Proyecto de Simulación y Programación Declarativa Agentes

Rodrigo García Gómez CC-412

El problema

Se tiene un ambiente de información completa dado por un tablero rectangular de N x M casillas. Cada t turnos el ambiente realiza un cambio aleatorio dado por los movimientos de los niños y sus respectivas generaciones de suciedad. Cada turno los agentes ejecutan sus acciones modificando el medio. Cada vez que se alcanza el turno de cambios aleatorios en el ambiente, estos ocurren junto con las acciones de los agentes en una misma unidad de tiempo. Los elementos que pueden existir en el ambiente son obstáculos, suciedad, niños, corral y los robots (que son los agentes).

1. Obstáculos:
   Ocupan una única casilla. Pueden ser movidos sólo por los niños. No pueden ser movidos a ninguna casilla ocupada por algún otro elemento del ambiente.

2. Suciedad:
   Ocupan una única casilla del ambiente. Es generada por los niños y sólo puede surgir en una casilla previamente vacía. También pueden aparecer en el estado inicial del tablero.

3. Corral:
   El corral ocupa casillas adyacentes en número igual al total de niños presentes en el ambiente. El corral no puede ser desplazado. En una casilla del corral puede coexistir un único niño. En una casilla vacía del corral puede entrar un robot. En una casilla del corral pueden coexistir al mismo tiempo un niño y un robot sólo si el robot está cargando al niño o si lo acaba de soltar.

4. Niño:
   Ocupa una única casilla. Durante el turno de cambio en el ambiente se mueven de forma aleatoria de a una casilla adyacente ser posible (No se puede mover a casillas ocupadas por otros niños, robots, suciedad o el corral). Si la casilla objetivo a moverse está ocupada por un obstáculo, este se desplaza en la dirección de la que venía el niño. En el caso de haber más obstáculos en esa dirección, todos son desplazados a menos que el más lejano de estos no pueda ser desplazado por estar ocupada la casilla objetivo. Luego de mover un obstáculo el niño ocupa su posición. Los niños son responsables de que aparezca la suciedad. La suciedad es generada de forma aleatoria en la cuadrícula de 3x3 centrada en la casilla del niño, luego de que este se mueva hacia alguna de sus casillas adyacentes. Si antes de realizar su movimiento, la cuadrícula tenía sólo

al niño en la que se centraba, entonces se genera hasta una suciedad. Si además del niño en el centro había otro niño en la cuadrícula, se generan hasta 3 suciedades. Si había más de un niño además del del centro, se generan hasta 6 suciedades. Al estar encerrados en una casilla del corral, o atrapados por un robot, los niños no pueden moverse ni ensuciar.

5. Robot de casa:
   Es el encargado de limpiar y controlar a los niños. El robot decide en cada momento que acción realizar. Si no carga a un niño, puede desplazarse sólo una a una casilla adyacente. Si carga a un niño, puede desplazarse hasta dos casillas consecutivas. Puede también realizar acciones de limpiar y cargar niños. Si se mueve a una casilla con suciedad, en el próximo turno puede decidir si limpiar o moverse. Si se mueve a una casilla ocupada por un niño, inmediatamente lo carga. Si se mueve a una casilla vacía del corral y lleva a un niño cargado, puede dedicar un turno a soltarlo o puede moverse y mantenerlo cargado. Al ser dejado el niño en una casilla coexisten hasta el otro turno el robot y el niño, sin este ser cargado por el primero.

El Objetivo del robot es mantener la casa limpia. La casa se considera limpia si menos del 60 porciento de las casillas vacías están ocupadas por suciedad.

Instrucciones de uso

Para ejecutar el proyecto desde *ghci* debe importar el script *main.hs* ubicado dentro de */app*. Luego se llama a la función `main`.

```
*Main Lib Paths_agents> main
```

Igualmente se puede teclear el comando `Stack run` en la consola estando parado en la raíz del proyecto (si tiene instalado Stack).

