



Tecnológico de Monterrey

Campus Santa Fe

MA. Actividad: Roomba

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 302)

Alumna:

Valentina Castilla Melgoza

Profesor:

Octavio Navarro Hinojosa

Sábado 22 de octubre del 2025

Se busca desarrollar un sistema de multiagentes que simulen el comportamiento de “roombas” y logren limpiar un cuarto con basura y obstáculos en un tiempo determinado (1000 steps). Los roomba tienen que evitar que se les acabe la batería, ya que si esto ocurre dejan de funcionar y no pueden cumplir su objetivo de limpieza.

Simulación 1:

La solución propuesta consiste en un ambiente donde existen cuatro tipos de agentes: roombas, obstáculos, basura y estaciones de carga. Los roombas recorren el tablero siguiendo un patrón de zigzag, perciben sus celdas vecinas para detectar basura y obstáculos, se mueven hacia las celdas sucias para limpiarlas y si encuentran un obstáculo, lo esquivan moviéndose a una celda libre e intentando regresar a su fila anterior. Además, monitorean constantemente su nivel de batería y cuando este es menor a un valor definido regresan a su estación de carga, donde se recargan hasta llegar al 100 por ciento para después volver a la posición que habían guardado y continuar limpiando. Los obstáculos bloquean el paso, la basura permanece en su lugar hasta ser limpiada por un roomba y las estaciones de carga permiten recuperar batería a los roombas, mientras que el sistema recolecta estadísticas como la cantidad de roombas activos y la cantidad de basura recolectada.

Simulación 2:

La simulación con varios agentes tiene en esencia el mismo comportamiento que las simulación con un solo agente. Los agentes estáticos se comportan igual, los agentes de tipo roomba son los que sí tienen un comportamiento de movimiento diferente, las demás acciones de este agente son iguales. La diferencia de comportamiento de movimiento de los roombas es que, para la primera simulación como solo es un agente y empieza en la misma posición, para esta la dirección vertical siempre es para arriba. En el caso de varios agentes para optimizar la recolección de basura los agentes que empiezan en la mitad inferior comienzan con una dirección vertical hacia abajo y los que están en la mitad superior hacia arriba. Y para limpiar con mejor eficiencia las celdas de en medio los roombas que aparezcan en las celdas de la mitad empiezan con una dirección vertical aleatoria, es decir pueden hacer el zigzag hacia arriba o hacia abajo, lo que permite que recorran el tablero más rápidamente. Por último, otra acción extra es que cuando

llegan al borde superior o inferior del tablero cambian de dirección vertical, por lo que empiezan el recorrido hacia el otro lado.

En el sistema existen 4 tipos de agentes:

1. El primero son los “**roombas**”, su objetivo es recorrer el ambiente y encontrar celdas sucias para limpiarlas. Este tipo de agentes se puede mover, pueden limpiar celdas sucias y se deben cargar para que no se les acabe su batería. Para lograr su meta de recoger toda la basura, el roomba recorre el ambiente en un patrón de zigzag, en cada paso escanea a las celdas vecinas, si una de estas contiene basura se mueve a esa posición y regresa. Si encuentra un obstáculo en el camino se mueve a una celda libre, avanza e intenta regresar a la fila donde estaba antes de moverse por el obstáculo. En cada paso el agente revisa su nivel de batería y si es menor a un valor establecido regresa a una celda de carga, guardando su posición actual y se queda ahí hasta llegar al 100%. Cuando ya está cargada regresa a la posición que guardó y continúa su comportamiento de limpieza. En la primera simulación este agente no interactúa con más agentes de su mismo tipo, por lo que no tienen habilidad social. En la segunda simulación hay más agentes de ese tipo, por lo que si se encuentra con otro, le comunica dónde está su estación de carga, así el agente cuando necesite cargarse, puede ir a la más cercana.
2. Otro tipo de agente son los **obstáculos**, su objetivo es impedir que los agentes de tipo “roomba” pasen por encima de su celda. En cada paso los obstáculos se quedan en el mismo lugar. No son agentes proactivos pues no cambian de comportamiento dependiendo de su objetivo. Igual no son reactivos, no perciben su entorno por lo que no actúa de acuerdo a este. Y no tiene habilidad social porque no se comunica con otros agentes para completar sus objetivos.
3. Otro tipo de agente presente en el ambiente es la **basura**, y no tiene un objetivo establecido. En cada paso la basura se queda en el mismo lugar y se elimina si en la misma celda hay un agente de tipo roomba. No son agentes proactivos pues no cambian de comportamiento dependiendo de su objetivo. Pero son reactivos, pues en cada paso analiza a los agentes de su celda si encuentra a uno de tipo roomba se elimina. Por último

no tiene habilidad social porque no se comunica con otros agentes para completar sus objetivos.

