Proyecto de Simulación y Programación Declarativa

Agentes-Haskell

Alejandro Campos. C-411

Facultad de Matemática y Computación
Universidad de la Habana
2021

I. Introducción

Un agente es un sistema computacional situado dentro de un medio ambiente, dentro del cual es capaz de realizar acciones autónomas encaminadas a lograr sus objetivos. Un agente inteligente es aquel que es capaz de ejecutar autónomas flexibles para lograr sus objetivos, donde flexible significa: reactivo, proactivo y sociable. Se define reactivo como que el agente es capaz de percibir su ambiente y responder de un modo oportuno a los cambios que ocurren para lograr sus metas. Proactivo quiere decir que debe ser capaz de mostrar un comportamiento dirigido a objetivos tomando la iniciativa para lograrlos. Por último, un agente sociable debe ser capaz de interactuar con otros agentes para lograr sus propósitos.

El problema de agentes en el cual se centra el proyecto se detalla a continuación:

El ambiente en el cual intervienen los agentes es discreto y tiene la forma de un rectángulo de N x M. El ambiente es de información completa, por tanto, todos los agentes conocen toda la información sobre el agente. El ambiente puede variar 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 el 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, empujándolos, por los niños, una única casilla. El Robot de Casa, sin embargo, no puede moverlo. No pueden ser movidos por 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 casillas 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 aparezca 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 ni no 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 adyacentes, las que decida. Solo se mueve una casilla si no carga un niño. Si carga un niño puede 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.

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

Para la implementación del problema anterior se usó como lenguaje de programación Haskell, que es un lenguaje de programación puramente funcional, estandarizado multi-propósito, con evaluación no estricta y memorizada, y fuerte tipificación estática. Los programas escritos en Haskell se representan siempre como funciones matemáticas, pero estas funciones nunca tienen efectos secundarios ni derivados. De este modo, cada función utilizada siempre devuelve el mismo resultado con la misma entrada, y el estado del programa nunca cambia. Por esto, el valor de una expresión o el resultado de una función dependen exclusivamente de los parámetros de entrada en el momento. En Haskell no pueden hacerse construcciones de lenguaje imperativo para programar una secuencia de declaraciones.

Por todo lo expuesto anteriormente, el objetivo de este informe es explicar el método de resolución usado para resolver el problema mencionado. Además, explicaremos los detalles de implementación y ejecución de la aplicación de consola que se ha creado, no sin antes abordar los distintos modelos de agentes utilizados y cómo se adaptaron a este problema en particular. Por último, haremos un análisis de los resultados obtenidos a partir de la ejecución de las simulaciones del problema.

II. Ideas principales para la solución del problema

Para tratar este problema se siguieron las indicaciones de las conferencias de la asignatura Simulación y del libro "Temas de simulación", capítulo 6, los cuales, a muy groso modo, plantean que el agente recibe estímulos de su medio ambiente y realiza acciones sobre este. En secciones posteriores formalizaremos esta idea. Primeramente, en esta sección, haremos una interpretación del problema en cuestión y expondremos las ideas seguidas para su solución.

El problema en cuestión nos plantea que el ambiente es dinámico, es decir, que cada cierto tiempo t puede variar aleatoriamente. Esta variación del ambiente viene dada por el movimiento de los niños, los cuales pueden moverse a una casilla adyacente, empujar obstáculos y ensuciar el ambiente. Esto implica que el agente que se diseñe para controlar el robot no puede ser proactivo, pues cualquier estrategia planteada probablemente quedaría estropeada antes de cumplirse. En los modelos proactivos se asume que el ambiente no va a cambiar mientras el procedimiento se ejecuta. Si el ambiente cambia y las precondiciones se vuelven falsas durante el proceso, entonces el comportamiento del procedimiento se indefine o simplemente falla. También se asume que, durante la ejecución, el objetivo sigue siendo válido hasta que este termine, pero si este deja de ser válido, no hay razón para seguir ejecutando el procedimiento. En nuestro problema si seguimos el objetivo de capturar un niño, este puede dejar de ser visible para los agentes porque, por ejemplo, otro niño interpuso un obstáculo en el camino.