A continuación se pedirá al usuario dos entradas. Primero un número entero que se utiliza para seleccionar el tablero. El número 0 generará un tablero aleatorio de tamaño 8x8. Los números 1-4 se utilizan para seleccionar uno de los tableros perdefinidos en el cuerpo de la función: `selectInitialBoard :: Int -> IO Board`. Cualquier número mayor que 4 ingresado generará un tablero completamente vacío.

La segunda entrada representará la cantidad de turnos que deben pasar durante la simulación (en cada turno los robots de limpieza hacen sus movimientos) antes de que ocurra un cambio aleatorio en el ambiente (Los niños harán sus movimientos y provocarán el surgimiento de nuevas casillas con suciedad)

```
"insert index of initial board"
0
"insert number of iterations to shuffle"
5
```

A partir de este punto, cada vez que el usuario presione "enter", se llevará a cabo un turno y se imprimirá en la consola una representación del tablero con los resultados de los movimientos de los agentes (y del cambio en el ambiente si corresponde) durante el mismo. Los símbolos en el tablero representan los siguientes elementos:

$Empty >$ Casilla vacía

$(Obstacle) >$ Casilla ocupada por un obstáculo

$\sim\sim Dirt \sim\sim \ >$ Casilla con suciedad

$\{Corral\} >$ Casilla ocupada por un corral

$\{--C+K--\} >$ Casilla ocupada por un corral con un niño dentro

$\{[C+R]\} >$ Casilla ocupada por un corral con un robot dentro

$\{[--C+K+R--]\} >$ Casilla ocupada por un corral con un robot dentro mientras el robot lleva un niño

$\{[--C+R => K--]\} >$ Casilla ocupada por un corral con un robot dentro que acaba de soltar al niño que llevaba

$--Kid-- >$ Casilla ocupada por un niño

$[Robot] >$ Casilla ocupada por un robot

$[\sim\sim R+D \sim\sim] >$ Casilla con suciedad ocupada por un robot

$[--R+K--] >$ Casilla ocupada por un robot que lleva a un niño

$[--\ \sim\sim\ R + K + D\ \sim\sim\ --] >$ Casilla con suciedad ocupada por un robot que lleva a un niño

Para detener la simulación, antes de presionar "enter" durante cualquier turno puede escribir `abort` en la consola. Además, la simulación se detendrá automáticamente si la cantidad de casillas con suciedad supera el 60 porciento del total de casillas del tablero o si todos los niños se encuentran dentro del corral o siendo llevados por robots y no queda ninguna casilla sucia (al estar todos los niños atrapados, ya no se podrá generar suciedades nuevas).

Detalles de implementación

El tipo tablero está constituido de la siguiente forma

```
data Board = Board {b :: [[BoardObject]], w :: Int, h :: Int,
                    obstacles :: [(Int,Int)],
                    kids :: [(Int,Int)],
                    corrals :: [(Int, Int)],
                    robots :: [(Int,Int)],
                    dirts :: [(Int,Int)]
                    } deriving (Show)
```

A la matriz retornada por `b` se le asocian las casillas. Cada una estará formada por un `BoardObject` (Los `BoardObject` se explicarán más adelante). Además se tiene en `w` y `h` las dimensiones del tablero y en `obstacles, kids, corrals, robots y dirts`, unas listas de tuplas con las correspondientes coordenadas x, y para cada elemento.

```
data BoardObject = Empty {x :: Int, y :: Int, typ :: String} -- em
                 | Obstacle {x :: Int, y :: Int, typ :: String} -- ob
                 | Dirt {x :: Int, y :: Int, typ :: String} -- di
                 | EmptyCorral {x :: Int, y :: Int, typ :: String} --
                    ec
                 | CorralAndKid {x :: Int, y :: Int, typ :: String}
                    -- ck
                 | CorralAndKidAndRobot {x :: Int, y :: Int, typ ::
                    String} -- ckr
                 | CorralAndRobot {x :: Int, y :: Int, typ :: String}
                    -- cr
                 | Kid {x :: Int, y :: Int, typ :: String} -- ki
                 | Robot {x :: Int, y :: Int, typ :: String} -- ro
```

```
  | RobotAndKid {x :: Int, y :: Int, typ :: String} --
      rk
  | CorralAndKidAndRobotRelease {x :: Int, y :: Int,
      typ :: String} -- ckrr
  | RobotAndDirt {x :: Int, y :: Int, typ :: String}
      -- rd
  | RobotAndKidAndDirt {x :: Int, y :: Int, typ ::
      String} -- rkd
```

