Proyecto de Simulación y Programación Declarativa Agentes.

Gabriela B. Martínez Giraldo C 412

El presente proyecto pretende hacer una implementación del problema presentado en el lenguaje de programación Haskell.

Indicaciones generales:

El ambiente en el cual intervienen los agentes es discreto y tiene la forma de un rectángulo de

N × M . El ambiente es de información completa, por tanto todos los agentes conocen toda la información sobre el agente. El ambiente puede varıiar aleatoriamente cada t unidades de tiempo. El valor de t es conocido.

Las acciones que realizan los agentes ocurren por turnos. En un turno, los agentes realizan sus acciones, una sola por cada agente, y modifican el medio sin que este varíe a no ser que cambie por una acción de los agentes. En lo siguiente, el ambiente puede variar. Si es el momento de cambio del ambiente, ocurre primero el cambio natural del ambiente y luego la variación aleatoria. En una unidad de tiempo ocurren el turno del agente y el turno de cambio del ambiente.

Los elementos que pueden existir en el ambiente son obstáculos, suciedad, niños, el corral y los agentes que son llamados Robots de Casa. A continuación se precisan las características de los elementos del ambiente:

Obstáculos: estos ocupan una única casilla en el ambiente. Ellos pueden ser movidos, empujándos por los niños, una única casilla. El Robot de Casa sin embargo no puede moverlo. No pueden ser movidos a ninguna de las casillas ocupadas por cualquier otro elemento del ambiente.

Suciedad: la suciedad es por cada casilla del ambiente. Solo puede aparecer en casillas que previamente estuvieron vacías. Esta, o aparece en el estado inicial o es creada por los niños.

Corral: el corral ocupa casillas adyacentes en número igual al del total de niños presentes en el ambiente. El corral no puede moverse. En una casilla del corral solo puede coexistir un niño. En una casilla del corral, que esté vacía, puede entrar un robot. En una misma casilla del corral pueden coexistir
un niño y un robot solo si el robot lo carga, o si acaba de dejar al niño.

Niño: los niños ocupan solo una casilla. Ellos en el turno del ambiente se mueven, si es posible (si la casilla no está ocupada: no tiene suciedad, no está el corral, no hay un Robot de Casa), y aleatoriamente (puede que no ocurra movimiento), a una de las casilla adyacentes. Si esa casilla está ocupada por un obstáculo este es empujado por el niño, si en la dirección hay más de un obstáculo, entonces se desplazan todos. Si el obstáculo está en una posición donde no puede ser empujado y el niño lo intenta, entonces el
obstáculo no se mueve y el niño ocupa la misma posición.

Los niños son los responsables de que aparezla suciedad. Si en una cuadrícula de 3 por 3 hay un solo niño, entonces, luego de que él se mueva aleatoriamente, una de las casillas de la cuadrícula anterior que esté vacía puede haber sido ensuciada. Si hay dos niños se pueden ensuciar hasta 3. Si hay tres niños o más pueden resultar sucias hasta 6.

Los niños cuando están en una casilla del corral, ni se mueven ni ensucian.

Si un niño es capturado por un Robot de Casa tampoco se mueve ni ensucia.

Robot de Casa: El Robot de Casa se encarga de limpiar y de controlar a los niños. El Robot se mueve a una de las casillas adyacentee, las que decida. Solo se mueve una casilla sino carga un niño. Si carga un niño
pude moverse hasta dos casillas consecutivas.

También puede realizar las acciones de limpiar y cargar niños. Si se mueve a una casilla con suciedad, en el próximo turno puede decidir limpiar o moverse. Si se mueve a una casilla donde está un niño, inmediatamente lo carga. En ese momento, coexisten en la casilla Robot y niño.

Si se mueve a una casilla del corral que está vacía, y carga un niño, puede decidir si lo deja esta casilla o se sigue moviendo. El Robot puede dejar al niño que carga en cualquier casilla. En ese momento cesa el movimiento del Robot en el turno, y coexisten hasta el próximo turno, en la misma
casilla, Robot y niño.

Objetivos: El objetivo del Robot de Casa es mantener la casa limpia. Se considera la casa limpia si el 60 % de las casillas vacias no están sucias.