En nuestro ambiente el agente tiene información completa y actualizada sobre el estado del medio. Además, su determinismo garantiza que cualquier acción tiene un único efecto, es decir, no hay incertidumbre sobre el estado en el que quedará el ambiente después de realizar una acción. En este problema, el agente debe decidir solamente en base al episodio actual, no tiene que razonar las consecuencias en episodios futuros, cualquier estrategia se verá condenada a fracasar en la mayoría de los casos por la aleatoriedad de cambio del ambiente.

En estos ambientes dinámicos, el agente debe ser reactivo, o bien una combinación entre reactivo y proactivo. Esto es, que debe ser sensitivo a los eventos que ocurran en el ambiente, donde estos eventos afectan los objetivos del agente, o las suposiciones en las que se basa el proceso que el agente está ejecutando, para lograr sus objetivos. Por otro lado, no queremos que nuestro agente esté constantemente reaccionando y, en consecuencia, nunca se enfoque en un objetivo el tiempo necesario para lograrlo. Es por ello que debemos construir sistemas que consigan un balance efectivo entre estos comportamientos.

Por lo tanto, las mejores estrategias tendrán que ser una combinación entre agente reactivo y proactivo, dado que el reactivo no podrá planificar a más de un paso y el proactivo verá sus planes atrofiados con el dinamismo del ambiente.

Finalmente, podemos afirmar que existen en nuestro ambiente un número fijo y finito de percepciones y acciones que lo pueden modificar. Ya sea por parte de los niños, como ya vimos, o por parte de los agentes, cuyas acciones y percepciones se definen en la siguiente sección.

III. Modelos de agentes considerados

Como vimos en la sección anterior, nuestro ambiente está definido por un conjunto de estados $E = \{s_1, s_2, s_3, ...\}$, que se generan producto de un cambio aleatorio dado por el movimiento de un niño (que genera suciedad o mueve obstáculos). También los agentes modifican el ambiente con las acciones que estos puedes realizar sobre él. Dado un estado s_i del ambiente, si aplicamos una acción del agente obtenemos un estado s_{i+1} producto de aplicar la acción a_i al estado s_i. Nuestros

agentes son capaces de realizar las siguientes acciones:

- Moverse a casillas adyacentes sin niño cargado (libre).
- Cargar niño.
- Moverse a casillas adyacentes, uno o dos pasos, con niño cargado.
- Dejar niño en cualquier casilla de las que puede moverse, menos en una suciedad.
- Limpiar una casilla sucia.

Un agente solo puede moverse a una casilla vacía, con corral vacío o con suciedad. También pueden moverse a casillas que contengan niños, si el agente no está cargando uno. Tampoco se permite que un agente pueda limpiar con un niño en la mano.

Los agentes captan la habilidad de observar el ambiente, por lo que se definen las siguientes percepciones que los agentes tendrán sobre el tablero:

- Camino al niño más cercano.
- Camino a la suciedad más cercana.
- Camino al corral más cercano.
- Por ciento de limpieza del ambiente.

Luego, sabemos que el conjunto de estados del ambiente son aleatorios, por lo que no podemos trazar una estrategia a largo plazo. Los agentes cada vez que van a realizar una acción deben revisar las percepciones basadas en el estado actual del ambiente para realizar una acción consecuente.

Definimos además en nuestros agentes dos estados, que cambian de acuerdo a las percepciones que el agente toma del ambiente:

- Cargando niño.
- Libre.