Cada constructor de `BoardObject` representará uno de los posibles objetos que pueden estar en una casilla durante un turno. A `x` y `y` se le asocian las coordenadas de cada objeto y el string `typ` tendrá un string único para cada tipo (comentado al final de cada línea).

Los números aleatorios son generados mediante la función `randomRIO :: (Random a, Control.Monad.IO.Class.MonadIO m) => (a, a) -> m a`, importada de System.Random. Esta función retorna un $IOInt$ por tanto, teniendo en cuenta la pureza de las funciones de haskell, cualquier función que involucre un random deberá estar contenida en un *do*.

Es importante destacar que cualquier función que afecte la situación del tablero, recibirá al mismo como parámetro, creará un tablero nuevo y lo retornará. Durante cada turno (cada iteración del ciclo principal), se mueve a cada robot y se crea un tablero nuevo con los resultados de estos movimientos. De ser turno de cambio aleatorio, también se crea nuevo tablero con los cambios hechos por los niños.

El ciclo principal se ejecuta en la función `mainLoop :: Int -> Int -> Board -> IO ()`. En cada iteración se realiza primero las acciones de los agentes mediante la función `moveRobots :: [(Int, Int)] -> Board -> Board` que recibe. Luego, en dependencia de si se alcanzó un turno de movimiento del ambiente, se mueve a los niños con `moveKids :: [(Int, Int)] -> [[BoardObject]] -> [(Int, Int)] -> [String] -> IO ([[BoardObject]], [(Int, Int)], [String])` y se reinicia el contador de turnos, o se sigue hacia la siguiente iteración con el contador disminuido en uno.

En `moveKids` se iterará por la lista de niños (lista que tiene a todo aquel que no haya sido encerrado en un corral o atrapado por un robot) y actualizará el tablero con el movimiento de cada uno y su respectiva generación de suciedad. En `moveKid` se tratará siempre de mover a los niños a su alrededor de forma aleatoria. Su intento de movimiento puede tener como objetivo cualquiera de las 9 casillas pertenecientes a la cuadrícula de 3x3 centrada

en dicho niño (incluyendo la central que implicará no moverse). Para esto se generan dos enteros random entre -1 y 1 y se suman a la posición actual, el resultado será la posición objetivo. De resultar como objetivo una posición que sale de los límites del tablero se tomará como no movimiento, de lo contrario se invocará a `tryMoveKid`. Esta última función hará algo de lo siguiente en dependencia del tipo de la casilla objetivo.
1- Si es una casilla vacía, se sustituye a esta por una casilla de tipo Kid, mientras que la casilla desde la que se mueve pasará a estar ocupada por un Empty.
2- Si es una casilla de tipo Obstáculo, se invocará a `tryPush` para tratar de mover a dicho obstáculo en la dirección del movimiento (y a todos los obstáculos que estén en esa dirección). Si es posible mover el obstáculo, la casilla objetivo se sustituye por un objeto kid, la casilla desde la que se mueve tendrá un Empty y la primera casilla vacía en la dirección del obstáculo tendrá un tipo Object
3- De no ser alguno de los casos anteriores, el niño no podrá moverse.
De encontrarnos con alguno de los casos 1 o 2, entonces también se tratará de generar suciedad alrededor de la casilla desde la que se ha movido el niño. La función `checkAround` retornará n = (1, 3 o 6) en dependencia de la cantidad de niños de la cuadícula de 3x3 centrada en esta casilla. Luego `fillWithDirt` retornará un tablero nuevo con hasta n casillas ocupadas anteriormente por un Empty, que tendrán ahora objetos Dirt.