Principales Ideas seguidas para la solución del problema:

Para solucionar el problema presentado se implementó en haskell un pequeño mecanismo para simular el ambiente dado: un habitación representada por baldosas numeradas tipo matriz, con suciedad, bebés y obstáculos de por medio; y la respuesta de los llamados Robots de Casa en orden de mantener el sitio limpio.

Las simulaciones ocurren por un período de tiempo máximo que se puede especificar, hasta que la antidad de casillas con suciedad haya excedido el 40% o bien hasta que los robots hayan culminado con sus tareas o no tengan más movimientos posibles (puesto que pueden haber quedado en una situación donde los ebés en coordinación con el medio no les permitan moverse).

Se puede realizar la simualción con un ambiente dado o generar uno especificando el número de elementos de cada tipo que se desean colocar.

Modelos de Agentes considerados:

En este implementación se proveen dos modelos de Robots de Casa, que son los agentes inteligentes de nuestro problema:

R2B2: Un agente reactivo, que es capaz de percibir el ambiente y responder oportunamente en función de lograr sus objetivos.

Su objetivo es mantener la casa limpia, y para ello se enfoca en pequeñas tareas según el estado en que se encuentre:

-Si está sobre una baldosa con suciedad limpiarla.

-Si tiene un niño en brazos y no hay mas niños sueltos, buscar un camino hacia la casilla con suciedad más cercana.

-Si tiene un niño en brazos y se ha alcanzado el corral, dejarlo.

-Si tiene un niño en brazos y hay más sueltos, ir a dejar el que tiene en el corral.

-Si no tiene ningún bebé consigo, verificar si es mejor ir a perseguir uno o está más cercano a una suciedad.

Note que este robot es bastante práctico, va tratando de recoger a los niños para que nogeneren suciedad y por el camino va limpiando todo lo que se encuentre. Trata de seguir siempre el objetivo más próximo y optimiza su modo de limpieza, o sea se aprovecha de la ventaja de limpiar con un bebé en brazos para dar una mayor cantidad de pasos en un turno.

C3PO: Un agente pro-activo, o sea GOAL DIRECTED. Este se enfoca en lograr el objetivo de capturar a los pequeños. Haciendo referencia a la famosa saga, podemos decir que C3PO es más fino que R2B2, así que deja para el final las tareas de limpieza.

En este caso las intrucciones de C3PO son claras:

-Si tiene un niño en brazos y no hay mas niños sueltos, buscar un camino hacia la casilla con suciedad más cercana.

-Si tiene un niño en brazos y se ha alcanzado el corral, dejarlo.

-Si tiene un niño en brazos y hay mas sueltos o ya no hay suciedad por limpiar, ir a dejar el bebé que lo acompaña en el corral.

-Si no tiene ningún bebé consigo, ir a buscar al más cercano.

-Si está sobre una baldosa con suciedad, limpiarla.

-De lo contrario, ir a limpiar la baldosa sucia más cercana.

Como se puede ver por el orden que da C3PO a las tareas, los niños son su prioridad número 1. Esto puede implicar a veces que se ignoren suciedades cercanas o incluso en su misma casilla en aras de alcanzar algún bebé cercano, lo cual puede resultar provechoso cuando estos peques se anden moviendo mucho, pues así se alcanzarían antes.

Ideas seguidas para la implementación:

Para implementar las ideas descritas se utilizó el lenguaje de programación haskell como ya se había comentado.

El uso de elementos propios de lenguajes funcionales como el **lazy loading** fueron fundamentales para la obtención de un código eficiente y robusto sin la necesidad de hacer un trabajo extra en ingeniería de sofware.

En el archivo *Env.hs* se definen los elementos básicos de nuestro juego o simulación como se le quiera llamar, los cuales están dados por constructores del data type *Element*, cada uno con su nombre y sus coordenadas. En el caso de los robots o los bebés se lleva también un boleano *wcompany* que nos dice si están acompañados, o sea en el caso del robot si carga a alguien y del bebé si está siendo cargado.

```
--Data type to represent the elements present in the environment
data Element = Dirt {row :: Int, column :: Int}
             | Obstacle {row :: Int, column :: Int}
             | Playpen {row :: Int, column :: Int}
             | Baby {row :: Int, column :: Int, wcompany :: Bool}
             | Robot {row :: Int, column :: Int, wcompany :: Bool} deriving
(Show, Data, Typeable)
```

Como se puede observar en la imagen, se utilizó la sintaxis de **records** de Haskell para poder hacer referencia de forma rápida a sus propiedades, en este caso fila (*row*), columna (*column*) y el estado (*wcompany*). Así nos ahorramos el tener que definir funciones aparte para extraer dichos datos.