Las especificidades de cada modelo implementado se describen a continuación en las siguientes subsecciones:

I. Robot Random

En cada turno, este robot analiza cada una de las casillas a las que puede moverse, y selecciona su próximo paso aleatoriamente. Si no puede moverse a ninguna casilla, simplemente se queda en su lugar. Se tuvo en cuenta que si el robot se mueve a una casilla sucia, en el próximo turno la limpiará. Si, además, se mueve con un niño cargado, en el próximo turno lo continúa cargando. Podemos decir que este robot sigue un modelo reactivo, puesto que reacciona a los cambios del ambiente, ya que en cada turno calcula sus posibles movimientos, pero no tiene un objetivo definido.

El resto de los agentes que veremos se dice que son proactivos-reactivos porque aunque planean que su próximo movimiento es ir al niño más cercano, el corral más cercano o la casilla sucia más cercana, siguiendo un objetivo específico, el dinamismo del ambiente obliga a que la ruta a tomar pueda cambiar a medida que pasa el tiempo. Esto obliga a que en cada instante de tiempo se recalcule la ruta a tomar por el robot.

II. Robot niñera

La prioridad de este robot es capturar niños y meterlos en el corral. Se basa en la idea de que una vez todos los niños sean capturados, ya sea porque estén en el corral o cargados, no se ensuciará más el ambiente, y a partir de ahí solo quedaría limpiar. Si este robot se encuentra en estado libre, en su turno intentará buscar el niño más cercano alcanzable, si existe, entonces se moverá a aquella casilla que lo acerque más al niño, con el fin de capturarlo. Una vez se mueva a la casilla donde está el niño, inmediatamente lo captura y pasa al estado de "cargando niño". Cuando está en este estado, busca el corral más cercano alcanzable y se mueve una casilla o dos (si puede), acercándose más a este. Si está en el corral, busca la mejor forma de acomodar el niño, evitando, en cierta medida, que haya "estancamientos" de niños en el corral. Cuando encuentra la casilla del corral donde dejar al niño, lo deja y pasa al estado "libre" nuevamente.

En caso de que, cuando esté buscando niño, no encuentre ninguno alcanzable, lo que hace es buscar suciedades (las más cercanas) y limpiarlas. Si tampoco tiene suciedades alcanzables, se mueve aleatoriamente como un robot random. Por último, si este agente se encuentra en el estado de "cargando niño", y no ve ningún corral alcanzable, se mueve aleatoriamente con el niño cargado.

La estrategia de este robot es un tanto arriesgada, puesto que el ambiente puede ensuciarse más de lo permitido antes de que se capturen todos los niños. La victoria de este robot depende, en parte, de la cantidad de veces que el ambiente cambia en una simulación.

III. Robot limpiador

Como su nombre lo indica, el objetivo de este robot es mantener el ambiente limpio. El agente limpiador trata de prolongar el mayor tiempo posible la derrota por suciedad en el ambiente. Como su prioridad es limpiar, en cada turno este robot busca la suciedad más cercana, y se mueve con el objetivo de acercarse más a esta para limpiarla. Una vez llega a la casilla sucia, en el próximo turno, la limpia. Si no encuentra suciedad alcanzable, se pone a capturar niños para llevarlos al corral. En caso de que esté cargando niños, si ocurre el cambio de ambiente y se genera una nueva suciedad alcanzable, en el próximo turno deja al niño (pasa al estado "libre"), y posteriormente se mueve con el objetivo de limpiarla. En los casos en que esté cargando niño y no vea suciedad ni corral alcanzable, o esté libre y no vea niños ni suciedad alcanzable, se mueve aleatoriamente por el ambiente, manteniendo su estado. Las probabilidades de victoria de este robot son generalmente bajas, y mayormente dependen de la cantidad de niños que hay en el ambiente.

