ITESM Campus Santa Fe

Nombre del bloque:

TC2008B – Modelación de sistemas multiagentes con gráficas computacionales (Gpo. 302)

Nombre del entregable:

Informe de Actividad Roomba

Equipo 3:

Jin Sik Yoon, A01026630

Profesores:

Gilberto Echeverría Furió

Octavio Navarro Hinojosa

Fecha de entrega:

19 de Noviembre de 2025

Problema y la propuesta de solución

El problema de la actividad consiste en limpiar una habitación discreta de tamaño M x N representada como una grilla de celdas.

En esta habitación existen:

- Celdas limpias
- Celdas sucias (inicialmente un porcentaje aleatorio)
- Celdas con obstáculos (también ubicadas aleatoriamente)
- Una o varias estaciones de recarga de batería.

El agente (o multiagentes) aspiradora debe tener los objetivos de:

- Limpiar la mayor cantidad de celdas sucias posible.
- Gestionar la batería (no quedarse muy lejos del cargador sin batería)
- Operar dentro de un tiempo máximo de simulación (máx. 50000)

Además, se quiere analizar el impacto de la cantidad de agentes sobre:

- El tiempo total de limpieza, así como el número de pasos que siguieron.
- El número total de movimientos realizados.

La solución se implementa como un modelo basado en agentes usando la librería **Mesa**:

- Simulación 1: Solamente existe un agente aspiradora, con un estación de carga fija en la posición (1,1).
- Simulación 2: Existen múltiples agentes, cada uno con su propia estación de carga inicial (una por agente) colocada aleatoriamente.

En ambas simulaciones, los agentes tiene un comportamiento reactivo con una política sencilla:

1. Si la celda actual está sucia, limpia.
2. Si la batería es baja y está lejos del cargador, regresa a su estación a cargar.
3. Si se ubica en el cargador y no está al 100% de pila, cargar.
4. En otros casos, explorar aleatoriamente el entorno, si existe una celda sucia en sn vecino, tiene la prioridad de ir a dicha celda primero.

Durante la simulación, se recolectan las estadísticas para evaluar el desempeño.

Diseño de los agentes

- Objetivo de los agentes

El objetivo principal del agente aspiradora es maximizar el porcentaje de celdas limpias más rápido posible dentro del tiempo máximo, evitando quedarse sin batería.

En el modelo se refleja en limpiar celdas sucias cuando se detecta, regresa a la estación de carga cuando la batería está a punto de agotarse y mantenerse activo el mayor tiempo posible.

- Capacidades efectoras (acciones)

Cada agente *RandomAgent* puede realizar las acciones como moverse a una celda vecina siempre que no haya un obstáculo ni esté ocupada por otro agente aspiradora, limpiar la celda actual si contiene el *DirtyPatch* con *dirty=True* que significa que está sucio, por último, recargar la batería si está en la celda con *ChargingStation*.

Mientras haces esos movimientos, se consume 1% de la batería. Pero, cada paso en una estación de carga, recupera 5% hasta un máximo de 100%.

- Percepción

El agente percibe la batería actual (*self.battery*), el contenido de la celda actual si hay suciedad (*DirtyPatch.dirty*) o si hay una estación de carga (*ChargingStation*), las celdas vecinas (*self.cell.neighborhood*) si contienen obstáculos (*ObstacleAgent*), las celdas sucias (*DirtyPatch*) o si encuentra otros agentes (*RandomAgent*), y la distancia Manhattan desde cualquier celda al cargador principal.

- Proactividad y política de decisión

La política básica del agente es limpieza prioritaria que se debe de limpiar la celda sucia antes de que haga otra cosa, gestión de energía que si la batería está baja, se tiene que regresar a su cargador, navegación hacia cargador que se debe de seleccionar la celda vecina que se reduce la distancia entre el

agente y el cargador, y la exploración que si no hay falta de batería ni celdas sucias a limpiar, se explora a su celda vecina válida elegida al azar.