La simulación se inicia con el método *startSimulation* definido en el archivo GameFlow.hs, donde se implementan las funciones más generales de la simulación.
Este recibe los valores que se desean utilizar para una corrida y inicia el andamiaje necesario.

El flujo seguido en cada iteración (equivalente a un segundo de simulación) consiste en:

1. Efectuar el movimiento de los bebés.
2. Si el tiempo transcurrido es multiplo de t(con t el tiempo dado de variación aleatoria) asegurar que los bebés generen suciedad.
3. Mover los robots.

Para simular el movimiento de los bebés, en (EnvChanges.hs(), se implementaron funciones que se apoyan en valores random en pos de decidir si efectuar un movimiento o no en un turno dado, y la dirección del mismo en caso de decidir realizarlo (se genera un número random de 0 a 4 donde 0 indica no movimiento y el resto cada una de las direcciones, ver el código que se muestra a continuación). Se asegura siempre que la dirección escogida está disponible y que, si se encuentran obstáculos por medio, estos se corran también.

```
--returns one of 5 directions: none, rigth, down, left, up
getDirection :: Int -> (Int, Int)
getDirection n = case n of 0 -> (0,0)
                           1 -> (0,1)
                           2 -> (1,0)
                           3 -> (0,-1)
                           4 -> (-1,0)
```

A la hora de realocar los bebés se deben considerar sólo los que se encuentren fuera de los corrales y no estén cargados. Para ello de definió la función takeBabies que se muestra a continuación:

```
--Takes an envrionment end return a list with the free babies found on it
--(free babies aka babies without company and not in playpens)
takeBabies :: [Element] -> [Element] -> [Element]
takeBabies [] env = []
takeBabies (Baby x y c:rest) env | not ( inPlayPen (Baby x y c) env ) && not c =
[Baby x y c] ++ takeBabies rest env
                            | otherwise =  takeBabies rest env
takeBabies (e:rest) env = takeBabies rest env
```

Como se puede apreciar es tan simple como recorrer por la lista de elementos del **environment** en busca de los que sean de tipo **Baby**. Esto se comprueba a través de un **pattern matching** que como bien sabemos es una de las bases de haskell, buscando coincidencias entre el patrón buscado y el elemento actual.

Se considera que los bebés sólo ensucian cuando se encuentran en una cuadrícula de 3x3 junto a los que no estan cargados ni en corrales, puesto que estos son los que pueden jugar con ellos.

Para la generación de suciedad se analizan los niños sueltos.
Se toma la cuadricula de 3x3 centrada en la casilla de cada bebé dado, y se calcula el número de compañeritos que hay en ella. A partir de este valor se estima la cantidad de casillas que debe ensuciar según no. de bebés igual a:

- **0** : de 0 a 1
- **1** : de 0 a 1
- **2 o más**: de 0 a 2

En función de este valor se genera tanta suciedad por dicho peque como sea posible, atendiendo al
no. de casillas disponibles en su cuadrícula.

Todos los robots en simulación deben ser del mismo tipo. Al inicio se elige que tipo de robot se desea utilizar, así C3PO y R2B2 no discuten entre ellos.

En el archivo *Agents.hs()* se definen los métodos para su funcionamiento.

Resaltar que se intenta que persigan el camino más cercano a su objetivo en el momento dado, lo cual se asegura con el método bfs implementado, el cual devuelve una matriz de descubrimiento del bfs clásico, realizado desde una fuente dada hasta un objetivo. Aquí se reciben los elementos que deben ser considerados como impedimentos para dar un paso en el bfs, por ejemplo los obstáculos o los corrales según el caso. Con la matriz dada se elige entonces una próxima casilla para el robot que lo lleve al objetivo en menos pasos.

Indicaciones para su uso:

Para el proyecto se utilizó **stack**, por lo que para efectuar una simulación se debe ejecutar el comando *stack run* en la carpeta del codigo fuente. En el archivo *Main.hs* hay varios casos de prueba, se puede comentar y descomentar los que se desean o no correr, o añadir otros nuevos.