IV. Robot balanceado o equilibrado

Este tipo de agente es más inteligente que el resto. Pues basa sus decisiones de acuerdo a lo que sea mejor realizar en el momento en que le toque moverse. Antes de ejecutar su movimiento, este robot analiza el por ciento de limpieza del ambiente. Sabemos, según la orden del problema, que el por ciento de casillas limpias debe ser de al menos el 60% del total de casillas vacías. Si este valor baja del 70% quiere decir que el ambiente está cerca de ensuciarse provocando la derrota. Por ello, cuando esto sucede, este robot se pone a limpiar el ambiente tratando de aumentar las casillas limpias y poder, posteriormente, capturar niños y llevarlos al corral, con la tranquilidad de que durante ese proceso el ambiente no debe ensuciarse a tal punto que provoque que no se cumpla el objetivo. Entonces, en síntesis, el robot balanceado se comporta como el robot limpiador, si el ambiente está muy sucio. En caso contrario, se comporta como el robot niñera. Por ello, si en el turno de moverse, el ambiente está muy sucio y el robot no tiene suciedades alcanzables, pues

captura niños, si no puede hacer ninguna de las dos, se mueve aleatorio. Lo mismo ocurre si es el momento de coger niños y no tiene ninguno alcanzable: busca suciedades, si tampoco encuentra, se comporta como un robot random.

V. Robot de trabajo en equipo

Este robot funciona, por supuesto, cuando hay al menos dos robots en el ambiente. No podemos decir que son sociables, ya que no existe comunicación entre ellos. Sin embargo, en cierto modo, el trabajo en equipo y la cooperación para lograr sus objetivos los hace que cumplan algunas características de este tipo de agentes. Nuestros agentes se dividen el trabajo para garantizar la victoria en el menor tiempo posible. En lugar de que todos los agentes capturen niños o limpien, lo que hacemos es darle a unos robots la tarea de limpiar la casa, y a otros la tarea de capturar niños. De esta forma los que capturan niños pueden trabajar tranquilos sabiendo que sus compañeros evitarán que el ambiente se ensucie. Si algún robot de los que se les encargó la tarea de limpiar no encuentra suciedad, ayuda a los que capturan niños y viceversa, si los robots que capturan niños no encuentran niño, ayudan a limpiar.

IV. Detalles de implementación

La implementación de la simulación del problema en cuestión se basa en las ideas anteriores. Nos dedicaremos en esta sección a explicar la funcionalidad de cada módulo que interviene en el proyecto. Abordando aquellas funciones que consideremos importantes. Cabe destacar que en los comentarios del código se expresa lo que realiza cada función implementada.

I. `Functions.hs`

Este módulo contiene en su mayoría funciones para pintar el ambiente en consola y tener una visualización de lo que está pasando. Contiene otras funciones útiles que el resto de los módulos utilizan como *cleanPercent* que calcula el por ciento de casillas limpias en el ambiente. Si definimos l = cantidad de casillas limpias y s = cantidad de casillas sucias, entonces el porciento de las casillas vacías que no están sucias es $\frac{l}{l+s} * 100$, considerando como casillas vacías, aquellas que no contienen niño, robot, corral u obstáculo.

II. `Objects.hs`

En este módulo se define el tipo *Object* que contiene el nombre del objeto ya sea Obstáculo, Suciedad, etc. En el caso de los robots su nombre viene dado por "Robot <state>", donde state es el estado en que se encuentra. Este tipo contiene, además, la localización de cada objeto: posición x y posición y.

Se implementan en este archivo funciones útiles para el trabajo con los objetos del ambiente, como las direcciones de las casillas adyacentes a cada objeto: arriba, abajo, izquierda, derecha. Además de funciones para ubicar objetos de forma aleatoria, desplazarlos, obtener sus posiciones, etc.

III. `Environment.hs`

Aquí se encuentra *changeEnvironment*, que es la encargada de simular el cambio aleatorio del ambiente, lo cual se simula de la siguiente forma:

Tomamos cada uno de los niños presentes en el ambiente y se verifica si se pueden mover. Un niño puede moverse si no está en el corral o en un robot, y si en la dirección en que se va a mover hay una casilla libre o una sucesión, de uno o más obstáculos, seguidos por una casilla libre. En otro caso, el niño no puede moverse. Primeramente, se selecciona una dirección aleatoria, si esta es válida (está dentro del tablero) y cumple las condiciones anteriores, entonces el niño se mueve. En caso de que haya obstáculos, los desplaza.

