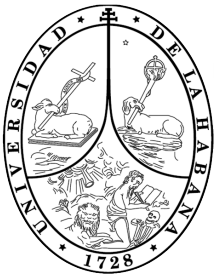


# Proyecto de Simulación
## *"Agentes"*

***Est.*** **Lázaro Raúl Iglesias Vera** L.IGLESIAS@ESTUDIANTES.MATCOM.UH.CU

*C-412*

# Índice

# 1 Introducción

En este proyecto se mostrarán implementaciones del sistema de agentes mostrado a continuación.

## 1.1 Orientación. Resumen básicos del sistema

El ambiente en el cual intervienen los agentes es discreto y tiene la forma de un rectángulo de $N \times M$. El ambiente es de información completa, por tanto todos los agentes conocen toda la información sobre el ambiente. 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.

## 1.2 Detalles de los elementos

La definición detallada de cada elemento del ambiente puede encontrarse en [16].

# 2 Desarrollo

## 2.1 Ambiente

Para poder simular el sistema se creó un ambiente [2], el cual se encarga del control de los turnos de los agentes, de sus propias variaciones aleatorias y de verificar los estados finales de la simulación.

De la misma manera fueron implementados todos los elementos simples que conforman el ambiente, dígase suciedad [3], obstáculos [4], corrales [6] y bebés [5].

## 2.2 Ideas de solución

Los robots de casa necesitan hacer su trabajo de la mejor manera posible para evitar ser despedidos. Para intentar lograr un buen desempeño en estos agentes se valoraron las siguientes ideas:

1. **Siempre, dado un objetivo, ir al elemento más cercano**
   Si el robot desea ir a recoger un bebé, limpiar una basura, o llevar el bebé que carga a la cuna, siempre ir a el bebé, la basura o la casilla más cercana del corral respectivamente.

2. **Dar prioridad a la tarea de recoger los bebés**
   La suciedad es únicamente generada por los bebés, por tanto tener un mejor control de los bebés reduce la aparición de suciedad.

3. **Dar prioridad a la tarea de recoger basura en caso de despido inminente**
   Si el porciento de casillas sucias se hace cercano al límite por el cual el robot sería despedido (60 %), intentar disminuir ese valor realizando la tarea de limpieza.

4. **Siempre limpiar las casillas sucias a las que se llegue**
   Aunque la planificación del robot al llegar a una casilla sucia sea otra diferente a la tarea de limpieza poder, en caso de pasar por una casilla sucia detenerse a limpiarla, dado que de no hacerse, más adelante serán gastados almenos 2 turnos extras para volver a alcanzar la casilla y otro para limpiarla.

5. **Crear un mecanismo de percepción para detectar el período de cambio aleatorio del ambiente** Conocer cada qué tiempo ocurren los cambios aleatorios del ambiente le sería de gran utilidad al robot dado que le permite crear planificaciones realistas, si el ambiente cambia cada 10 turnos ninguna planificación que requiera más de 10 turnos en su ejecución será finalizada correctamente. Por tanto tener dominio de esta información permite evitar hacer planes que no podrán ser completados.

De todas las ideas mencionadas la única no utilizada en la implementación brindada es la idea #3, dado que en casos de despido inminente seguir contolando los bebés es también una posible estrategia para evitar sobrepasar el límite de suciedad.

## 2.3 Modelos

### 2.3.1 Generales

Los dos modelos de robot de casa analizados en este proyecto son el puramente reactivo y el de razonamiento práctico. Ambos modelos comparten la manera de percibir el ambiente [7] y el mecanismo de decisión de las acciones a realizar se basa en el mismo sistema de prioridades o capas.

### 2.3.2 Puramente Reactivo

Este robot reactivo [8], en cada turno evalúa qué acción realizar independientemente de sus acciones previas. Su decisión en cada turno consiste en hallar la primera capa [9] tal que su condición o precedente sea válido y ejecutar la acción que lo acerque más al cumplimiento del objetivo de esa dicha capa.

### 2.3.3 Razonamiento práctico

Este robot tiene un comportamiento proactivo [10]. Inicialmente necesita definir una intención para lo cual utiliza las capas [11] antes mencionadas. Una vez seleccionada la capa queda implícita una intención, la cual va a planificar, llevar a cabo e intentar completar en los turnos siguientes. Dicha planificación abarca todo el proceso de ejecución de la tarea y solo será abandonada en casos en los cuales se detecte una irregularidad en la planificación, como puede ser un movimiento a una casilla que ya no es adyacente, que puede ser provocado por una variación aleatoria del ambiente.

## 2.4 Implementación

Para la implementación de todo el sistema se creó un módulo source(src) que consta de 2 submódulos principales. Estos serán discutidos a continuación.