- Métricas del desempeño

Se recolectan las siguientes métricas a nivel del modelo:

- Pasos hasta que todas las celdas estén limpias (*time_to_clean*), o hasta *max_steps*.
- Porcentaje de celdas limpias (*CleanPercent*)
- Número de movimientos totales (*Movement*s)
- Nivel de batería del agente o promedio de batería en múltiples agentes (*BatteryLevel* / *AvgBattery*)

En la simulación 2, además se registran por agente:

- Movimientos individuales.
- Celdas limpiadas por cada agente.
- Batería individual.

Arquitectura de subsunción de los agentes

El comportamiento del agente se organiza mediante una arquitectura de subsunción compuesta por capas jerárquicas de prioridad. Cada capa controla una necesidad específica del agente e inhibe a las capas inferiores cuando es necesario:

- Capa 0: Seguridad energética

Si la batería es baja y la distancia al cargador es significativa, el agente debe regresar a la estación.
Actualmente usa una heurística simple de distancia Manhattan para elegir celdas que reduzcan la distancia al cargador, aunque podría mejorarse aplicando un algoritmo de planeación como A* para encontrar rutas óptimas y evitar trayectorias largas o con obstáculos.

- Capa 1: Limpieza inmediata

Si la celda actual contiene suciedad (*DirtyPatch.dirty = True*), el agente prioriza la limpieza por encima de cualquier otro comportamiento y reduce la batería en 1%.

- Capa 2: Recarga

Si se encuentra en una *ChargingStation* y si no tiene la batería completamente cargada al 100%, se aumenta la batería en 5% cada paso sin moverse.

- Capa 3: Exploración hacia la celda sucia

Si no hay suciedad en la celda actual ni la batería no es crítica ni está cargando el agente, se puede mover aleatoriamente a una celda vecina que está sucia.

- Capa 4: Exploración aleatoria

Si no hay suciedad en la celda vecina ni la batería no es crítica ni está cargando el agente, se puede mover aleatoriamente a cualquier celda vecina válida.

La prioridad de las capas es:

Regreso a cargar → Limpieza inmediata → Recarga → Exploración

Esto permite que el agente sea reactivo y proactivo al momento de limpiar cuando puede, proteger de quedarse sin batería y explorar cuando es seguro.

Aunque la implementación actual utiliza reglas reactivas simples, esta arquitectura permite incorporar una capa adicional de planeación utilizando **A***.

Este algoritmo permitiría:

- Encontrar rutas óptimas hacia la estación de carga.
- Evitar obstáculos y otros agentes de manera anticipada.
- Reducir el riesgo de que el agente quede sin batería lejos del cargador.
- Mejorar la eficiencia total al disminuir movimientos innecesarios.

De este modo, A* funcionará como una “Capa 0.5” o una extensión de Seguridad energética, actuando antes de tomar movimientos locales y mejorando significativamente la proactividad del agente.

Características del ambiente

Accesible vs inaccesible:
Es inaccesible porque el agente solamente puede conocer su celda actual y sus vecinas del contorno. No puede ver todo el entorno ni la ubicación global de la suciedad ni obstáculos.

Determinista vs no determinista:
El ambiente es no determinista porque el resultado puede variar según el estado actual del sistema, también por ser aleatorio la distribución inicial de los agentes.

Episódico vs no episódico:

Es no episódico porque las acciones de un agente en un paso afectan directamente los estados futuros.

Estático vs dinámico:

El ambiente es estático porque el ambiente no cambia por sí solo, necesita las acciones del agente aspiradora para que se modifique el entorno.

Discreto vs continuo

Es discreto porque el espacio está dividido en las celdas claramente definidas, las acciones ocurren en los pasos y las posiciones posibles son finitas por inexistencia de torus.

El ambiente se modela como una grilla de tamaño M x N celdas que vas modificando durante la simulación con las siguientes características:

- Topología: Grilla finita (sin torus), bordes con límites.
- Contenido de las celdas:
    - Celdas con obstáculos (*ObstacleAgent*), colocados de forma aleatoria según el porcentaje.
    - Celdas sucias (*DirtyPatch*), también asignadas aleatoriamente con un porcentaje inicial.
    - Celdas con estación de carga:
        - Simulación 1: Una sola estación en la coordenada (1,1)
        - Simulación 2: Una estación en la posición inicial de cada agente.
    - Celdas libres.
- Tiempo: Se trabaja en pasos discretos, en cada paso los agentes realizan una acción. El límite máximo en este caso es *max_steps = 50000*.
- Aleatoriedad: Las posiciones iniciales de celdas sucias y los obstáculos se generan de forma aleatoria. Pero en la simulación 2, las posiciones iniciales de los agentes y las estaciones de carga también se generan aleatoriamente en el mapa.