4. Por último tenemos a los agentes que son **estaciones de carga**, y su objetivo es darle más batería al agente de tipo roomba para que pueda seguir limpiando. En cada paso la estación de carga se queda en el mismo lugar. No son agentes proactivos pues no cambian de comportamiento dependiendo de su objetivo. Tampoco son reactivos, pues no perciben su entorno por lo que no actúan de acuerdo a este. Por último no tiene habilidad social porque no se comunica con otros agentes para completar sus objetivos.

La arquitectura de subsunción de los agentes es la siguiente:

- **Performance:**
 - Roombas - Este tipo de agentes se puede mover, esquivar objetos y pueden limpiar celdas sucias. Se cargan para que no se les acabe su batería, si se le acaba ya no se mueve.
 - Obstáculos - Los obstáculos se quedan en el mismo lugar.
 - Basura - Se queda en el mismo lugar y se elimina si en la misma celda hay un agente de tipo roomba.
 - Estación de carga - La estación de carga se queda en el mismo lugar.
- **Environment:** En el ambiente se encuentran todos los tipos de agentes mencionados anteriormente. Las basuras y obstáculos aparecen en posiciones aleatorias, también los agentes pero sólo en la simulación 2 (en la primera el único agente está en 1,1). Y las estaciones de carga están en donde aparecen inicialmente los agentes.
- **Actuators:** Los actuators del sistema pertenecen principalmente a los agentes de tipo roomba, que puede desplazarse por el tablero moviéndose de una celda a otra, activar su función de limpieza al entrar en una celda con basura para recoger al agente de tipo basura correspondiente e incrementar el contador de basura recolectada en el modelo. También conectarse a la estación de carga cuando llega a la celda de un agente estacion de carga para recargar su batería y seguir operando. Los obstáculos, basura y estaciones de carga son agentes fijos que no tienen actuators propios, solo modifican el entorno por

su presencia, obstaculizando el paso, representando basura o funcionando como punto de carga.

- **Sensors:** En la simulación los dos agentes que perciben el ambiente son los roombas y la basura. Los roombas en cada movimiento analizan sus celdas vecinas para ver si hay agentes de basura. También antes de hacer un movimiento analiza la celda a la que se quiere mover y si esta tiene un agente de tipo obstáculo lo esquiva. Por otro lado, el agente de basura en cada paso analiza si hay un agente de tipo roomba en su celda, para así removérse.

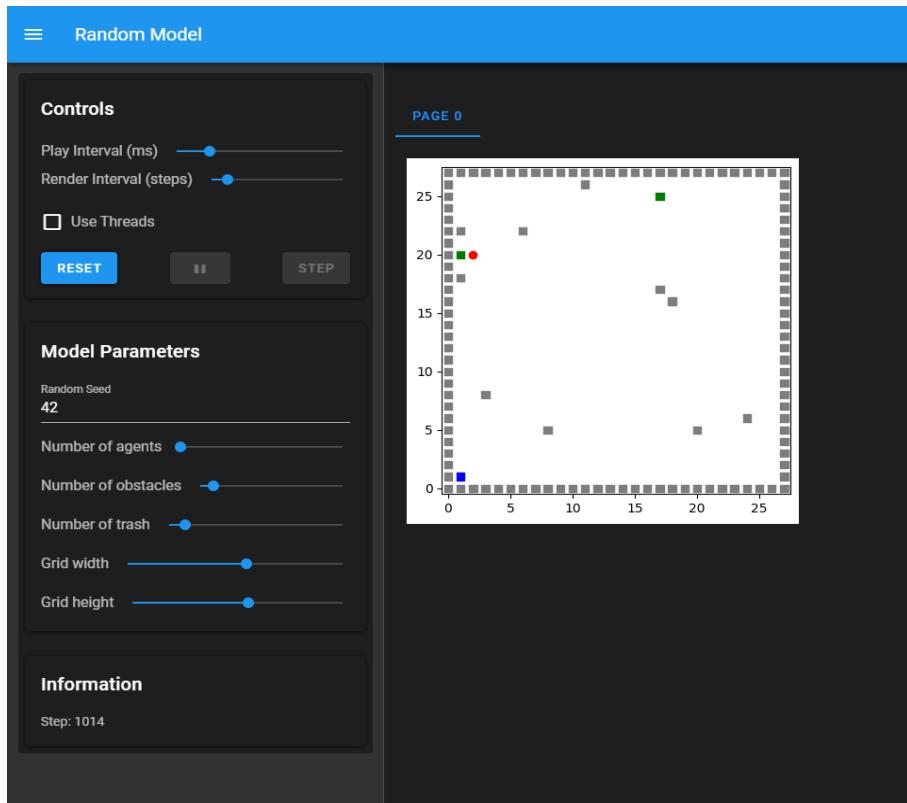
Las características del ambiente son las siguientes:

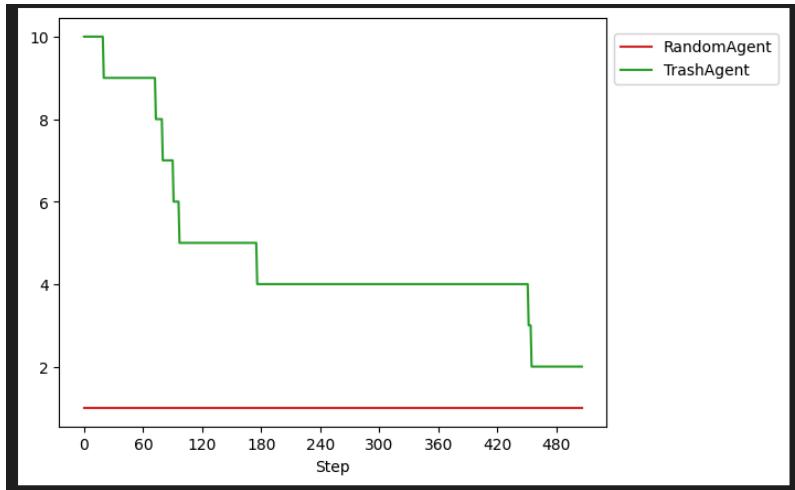
- Es un ambiente **inaccesible**, pues los agentes solo conocen información de lo que han recorrido, no tienen la información completa del ambiente. Además si llegan a una sección solo conoce la información en el momento de recorrerla, no información actualizada.
- El ambiente es **determinista**, cada acción tiene una consecuencia general determinada, es decir que las acciones no generan incertidumbre al hacer una acción. La única acción que genera resultados un poco diferentes es la de esquivar los obstáculos pues el camino que toma para hacerlo puede ser diferente, sin embargo el objetivo general que es esquivar siempre se logra.
- Otra característica del ambiente es que es **no episódico**, el agente decide qué acción tomar dependiendo de lo que está pasando en el episodio actual. Aunque son importantes los episodios futuros el agente no razona sobre sus interacciones entre el actual y los futuros.
- El ambiente es completamente **estático**, es decir que no cambia a menos que el agente haga cambios sobre él. En el caso de estas simulaciones el ambiente solo cambia cuando un agente de tipo roomba limpia las basuras, no hay otro agente que cambie el ambiente o alguna otra acción que lo cambie.
- Por último el ambiente es **discreto**, solo hay un número finito y fijo de acciones que los agentes pueden hacer. Los agentes no pueden actuar por sí solos y en la simulación ya

tienen sus diferentes estados definidos, por lo que si llega a un estado específico actúa conforme está establecido en la jerarquía.

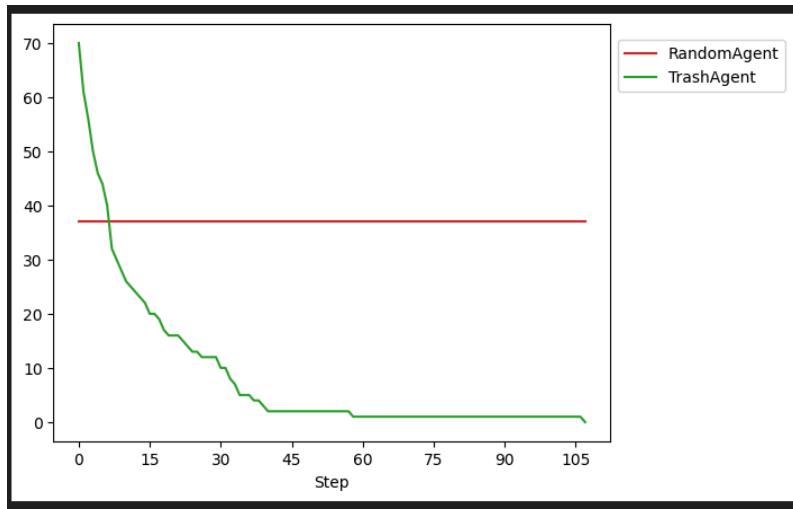
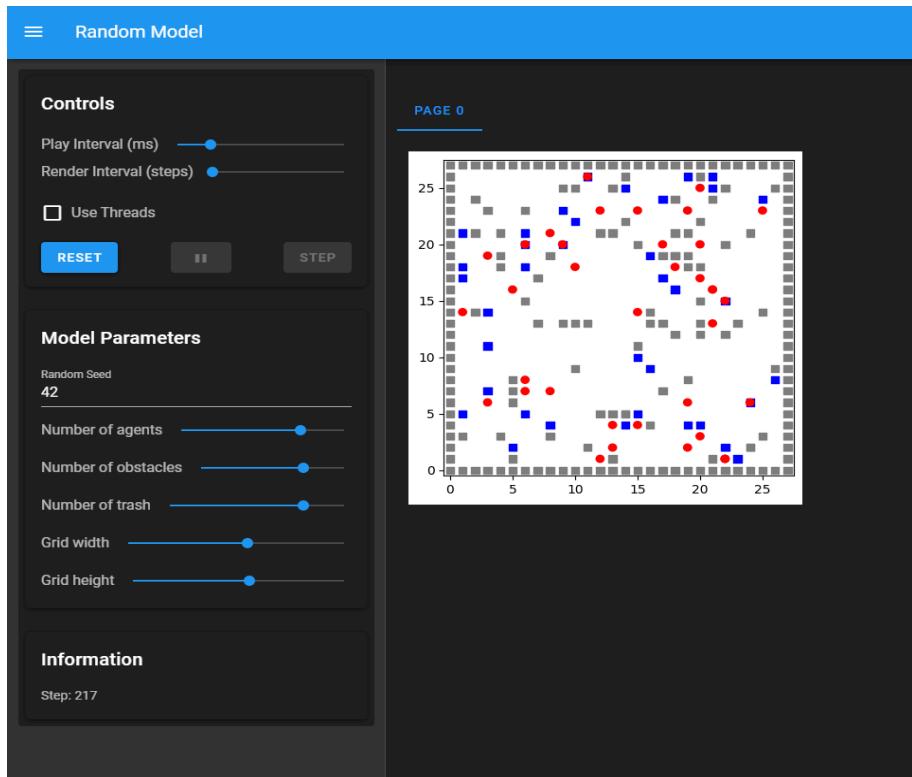
Estadísticas recolectadas:

- Cantidad de roombas activos
- Cantidad de basura recolectada





Para la simulación con un solo agente, este no logra limpiar todo el cuarto en los steps definidos. Aunque el roomba hace un patrón de movimiento óptimo, esto es más que nada por mi condición de carga. Esta establece que cuando la batería sea menor a la suma de la altura y longitud del tablero (es decir el máximo que le tomaría al roomba llegar desde el punto más lejano), este tiene que regresar a su estación de carga. Una solución para agilizar el recorrido podría ser poner un valor más bajo en esta condición. Sin embargo, una de mis prioridades al desarrollar la simulación fue que los roombas no se quedarán sin batería, y este cambio a la larga tendría un efecto más negativo. Por eso en la gráfica se ve que la línea roja nunca baja, es decir al roomba nunca se le acaba la batería.



Por otro lado, en la simulación con varios agentes, logró que toda la basura se recogiera en un tiempo más óptimo, y muy por debajo del límite establecido. En la gráfica podemos ver como toda la basura es recogida y se puede hacer énfasis en la importancia de la condición de la batería. Como podemos observar en la gráfica el número de roombas nunca baja, por lo que ninguno se queda sin batería, reduciendo considerablemente el número de pasos cuando hay muchas roombas y en varias posiciones.

Conclusión

Los roombas logran cumplir su objetivo principal, que es limpiar todas las celdas del ambiente hasta dejarlo sin basura. Sin embargo, en algunos casos esta limpieza requiere una gran cantidad de pasos, lo que evidencia varias mejoras posibles que, por tiempo y complejidad, no alcancé a implementar. La primera es que los roombas no se dirigen a la estación de carga más cercana, sino únicamente a su propia estación, lo que vuelve menos eficiente la recolección conforme se alejan de ella. Además, los roombas no mapean las celdas recorridas ni guardan información sobre otros agentes que hayan encontrado, por lo que no aprovechan ese conocimiento para tomar decisiones más óptimas y se limitan a seguir el comportamiento predefinido. Otra mejora que considero importante habría sido modificar la función **moveToPoint()** para que, al regresar a la estación de carga, también revisará si hay basura en las celdas vecinas, ya que en su estado actual solo busca llegar al punto objetivo por una ruta corta evitando obstáculos.

Finalmente, en la parte de diseño del código, al inicio concentré casi toda la lógica de comportamiento en la función **move()**, lo que la volvió muy compleja y difícil de debugear. A partir de los comentarios del profesor comencé a separar mejor las acciones del roomba y a implementar funciones independientes de **move()** y mejor como condiciones en **step()**. No obstante, algunos comportamientos como la evasión de obstáculos y el estado de limpieza, no alcancé a implementarlos por completo fuera de **move()** por la complejidad de este cambio. Estas limitaciones permiten identificar con claridad posible trabajo futuro para hacer que los roombas sean más eficientes, más “inteligentes” y con un código mantenable.