En este proceso el niño también puede ensuciar. Para ensuciar, de todas las cuadrículas posibles válidas de 3x3 (máximo 9), se selecciona una aleatoria y se ensucia de acuerdo a como se plantea en el problema, atendiendo a la cantidad de niños que están en la cuadrícula. Entiéndase por cuadrícula válida, aquella que contiene al niño y está completamente contenida en el tablero.

El resto de las funciones implementadas en este módulo son funciones auxiliares del método antes explicado, además de otras para el trabajo con las posiciones del ambiente.

IV. `UtilsRobots.hs`

En este archivo se encuentran funciones que ayudan a la implementación de cada tipo de robot. Tenemos dos funciones fundamentales que rigen los movimientos de robots: *canRobotMove* y *bfs*. La primera define a qué casillas un robot puede moverse, tal y como se dijo en la sección anterior. La segunda devuelve el objeto, que le pasamos como parámetro, más cercano, rigiéndose por las casillas válidas definidas en la primera. Devuelve además una lista que representa el árbol del bfs, a partir del cual calculamos el camino del robot al objeto, mediante la función *getPathToMoveRobot*.

El resto de funciones que se definen en este archivo son para simular las acciones que puede hacer el robot, las cuales se definen al inicio de la sección III.

V. `Robots.hs`

Este módulo contiene la implementación de cada uno de los modelos descritos en la sección III, tal y como se describieron. En síntesis, cada una de estas funciones aplican bfs para encontrar las mejores casillas a las que moverse dependiendo de lo que buscan, luego ejecutan alguna de las acciones implementadas en el módulo que explicamos anteriormente según el objetivo que persiguen.

VI. `Simulation.hs`

Este módulo contiene dos funciones. *createEnvironment* es el encargado de crear el ambiente inicial con los parámetros del usuario. Ubica la cantidad especificada de cada objeto en una casilla aleatoria del ambiente, se auxilia de algunas funciones definidas en *Objects.hs*. En el caso del corral, este se ubica en casillas aleatorias adyacentes atendiendo a la cantidad de niños, haciendo dfs a partir de una casilla seleccionada de forma al azar del tablero. Las direcciones para moverse en el dfs también son seleccionadas al azar en cada momento. *createEnvironment* recibe la cantidad de simulaciones a realizar, y devuelve los resultados una vez se hayan completado todas. De estos resultados hablaremos en la siguiente sección. También, recibe como parámetro el tipo de robot con el que se van a ejecutar las simulaciones y llama a la simulación con la función correspondiente al modelo de agente seleccionado por el usuario.

La función *simulation* es la encargada de controlar el flujo de una simulación, recibe el ambiente inicial creado con la función que explicamos anteriormente. Como la orden del problema indica, la simulación ocurre por turnos, en un turno todos los agentes realizan sus acciones, y esto constituye una unidad de tiempo. Entonces, la simulación ejecuta las acciones de cada uno de los agentes presentes en el ambiente, y cuando termina el último, ocurre el cambio de turno. En este momento el ambiente puede variar, y se considera parte del mismo turno, como plantea el problema. El ambiente cambia cada t unidades de tiempo o turnos, es decir, que si $t = 3$ cada agente debe ejecutar acciones tres veces antes de que el ambiente cambie de nuevo. Por tanto, un turno está formado por las r iteraciones de los agentes (siendo $r =$ número de agentes), más un posible cambio aleatorio del ambiente.

La simulación tiene tres casos de parada. El primero de ellos es cuando se declara derrota. Esto sucede si el por ciento de limpieza del ambiente cae por debajo del 60% y el número de turnos es mayor que 5. Se le agrega esta última condición por si el usuario define un número de suciedades que incumplan el objetivo, no se declare derrota de inmediato, y para darle oportunidad a los robots de revertir esta situación al inicio.