### 2.4.1 Agents

Este módulo contiene la implementación de todos los elementos del ambiente. Con la intención de crear un convenio para los mismos, todos fueron implementados a partir de la clase base que representa a un agente [12]. Dicha clase define los diferentes métodos a implentar por los elementos del ambiente, entre los cuales se encuentran: *push* para situaciones en las que el elemento esta siendo empujado, *action* para definir la interacción con el ambiente, entre otros.

Una de las propiedades más importantes de los agentes implementados es su *tag* [13] [14]. Este le servirá como identificador a cualquier otro agente que intente interactuar con él. Ejemplo de esta situación es el *action* de un bebé, el cual, al decidir hacer un movimiento, trata de empujar al elemento que esté en esa dirección; este movimiento solo se realizará si el *push* del elemento empujado resulta satisfactorio al interactuar con un bebé.

La implementación de cada elemento no será explicada dada la sencillez de sus funcionalidades, a continuación se hablará sobre los robots solamente.

Para los robot se creó una clase más específica que ampliara el comportamiento de un agente común [7]. Dicha clase añade las funcionalidades *see* para obtener los datos del ambiente y *perceive* para procesarlos.

Este nuevo método *perceive* es el encargado de detectar el período de cambio del ambiente, lo cual se consigue detectando cambios inesperados en la posición del robot. De esta manera registra cuánto tarda en ocurrir el cambio y logra, mediante la minimización del valor obtenido, converger al tiempo real de cambio del ambiente. La otra funcionalidad de este este método, y no menos importante, es la

de definir el camino hacia las celdas alcanzables a una distancia menor que el actual tiempo de cambio registrado.

Teniendo esta información los robots en cada una de sus capas encuentran el elemento más cercano que representa su objetivo y deciden/trazan su próxima acción/planificación en el caso de el robot reactivo/proactivo.

### 2.4.2 Environment

Este módulo define la estructura y el comportamiento de un ambiente [2]. La implementación del ambiente principal utilizado puede ser encontrada en [15].

La estructura básica provee a los ambientes de una función *run* la cual se encarga de manejar los turnos y ejecutar el *action* de cada agente. Además, mediante la implementación de su método *stop* y *variate* se permite el manejo y control de los estados finales del sistema, los cuales son verificados al final de cada turno, y la ejecución de las variaciones aleatorias del mismo.

Dicho esto, solo resta mencionar que cualquier ambiente implementado solo debe definir su estructura(tabla de posiciones), su manera de variar, y cómo se modifican y obtienen los datos del mismo (*set*, *unset*, *get*), así como sus estados finales; siempre permitiendo modificar, en caso de ser necesario, cualquier método definido en la clase base, como es el caso de *run*.

### 2.4.3 Aplicación

La aplicación brindada permite ejecutar simulaciones y escoger entre los robots y ambientes implementados cuáles serán los utilizados. También permite fácilmente añadir nuevas implementaciones al conjunto de opciones. Para leer al respecto puede utilizar el readme del proyecto [16].

## 2.5 Experimentos

### 2.5.1 Ambientes a simular

Para calcular los resultados de cada ambiente fueron ejecutadas 100 repeticiones de cada uno, tomando como tiempo de finalización de la simulación $t * 100$, como se sugiere en la orientación del proyecto.

| Test Id | Filas | Columnas | Bebés | Tiempo(t) | Obstaculos | Suciedad |
|---|---|---|---|---|---|---|
| t0 | 10 | 10 | 6 | 2 | 20 | 40 |
| t1 | 10 | 10 | 6 | 3 | 20 | 40 |
| t2 | 10 | 10 | 6 | 4 | 20 | 40 |
| t3 | 10 | 10 | 6 | 5 | 20 | 40 |
| t4 | 10 | 10 | 6 | 10 | 20 | 40 |
| t5 | 10 | 10 | 6 | 15 | 20 | 40 |
| t6 | 7 | 8 | 7 | 5 | 10 | 10 |
| t7 | 7 | 8 | 7 | 5 | 14 | 30 |
| t8 | 20 | 20 | 16 | 15 | 0 | 45 |
| t9 | 20 | 20 | 16 | 15 | 30 | 0 |
| t10 | 20 | 20 | 5 | 15 | 0 | 0 |
| t11 | 7 | 8 | 6 | 2 | 20 | 40 |