Estadísticas recolectadas y análisis

Todas las simulaciones se realizaron en una grilla de 18 x 18 con:

- dirty_percent = 0.4
- obstacle_percent = 0.1
- max_steps = 50000

Simulación 1 - un solo agente

Se realizaron 5 corridas con un solo agente:

Corrida	Pasos	% limpio final	Movimientos
1	778	100	615
2	1903	100	1532
3	732	100	570
4	1214	100	970
5	1281	100	1027

Promedio aproximados:

- Pasos promedio: 1181.6 pasos
- Movimientos promedio: 942.8 movimientos

En todas las corridas el agente se logró limpiar el 100% de las celdas sucias, pero con una variabilidad grande en tiempo y movimientos, debido por el aleatoriedad de:

- Obstáculos.
- Distribución de suciedad.
- Camino aleatorio de la exploración aunque tenga prioridad a la suciedad.

Simulación 2 - múltiples agentes

En la simulación 2, hizo una prueba con 1 a 10 agentes, 3 corridas por cada cantidad de agentes. A continuación se resume el promedio aproximado de los pasos y movimientos totales:

Agentes	Pasos promedio	Movimientos promedio
1	1410.7	1132
2	719.7	1152
3	315	757
4	239	765
5	305	1226

6	173.7	822
7	131.0	720
8	89.7	565
9	127.3	902
10	111.3	876

En todas las corridas el porcentaje final de las celdas limpias fue 100%.

En comparación de dos simulaciones, aumentar el número de agentes reduce fuertemente el tiempo de limpieza, sobre todo entre 1 a 4 agentes. Después de cierto punto, aparecen rendimientos decrecientes, que el tiempo sigue siendo bajo pero ya no mejora tanto, y puede variar por la aleatoriedad.

Conclusión

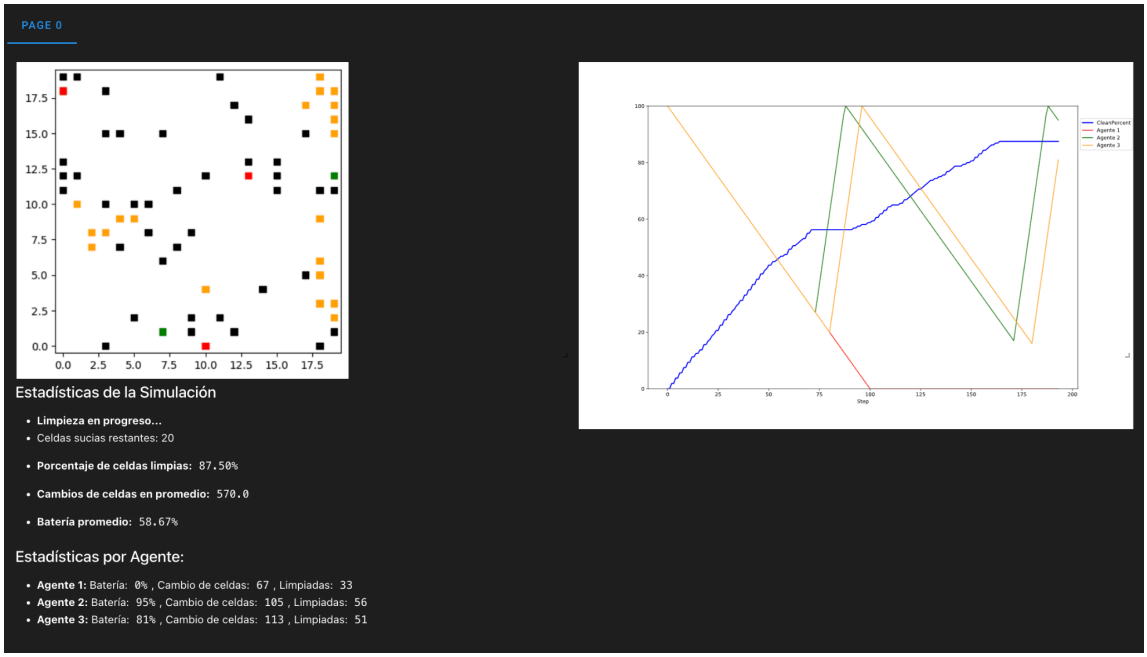
Los resultados obtenidos muestran que el número de agentes tiene un impacto directo en el desempeño del sistema de limpieza. En la Simulación 1, con un solo agente, aunque siempre se alcanza el 100% de limpieza, el proceso requiere muchos pasos y movimientos debido a que el agente debe recorrer prácticamente todo el entorno. La heurística mejorada (priorizar celdas sucias, luego celdas no visitadas y finalmente un movimiento aleatorio) ayuda a reducir exploración innecesaria, pero no elimina la limitación fundamental: un único agente no puede cubrir el ambiente con eficiencia en términos de tiempo.