El segundo caso de parada es cuando se superan los 150 turnos y no se ha perdido. Es decir, en el caso peor, el ambiente pudo cambiar 150 veces y no provocó la derrota de los robots. Por lo tanto, son capaces de mantener el ambiente limpio en estas condiciones. Sin embargo, no son capaces de coger todos los niños (debido a que están rodeados por obstáculos, porque se encuentra "estancado" el corral, porque necesitan más turnos etc) y, en iteraciones futuras, podrían perder. Pero, para evitar que en estos casos el programa se demore demasiado, entonces se declara victoria con número de turnos excedido.

La última condición de parada de una simulación es la de victoria. Se dice que ocurre esto cuando todos los niños están en el corral o sobre robots. Si esto pasa, se cumple el objetivo de tener el ambiente limpio y ningún niño podrá moverse ni ensuciar más, puesto que están todos capturados.

VII. `Main.hs`

Finalmente, el módulo *Main* es el principal de la aplicación. Es el que se llama para comenzar la ejecución. Contiene la presentación del proyecto, y la función encargada de recibir los datos por parte del usuario. Si estos valores no son consistentes se genera una excepción que se captura con las funciones *catchExceptions* y *handler*.

Uno de los parámetros de entrada que recoge la función *getInput* es el modo debug, que si está activo, muestra iteración por iteración de agentes, y debemos ir presionando *enter* para pasar a la siguiente acción. Si es el momento de cambiar el ambiente, se muestran el cambio de ambiente y el movimiento del robot en la misma iteración.

V. Ejecución de la aplicación

Para ejecutar la aplicación es necesario que el usuario tenga instalado algún intérprete de haskell que, a su vez, tenga instalado `System.Random`. En nuestro caso hemos usado stack.

Para ejecutar el proyecto debe escribir los siguientes comandos desde el directorio del proyecto:

```
> stack ghci
> :l Main.hs
> main
```

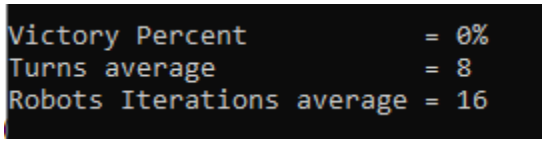
La aplicación pedirá al usuario que ingrese las dimensiones del tablero que constituirá el ambiente. Posteriormente, pedirá las distintas cantidades de objetos que estarán en el ambiente. Debe ingresar también cada qué tiempo desea que el ambiente cambie. Luego, debe seleccionar el tipo de agente que trabajará en el ambiente, escribiendo el número que corresponda al tipo que desea simular. Pueden realizarse más de una simulación con los mismos parámetros, este dato también debe ingresarlo. Por último, nuestra aplicación muestra si se ejecuta en modo debug o no. En caso de que el usuario ponga más objetos de los que soporta el ambiente, o ingrese algún dato incongruente, la aplicación notifica que hubo un problema y debe ingresar los datos nuevamente, de forma correcta.

Al finalizar cada simulación se muestra si el robot logró el objetivo o no (si fue victoria o derrota). Además, nos devuelven algunos datos de interés, como el número de turnos que se desarrollaron en la simulación, y la suma total de iteraciones de todos los robots. Al finalizar todas las simulaciones, nos muestra el por ciento de victorias de estas, y el promedio del total de turnos e iteraciones de robots que se ejecutaron en cada una.

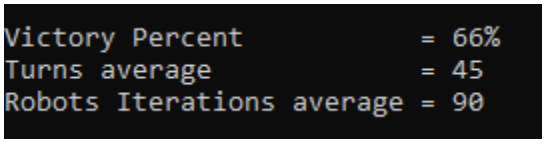
VI. Consideraciones obtenidas a partir de la ejecución de las simulaciones del problema

Para analizar la eficiencia de los robots realizaremos varias simulaciones con cada uno, con los mismos parámetros, y expondremos los resultados a fin de compararlos.