### 2.5.2 Resultados

| Test Id | Tipo de robot | Suciedad media | Ambiente limpio | Tiempo terminado | Despedido |
|---|---|---|---|---|---|
| t0 | reactive | 60.46 | 0 | 0 | 100 |
| | proactive | 60.06 | 0 | 1 | 99 |
| t1 | reactive | 59.11 | 2 | 0 | 98 |
| | proactive | 58.03 | 0 | 4 | 96 |
| t2 | reactive | 50.07 | 16 | 1 | 83 |
| | proactive | 57.25 | 5 | 0 | 95 |
| t3 | reactive | 48.82 | 19 | 0 | 81 |
| | proactive | 45.2 | 25 | 0 | 75 |
| t4 | reactive | 11.44 | 81 | 0 | 19 |
| | proactive | 9.03 | 85 | 0 | 15 |
| t5 | reactive | 1.2 | 98 | 0 | 2 |
| | proactive | 0.61 | 99 | 0 | 1 |
| t6 | reactive | 31.47 | 8 | 0 | 92 |
| | proactive | 32.9 | 4 | 0 | 96 |
| t7 | reactive | 29.37 | 14 | 0 | 86 |
| | proactive | 29.39 | 14 | 0 | 86 |
| t8 | reactive | 242.67 | 0 | 0 | 100 |
| | proactive | 242.4 | 0 | 0 | 100 |
| t9 | reactive | 4.82 | 98 | 0 | 2 |
| | proactive | 0.0 | 100 | 0 | 0 |
| t10 | reactive | 0.0 | 100 | 0 | 0 |
| | proactive | 0.0 | 100 | 0 | 0 |
| t11 | reactive | 23.81 | 14 | 17 | 69 |
| | proactive | 22.72 | 19 | 16 | 65 |

# 3 Conclusiones

Un dato importante a tener en cuenta que se hace notable en las simulaciones realizadas es la dependencia directa que existe entre la efectividad del robot en la realización de sus tareas y el tiempo que demora en ocurrir un evento aleatorio en el ambiente, dado que no solo afecta los planes previos que haya trazado, sino que también puede hacer que nunca se logre realizar alguna tarea específica, por ejemplo, con $t = 5$, si el robot tiene a un bebé cargado y nunca, tras las variaciones aleatorias, queda a distancia menor o igual que 10 de un corral, no podrá terminar de limpiar el ambiente.

Dicho lo anterior se puede apreciar en los resultados de las simulaciones que en aquellas con bajos $t$ predominan los despidos del robot. Con el aumento gradual del tiempo los resultados comienzan a ser más variados.

Se pudiera pensar que el robot reactivo siempre obtiene los mejores resultados dado que en cada turno decide efectuar la mejor tarea, pero los estudios realizados muestran que esto no es cierto. En algunos casos se observa que el proactivo logra mejores índices de limpieza. Este comportamiento surge debido a que el reactivo puede cambiar tantas veces de tarea que termine sin completar ninguna.

También es apreciable que las condiciones del ambiente tienen gran influencia sobre el desempeño pues en ambientes con grandes espacios vacíos los niños se mueven con mayor libertad y tienen mayores posibilidades de ensuciar. Incluso la existencia de cierto de nivel de suciedad al inicio de la simulación pudiera resultar favorable para el robot, dado que, de igual manera, se reduce la movilidad de los niños.

Durante el desarrollo del proyecto fueron valoradas distintas estrategias que también contribuyeron a llegar a los resultados obtenidos y sobre todo, hicieron notar aspectos importantes a tener en cuenta al crear un agente, como son:

1. **Hallar el orden o el valor de las tareas que sea más favorable**
   En las implementaciones iniciales se intentó dar prioridad a la limpieza de la suciedad sobre la recogida de niños. Esto no resultó ser favorable, incluso en casos donde se estaba cercano a ser despedido. Dar prioridad a esta tarea resultó ser una mala decisión.

2. **Aprovechar al máximo la información del ambiente**
   Al comenzar a trabajar con el tiempo $t$ los resultados cambiaron considerablemente, dada la posibilidad de lograr una planificación más realista.

3. **Hallar el balance correcto entre reactividad y proactividad**
   Si se conoce información sobre el comportamiento del ambiente, se puede decidir si ser más reactivo para ambientes muy variables, o más proactivo en ambientes con poca variabilidad, donde los planes a largo plazo puedan ser finalizados correctamente con seguridad.

# Referencias

[1] Orientación oficial del proyecto (ir)

[2] Ambiente básico (ir)

[3] Implementación de la suciedad (ir)

[4] Implementación de los obstáculos (ir)

[5] Implementación de los bebés (ir)

[6] Implementación de los corrales (ir)

[7] Robot básico / Percepción del ambiente (ir)

[8] Robot Reactivo (ir)

[9] Robot Reactivo Capas de decisión (ir)

[10] Robot Proactivo (ir)

[11] Robot Proactivo Capas de decisión (ir)

[12] Agente básico (ir)

[13] Definición de los tags (ir)

[14] Función para maejar los tags (ir)

[15] Implementación del Ambiente (ir)

[16] Proyecto en GitHub (ir)