En la Simulación 2, al aumentar gradualmente la cantidad de agentes, el tiempo total de limpieza disminuye de manera clara. A partir de tres o cuatro agentes, el ambiente se limpia mucho más rápido, incluso llegando a menos de 100 pasos en varios casos. Esto ocurre porque los agentes trabajan en paralelo y se distribuye mejor el espacio gracias a su memoria de celdas visitadas y su prioridad por celdas sucias.

Para lograr simulaciones más equilibradas y consistentes, se realizaron varias mejoras en el comportamiento de los agentes aspiradora. Primero, se ajustó la política de recarga para que los agentes no solo regresaran cuando su batería fuera baja, sino también considerando la distancia mínima necesaria para volver, evitando muertes prematuras. Además, se añadió un mecanismo de memoria mediante un conjunto de celdas visitadas, lo que permite priorizar áreas no exploradas y reducir movimientos

repetitivos e ineficientes. También se redefinió la lógica de movimiento: ahora los agentes siguen un orden de prioridades (celdas sucias → celdas no visitadas → movimiento aleatorio), lo que mejora significativamente la cobertura del mapa. En la simulación multiagente, además, se evitó que los agentes ocuparan la misma celda simultáneamente, reduciendo bloqueos y permitiendo una exploración más fluida. Finalmente, se corrigió la recolección de datos y los cálculos de métricas para asegurar que los resultados fueran comparables entre ambas simulaciones. Estas modificaciones lograron estabilizar el desempeño general, evitar sesgos y generar experimentos más justos y representativos.