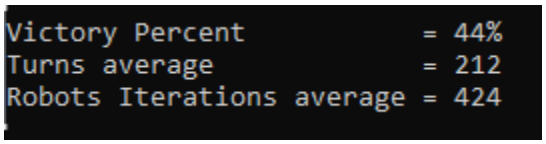
Se ejecutaron 30 simulaciones con los siguientes parámetros: ambiente de dimensión 6x7 con 2 robots, 5 niños, 7 casillas sucias, 6 obstáculos y con valor $t = 2$ de cambio de ambiente, es decir, que varía cada dos turnos. Cabe destacar que, para recopilar los resultados de las simulaciones, se modificó el código para que permitiera hasta 300 turnos para declarar victoria por exceso de turnos, y así dar resultados más precisos. En la aplicación que se entrega al usuario no se permiten más de 150, como ya se explicó anteriormente.



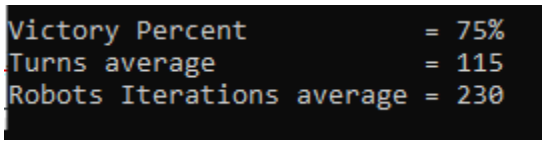
Resultados para el robot random



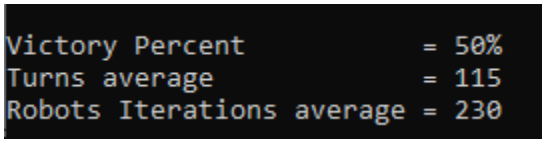
Resultados para el robot niñera



Resultados para el robot limpiador



Resultados para el robot balanceado o equilibrado



Resultados para el robot de trabajo en equipo

Los resultados para el robot random son los que esperábamos. En ninguna simulación logró mantener el ambiente limpio y capturar los niños. En un promedio de solo 8 turnos ya tenía el ambiente demasiado sucio. Dado que este tipo de robot únicamente se mueve a casillas válidas que escoge al azar y limpia "de casualidad", estos resultados son más que consistentes.

El robot niñera obtuvo uno de los más altos por cientos de victorias. En la mayoría de los casos, la variación del ambiente le permitió capturar a todos los niños antes de que se ensuciara más de lo debido.

El robot limpiador fue el que obtuvo más bajo promedio, pero fue el que más turnos demoró en detener la simulación. El tener que dedicarse a limpiar como prioridad y no capturar los niños hace que estos sigan ensuciando hasta caer por debajo del límite de limpieza. Sin embargo, en algunos casos, esta estrategia fue exitosa, ya que en estos los niños no ensuciaban demasiado en el momento de cambio de ambiente, y el robot pudo capturarlos.

El robot balanceado fue el que más alto porcentaje tuvo. Viene dado por su estrategia de hacer lo mejor en el momento que se va a mover. Logró una eficacia del 75%. Sin embargo, hubo casos de fracaso. Lo cual puede venir dado por la cantidad de suciedad nueva que se genera en los cambios aleatorios y el movimiento de los niños.

Finalmente, el robot de trabajo en equipo, con una eficacia promedio del 50%, logró su objetivo en la mitad de los casos. Su promedio no tan alto puede venir dado a que eran solo dos robots en el ambiente. En ambientes donde se permitan más robots, el número de cada "equipo" aumenta, haciendo más eficaz su estrategia.

Podemos concluir que, menos del robot random, cada estrategia es válida dependiendo de los cambios del ambiente. Como estos ocurren aleatoriamente, no podemos predecir cuándo y dónde un niño va a ensuciar, a dónde se va a mover etc, por lo que la estrategia funciona si, casualmente, el ambiente es el "idóneo" para esta. El robot balanceado demostró ser el más eficaz. Pero también existen casos donde su estrategia falla. Por otro lado, el robot de trabajo en equipo, podría ser tan eficiente como el balanceado si existen más robots en el ambiente.

VII. Github

En este enlace podemos encontrar el repositorio del proyecto en Github.