La función `moveRobots` iterará por cada robot del tablero para llamar a `robotMekeDecision`. Por cada robot, primeramente se llamará a la función `bfs` para obtener una matriz de dimensiones similares a las del tablero, que tendrá en cada posición una tupla con la distancia y el camino más rápido desde el robot en cuestión hasta la casilla del tablero correspondiente con su índice (de no haber camino, la distancia será -1). El estado de cada robot está representado por el *typ* correspondiente al *BoardObject* que lo representa y se tomará una decisión en dependencia de cadatipo de estado.

Agente reactivo

- *ro*, *ckrr/* o *cr* $\Longrightarrow$ El robot está sólo en la casilla vacía o del corral y no lleva ningún niño, o acaba de soltar (en el turno anterior) a un niño dentro del corral, por tanto se busca la tierra y el niño más sercano y, de estos, el objeto de menor distancia se convertirá en el objetivo a moverse del robot (dando prioridad a recoger niños sobre la suciedad). Una vez decidido el objetivo, se desplaza al robot hacia el primer paso de su camino más corto

(o de poder caminar dos pasos y ser el caso *ckrr*, hacia el segundo).
- *rk* $\Longrightarrow$ En este caso el robot lleva a un niño sin estar sobre una suciedad o dentro del corral. Los objetivos esta vez serán la casilla del corral más cercana o la casilla con suciedad más cercana. El objeto de menor distancia se convertirá en el objetivo de movimiento del robot (dando prioridad a alcanzar el corral). Una vez decidido el objetivo, el robot se desplaza, de ser posible, a la segunda posición del camino más corto al objetivo, y de no serlo, a la primera.
- *rd* o $\Longrightarrow$ En este caso el robot se encuentra sobre una casilla con suciedad, ya sea que cargue o no a un niño, siempre se tomará la decisión delimpiarla, y por tanto, la casilla objetivo será la misma casilla actual.
- *ckr* $\Longrightarrow$ El robot lleva a un niño y está en una casilla del corral. en este caso se invoca a `searchForTargetCorral` para encontrar cuál es la casilla del corral con menos espacios libres alrededor (la que esté más rodeada de bordes del tablero, casillas del corral con niños u obstáculos) y se convertirá esta en el objetivo del robot. De esta forma se evita llenar de niños el corral de forma que no se pueda eventualmente acceder a las casillas centrales. En caso de ser la casilla objetivo igual a la casilla actual, significa que se soltará al niño. De lo contrario se intenta avanzar dos o una posición en dependencia del camino más cercano.

Agente proactivo

- *ro*, *ckrr/* o *cr* $\Longrightarrow$ El robot está sólo en la casilla vacía o del corral y no lleva ningún niño, o acaba de soltar (en el turno anterior) a un niño dentro del corral, por tanto se buscará al niño más cercano y se pondrá su posición como objetivo. De ser -1 la distancia al niño más cercano, significará que o bien todos los niños han sido capturados, o que no hay acceso a los restantes; en este caso se pone como objetivo la suciedad más cercana. Una vez decidido el objetivo, se desplaza al robot hacia el primer paso de su camino más corto (o de poder caminar dos pasos y ser el caso *ckrr*, hacia el segundo).
- *rk* $\Longrightarrow$ En este caso el robot lleva a un niño sin estar sobre una suciedad o dentro del corral. El objetivo esta ocasión será la casilla del corral más cercano. De ser -1 la distancia a la casilla de corral más cercana, significará que o bien todo el corral ha sido llenado, o que ha quedado fuera de alcance y en este caso se desplazará hacia la tierra más cercana. Una vez decidido el objetivo, el robot se desplaza, de ser posible, a la segunda posición del camino más corto al objetivo, y de no serlo, a la primera.
- *rd* o $\Longrightarrow$ En este caso el robot se encuentra sobre una casilla con suciedad, ya sea que cargue o no a un niño, siempre se tomará la decisión delimpiarla

