cuadrado de 5 por 5 casillas. De las 5 piezas una (la más grande) es el *maestro*, mientras que las otras cuatro (las más pequeñas) son los *aprendices*. Las piezas comienzan en bordes opuestos del tablero, con el maestro en el medio. La casilla que ocupa el maestro al comenzar la partida se conoce como el *santuario*.

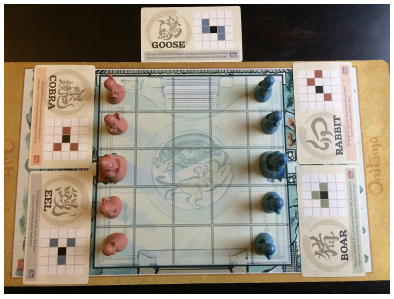

Figure 2: Tablero inicial de Onitama

Los jugadores toman turnos moviendo una pieza a la vez. Los movimientos están determinados por naipes, de las cuales hay 16 disponibles. Antes de comenzar la partida se barajan dichos naipes, a cada jugador se le dan dos naipes al azar, y un quinto naipe se elije también al azar y se deja a un costado del tablero.

En su turno el jugador debe elegir un naipe y una pieza para mover como indica el naipe. Luego de que se mueve una pieza, el jugador toma el naipe que determinó el movimiento y lo cambia por el naipe al costado del tablero. Si una pieza mueve a una casilla ocupada por otra pieza, ésta última es capturada y se quita del tablero.

El juego puede ganarse de dos maneras. Si uno de los maestros es capturado el jugador pierde la partida y su oponente gana (llamado *Way of the Stone*). Si un maestro logra ocupar la casilla santuario de su oponente, gana la partida (llamado *Way of the Stream*).

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 `data` de Haskell llamado `OnitamaGame`.

Figure 3: Cartas de Onitama

También se deben definir dos tipos `data` de Haskell más. Uno llamado `OnitamaAction` para representar los posibles movimientos de los jugadores. Otro llamado `OnitamaCard` para las cartas. Los jugadores se definen de la siguiente forma:

```haskell
data OnitamaPlayer = RedPlayer | BluePlayer deriving (Eq, Show, Enum)
```

Los posibles resultados de la partida se representan con el siguiente tipo:

```haskell
data GameResult p = Winner p | Loser p | Draw deriving (Eq, Show)
```

Las funciones a implementar son las siguientes:

- `deck :: [OnitamaCard]`: El lista con todas las cartas disponibles en el juego.
- `beginning :: [OnitamaCard] -> OnitamaGame`: El estado inicial del juego de Onitama. Esto incluye el orden en el cual están barajadas las cartas del mazo.
- `activePlayer :: OnitamaGame -> Maybe OnitamaPlayer`: Esta función determina a cuál jugador le toca mover, dado un estado de juego. Si ninguno de los jugadores puede mover, e.g. porque el juego está terminado, se retorna `Nothing`.
- `actions :: OnitamaGame -> [(OnitamaPlayer, [OnitamaAction])]`: 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 :: OnitamaGame -> OnitamaPlayer -> OnitamaAction -> OnitamaGame`: Esta función aplica una acción sobre un estado de juego dado, y retorna el estado resultante. Se debe levantar un error si el jugador dado no es el jugador activo, si el juego está terminado, o si la acción no es realizable.
- `result :: OnitamaGame -> [GameResult OnitamaPlayer]`: Si el juego está terminado retorna el resultado de juego para cada jugador. Si el juego no está terminado, se debe retornar una lista vacía.
- `showGame :: OnitamaGame -> 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.
- `showAction :: OnitamaAction -> String`: Convierte una acción a un texto que puede ser impreso en la consola para mostrarla.
- `readAction :: String -> OnitamaAction`: Obtiene una acción a partir de un texto que puede haber sido introducido por el usuario en la consola.

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. Se entregará vía Webasignatura hasta el 2 de julio a las 19:00 horas. Inmediatamente luego de la entrega se tomará una defensa a todos los miembros de cada equipo.

Figure 4: Partida de Onitama

Extras

Las siguientes tareas extra pueden ser realizadas por los equipos para agregar puntos a su calificación, por encima de los 100 puntos que vale el proyecto sin éstos.

Variantes de juego.

Se pretende habilitar variantes al juego original, agregando las siguientes definiciones para modificar al juego original.

```
data OnitamaConfig = OnitamaConfig { configDeck :: [OnitamaCard] } deriving (Eq, Show)

variant :: OnitamaConfig -> OnitamaGame
variant config = beginning (configDeck config)
```

Al tipo `OnitamaConfig` se le podrán ir agregando datos para soportar las modificaciones a implementar. Esto implicará cambios en `variant` y el resto de las funciones solicitadas. Las posibles modificaciones son:

- *Variante de cantidad de cartas en mano* (5 puntos). El juego original indica que cada jugador tiene 2 cartas en la mano. Agregar a `OnitamaConfig` el valor `configHandSize :: Int`, para indicar una cantidad de cartas diferente en la mano de los jugadores durante la partida. Debe validarse que la misma sea mayor que 1 y menor que 8.
- *Ahogamiento* (5 puntos). En el juego original, si el jugador no puede mover debe pasar y cambiar una carta. Agregar a `OnitamaConfig` el valor `configStalemate :: Bool`. Si ese valor es `False` el comporatmiento es el original, pero si es `True` implica que un jugador que no puede mover pierde la partida.

Jugador Inteligente.

La plantilla de código entregada contiene una mínima implementación de juego. Se define un tipo para agentes jugadores, y se implementan dos: el *jugador aleatorio* (que elige sus movimientos al azar) y el *jugador de consola* (que toma sus movimientos de la entrada estándar).

Una tarea extra de hasta 15 puntos consiste en implementar un jugador inteligente. Se considerará correcto si el jugador le gana significativamente al jugador aleatorio.

Figure 5: Set de Onitama

Referencias

- *Onitama* @ Wikipedia: `https://en.wikipedia.org/wiki/Onitama`.
- *Onitama* @ Board Game Geek: `https://boardgamegeek.com/boardgame/160477/onitama`.
- *Onitama* @ Pegasus Games: `https://www.arcanewonders.com/game/onitama/`