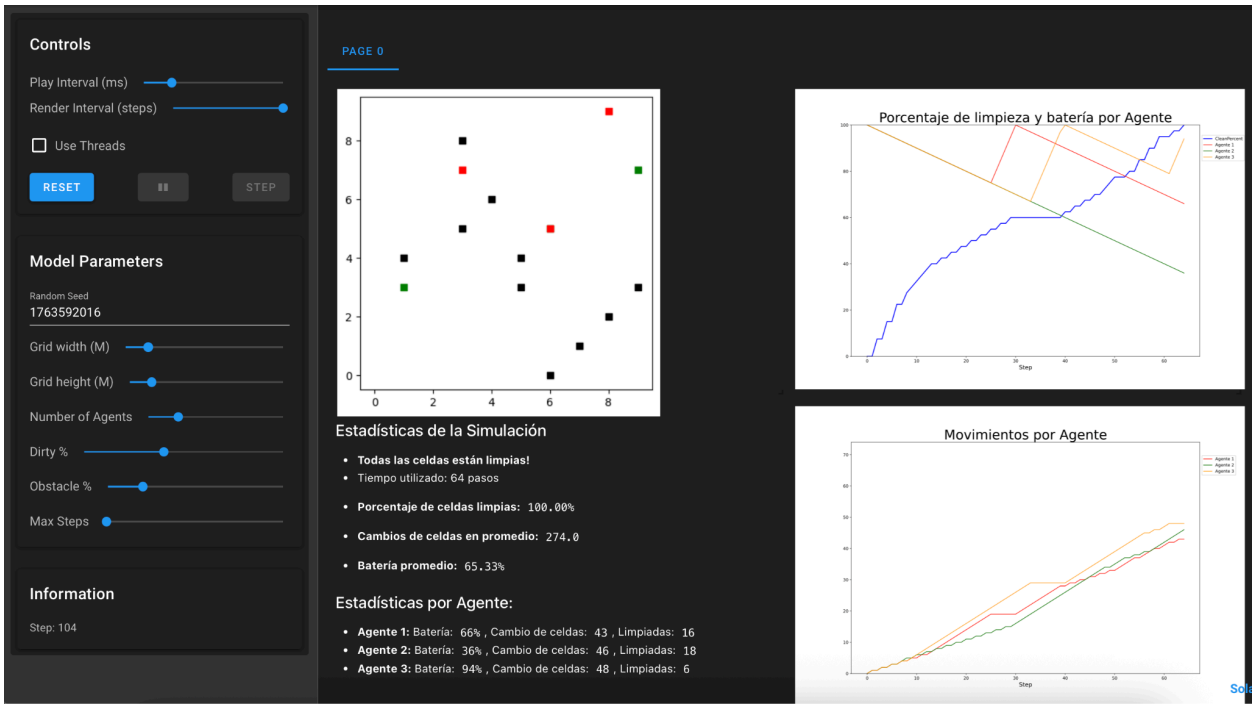
Sin embargo, el número total de movimientos no siempre disminuye; en algunos escenarios incluso aumenta, ya que los agentes no tienen coordinación global y pueden solaparse en ciertas áreas. Aun así, el beneficio principal es evidente: más agentes reducen significativamente el tiempo de limpieza, aunque no siempre reducen el esfuerzo total medido en movimientos.



En algunos experimentos de la simulación multiagente se observó que un agente agotó completamente su batería antes de llegar a su estación de recarga. Esto ocurrió porque, al explorar, el agente se alejó demasiado de su cargador y luego, debido a la presencia de obstáculos y otros agentes bloqueando rutas, tuvo que tomar caminos más largos para regresar. Durante ese trayecto extendido, la batería restante no fue suficiente y terminó inmovilizado en la cuadrícula.

Propuestas para la mejora de esta parte de la simulación:

- Implementar un sistema de ruta (Dijkstra o A* o cualquier sistema de ruta) más específica, que calcule caminos reales evitando obstáculos y agentes.
- Aumentar el margen mínimo de batería requerida para iniciar el regreso (por ejemplo, batería ≤ distancia real × 2).
- Agregar mecanismos de coordinación entre agentes para no bloquear pasillos estrechos.
- Incluir estaciones de recarga secundarias distribuidas en el mapa o permitir que los agentes compartan estaciones cuando están cerca.



En la Simulación 2, además de registrar los movimientos totales y los pasos necesarios para completar la limpieza, se generó una segunda gráfica en la que se muestran los movimientos individuales de cada agente. Esta visualización permite identificar un comportamiento importante: aunque el número de agentes reduce de forma significativa el tiempo total de limpieza, no todos los agentes colaboran de manera eficiente.

Propuestas para la mejora de esta parte de la simulación:

- Detección automática de patrones de movimiento ineficientes.
- Implementar un algoritmo de planeación de rutas más específicas (A, Dijkstra, o BFS por capas).
- Implementar sistema de zonas de trabajo o espacios asignados.
- Aplicar las reglas de paso en pasillos estrechos.

- Ajustar la política de recarga con rutas reales.

El análisis de la gráfica de movimientos individuales permitió identificar patrones de exploración redundante y bloqueos entre agentes. Esto sugiere que incorporar un sistema de planeación de rutas, mecanismos de coordinación y asignación de zonas podría mejorar significativamente la eficiencia colaborativa, reduciendo movimientos innecesarios sin sacrificar la rapidez global de limpieza.