ya que sería un desperdicio de tiempo no hacerlo, y por tanto, la casilla objetivo será la misma casilla actual.
- *ckr* $\Longrightarrow$ El robot lleva a un niño y está en una casilla del corral. en este caso se invoca a `searchForTargetCorral` para encontrar cuál es la casilla del corral con menos espacios libres alrededor (la que esté más rodeada de bordes del tablero, casillas del corral con niños u obstáculos) y se convertirá esta en el objetivo del robot. De esta forma se evita llenar de niños el corral de forma que no se pueda eventualmente acceder a las casillas centrales. En caso de ser la casilla objetivo igual a la casilla actual, significa que se soltará al niño. De lo contrario se intenta avanzar dos o una posición en dependencia del camino más cercano.

En cualquier ocasión que no se encuentre objetivo de desplazamiento, el robot decidirá quedarse quieto. Una vez decidida la casilla objetivo, la función `updateTypes` retornará los tipos que resultan del movimiento (el de la casilla objetivo y el de la casilla desde donde se mueve el robot, note que pueden ser los mismos si el robot decide no desplazarse por falta de objetivo o para limpiar suciedad o dejar a un niño) para ser puestos en el tablero nuevo que se retornará.

Modelos de agentes considerados

Los modelos de agentes que fueron considerados para la solución del priblema fueron el reactivo y el pro-activo Para la solución del problema se consideraron dos modelos de agente, el reactivo y el pro-activo.
El agente pro-activo debe ser capaz de tomar la iniciativa para lograr sus objetivos y de mostrar un comportamiento dirigido a estos. Para nuestro problema, el comportamiento de los niños (capaces de dejar suciedad) entorpecen al robot a la hora de cumplir con sus objetivos. Por tanto, el robot tendría un comportamiento centrado en dejar primero a todos los niños en sus corrales para luego dirigirse a limpiar las casillas con suciedad. Este comportamiento se mantendría hasta el cumplimiento de la meta de ocupar todos los corrales con niños, siempre que no ocurra la condición de parada de la simulación en la que la suciedad alcance antes la mayoría.
El agente reactivo, por el conrario, es capaz de analizar el ambiente y tomar decisiones óptimas de frente a los cambios que ocurren en este. En cada turno, el agente toma decisiones de acuerdo a su estado y a las casillas a su alrededor. Puede, de no llevar niño en ese momento, decidir si cargar a un niño o en cambio dirigirse a limpiar una casilla más próxima. O en cambio, si lleva un niño, puede decidir si moverse inmediatamente hacia el corral o si

posponer esta tarea para primeramente limpiar una suciedad. Por supuesto, siempre se da prioridad a las tareas de recolección de niños, pues mientras más de llos haya sueltos, más se ensuciará el ambiente, lo cual resulta contraproducente Este agente se adapta mejor a las variaciones del ambiente pues en cada turno analiza el mismo y toma una decisión independiente.

Consideraciones obtenidas a partir de la ejecución de las simulaciones del problema

Podemos llegar a la conclusión de que el agente reactivo logra representar una mejor solución al problema. Este se puede adaptar mejor al ambiente y las decisiones que toma, en su mayoría, llegan a ser las más óptimas. Es cierto que la tarea más importante es la de guardar a los niños, pero si de tener una suciedad cerca y un niño lejos (o cargar a un niño y tener lejos al corral), resultaría un desperdicio de tiempo no pasar primero por la suciedad para eliminarla. En dependencia de la cantidad de niños y de su velocidad de acción, es posible que se llene el tablero antes de lograr guardar a los niños con el agente pro-activo. En cambio, el reactivo puede demorarse más en lograr la limpieza, pero mantiene el ambiente más controlado y corre menos riesgo de fallo.