Una simulación dada imprimirá los siguientes detalles:

En cada iteración, o segundo simulado:

- Tiempo t trascurrido.
- Matriz resultante del movimiento natural y aleatorio del medio, o sea con el movimiento de los bebés y la suciedad
  generada repectivamente.
- Matriz tras la respuesta del robot.

Tras finalizar la simulación:

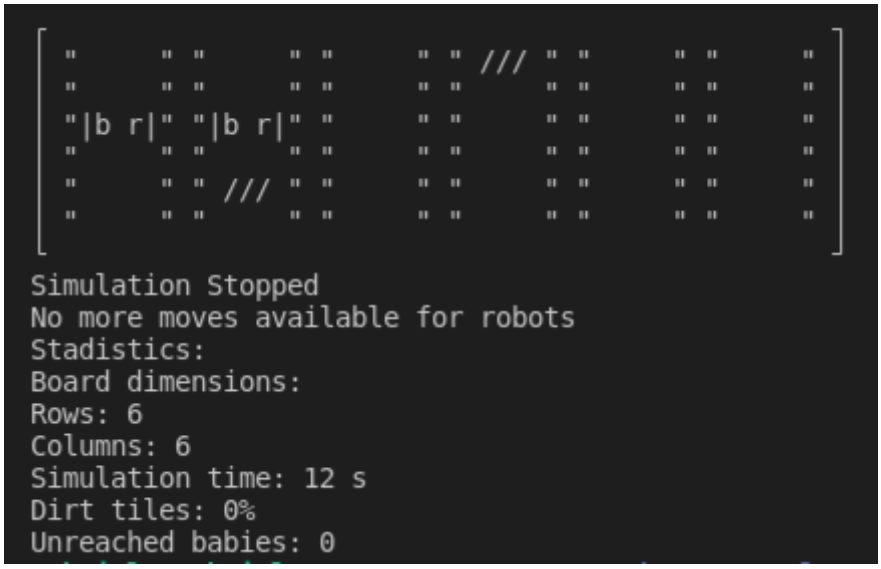
- Causa de la pausa realizada (tiempo límite alcanzado, objetivo no cumplido, etc).
- Dimensiones del tablero utilizado (filas y columnas).
- Tiempo transcurrido.
- Porciento de suciedad existente en el momento de la pausa.
- Número de bebés que no fue posible alcanzar.

Debe destacarse que en cada tablero se representan los elementos a partir de símbolos según la leyenda que se muestra a continuación:

- **r** : robot sólo
- **b** : bebé sólo
- **R** : robot con bebé en brazos
- **B** : bebé cargado
- **| |** : corral
- **///** : obstáculo

- **D** : suciedad

En la figura siguiente se muestra una captura del resultado final de una simulación sobre un tablero de 6x6 que inició con dos elementos de cada tipo.



Como se puede observar los robots cumplieron su objetivo, la casa quedó limpia de suciedad y los bebés terminaron en el corral.

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

Para comprobar la eficacia de los modelos de agentes implementados, se efectuaron numerosos test
con tableros de diferentes dimensiones y variando la cantidad de elementos de cada tipo.

Con simulaciones en tableros de 5X5, un límite de tiempo de 11 segundos y comenzando con dos elementos de cada tipo, los dos modelos de robots logran mantener el control de la situación, dejando siempre la casa totalmente limpia o con un nivel de suciedad no superior al 4%. De igual forma, los bebés son casi siempre alcanzados en este tiempo.

En tableros de 100 x 100 con 5 elementos de cada tipo se logra también alcanzar todos los bebés y limpiar toda la casa, lo que ya se va notando más la diferencia entre ambos agentes, puesto que mientras R2B2 demora un promedio de 212 segundos en culminar las tareas, C3PO tarda 263.

Por lo general ambos muestran un buen desempeño, pero el agente 2, C3PO, suele demorarse un poco más.
Esto es entendible, puesto que se centra en capturar a los bebés y no limpia hasta que no haya alcanzado este objetivo. Notemos que aunque su actuar puede resultar eficiente cuando tiene bebés mas cerca y los captura antes de que se alejen, a la vez da más tiempo a que los bebés generen más suciedad a medida que se mueven por la casa y evitan se capturados por el robot.

Ambos modelos podrían ser mejorados si se considera comunicar a los robots entre sí, de modo que se repartan las tareas y no todos caminen innecesariamente hacia un mismo objetivo.