Tecnológico de Monterrey

Campus Estado de México

M1. Actividad

Arellano Conde, Tadeo Emanuel | A01800840
Grajeda Martinez, Yael Sinuhe | A01801044

Profesor:
Dr. Jorge Uresti

MATERIA:
Modelación de sistemas multiagentes con gráficas computacionales

Grupo: 301

14 de noviembre del 2025

1. Introducción

En esta actividad implementamos un sistema multiagente sencillo para estudiar el comportamiento de robots de limpieza reactivos en una habitación representada como una rejilla discreta. Cada agente sigue reglas muy simples: si la celda donde está parado está sucia, la limpia; si está limpia, se mueve de forma aleatoria a alguna de las celdas vecinas.

Para desarrollar la simulación utilizamos Mesa, una librería de Python diseñada para construir modelos basados en agentes (Agent-Based Models, ABM). Mesa proporciona clases para definir agentes y modelos, estructuras de espacio (grids) y herramientas para manejar la activación de agentes y la recolección de datos.

El propósito de la actividad es:

- Implementar un modelo ABM que represente un robot de limpieza reactivo.
- Medir su desempeño bajo diferentes cantidades de agentes.
- Analizar cómo el número de agentes afecta el tiempo de limpieza, el porcentaje de celdas limpias y el número total de movimientos.

2. Descripción del problema

El problema se formula de la siguiente manera:

La habitación se representa como una rejilla de tamaño $M \times N$. Cada celda puede estar en uno de dos estados: sucia o limpia.

Los parámetros de entrada del modelo son:

- Dimensiones de la habitación: M(ancho) y N(alto).
- Número de agentes de limpieza.
- Porcentaje inicial de celdas sucias.
- Tiempo máximo de ejecución (número máximo de pasos de simulación).

2.1 Reglas de comportamiento del agente

En cada paso de tiempo (tick) todos los agentes siguen las mismas reglas:

- **Si la celda actual está sucia:** El agente ejecuta la acción de aspirar y la celda pasa al estado limpio.

- **Si la celda actual está limpia:** El agente elige de forma aleatoria una de las 8 celdas vecinas (vecindario de Moore: arriba, abajo, izquierda, derecha y diagonales).

    - Si la celda elegida está dentro de los límites de la habitación, el agente se mueve a esa celda.
    - Si la celda elegida está fuera de los límites, el agente permanece en su posición actual.

Todos los agentes inician en la misma posición [1,1] (equivalente a la esquina superior izquierda de la rejilla).

2.2 Objetivo de la simulación

La simulación se ejecuta hasta que se cumple alguna de estas condiciones:

- Todas las celdas sucias han sido limpiadas, o
- Se alcanza el tiempo máximo de ejecución definido.

Durante la ejecución se registran:

- El tiempo necesario para limpiar toda la habitación (si se logra).
- El porcentaje de celdas limpias al final de la simulación.
- El número total de movimientos realizados por todos los agentes.

3. Modelo multiagente en Mesa

Nuestro modelo sigue la estructura típica de Mesa, que consiste en definir:

- Una clase de agente, basada en mesa.Agent.
- Una clase de modelo, basada en mesa.Model.
- Un scheduler para activar a los agentes en cada paso.
- Una representación de espacio (MultiGrid) donde se mueven los agentes.
- Un mecanismo de recolección de datos (DataCollector) para guardar estadísticas de la simulación.

3.1 Agente de limpieza

Definimos la clase CleaningAgent, que hereda de mesa.Agent. Sus características principales son:

Cada agente tiene:

- Un identificador único unique_id.
- Una referencia al modelo al que pertenece.
- Un contador de movimientos realizados.

Los agentes comparten la misma lógica reactiva:

- En el método step() revisan si la celda actual está sucia.
- Si la celda está sucia, la limpian.
- Si está limpia, intentan moverse a una de las 8 celdas vecinas elegida al azar.

Cuando un agente realiza un movimiento válido:

- Se actualiza su posición en la rejilla.
- Se incrementa su contador de movimientos y una variable global de movimientos en el modelo.

En el diseño asumimos que solo se cuentan como movimientos aquellos en los que la posición del agente cambia efectivamente de celda. Los intentos de movimiento hacia una celda fuera de la habitación no se contabilizan como movimiento.

3.2 Modelo del entorno

La clase principal del modelo se llama CleaningModel y hereda de mesa.Model. Sus responsabilidades principales son:

- Recibir los parámetros de entrada:
    - width, height (tamaño de la habitación).
    - num_agents (número de agentes).
    - dirty_percentage (porcentaje inicial de celdas sucias).
    - max_steps (tiempo máximo de simulación).

- Crear la estructura del modelo:
  - Una rejilla MultiGrid(width, height, torus=False) para representar la habitación.
  - Un scheduler RandomActivation para activar a los agentes en orden aleatorio.

- Inicializar el estado:
  - Generar las celdas sucias de manera aleatoria:
    - Para cada celda de la rejilla, se marca como sucia con una probabilidad igual a dirty_percentage.
  - Colocar todos los agentes en la celda inicial (0,0), que corresponde a [1,1] en notación humana.
  - Inicializar contadores globales:
    - current_step: número de paso actual.
    - total_moves: número total de movimientos de todos los agentes.
    - dirty_cells: conjunto de celdas que aún están sucias.

- El método step() del modelo se encarga de:
  1. Activar a todos los agentes mediante el scheduler.
  2. Incrementar el contador de tiempo current_step.
  3. Registrar las estadísticas del paso actual usando el DataCollector.
  4. Verificar las condiciones de parada:
     - Si dirty_cells está vacío → la habitación está completamente limpia.
     - Si current_step alcanza max_steps → se detiene la simulación aunque queden celdas sucias.

3.3 Recolección de datos

Para poder analizar el desempeño del sistema, el modelo registra en cada paso:

- Número de paso (step).
- Número de celdas sucias restantes (dirty_cells).
- Porcentaje de celdas limpias (clean_percentage).
- Número total de movimientos (total_moves).

Estos datos se recolectan con mesa.DataCollector, lo que permite exportarlos fácilmente a un DataFrame de pandas para su análisis posterior.

4. Diseño experimental

El objetivo del experimento es analizar cómo influye el número de robots de limpieza en el desempeño del sistema, medido en términos de tiempo de limpieza, porcentaje final de celdas limpias y número total de movimientos realizados por los agentes.

Para ello se fija un entorno base común a todos los experimentos:

- Tamaño de la grilla: 20 × 20 celdas.
- Porcentaje inicial de celdas sucias: 40 % (0.4).
- Tiempo máximo de simulación: 1000 pasos de tiempo (max_steps = 1000).
- Tipo de agente: robot reactivo que limpia la celda actual si está sucia; en caso contrario se mueve a una celda vecina aleatoria en la vecindad de Moore (8 direcciones).
- Condición de paro: la simulación se detiene cuando todas las celdas están limpias o cuando se alcanza el tiempo máximo de 1000 steps.

El factor que se varía es el número de agentes de limpieza n_agents.

En nuestro caso probamos los siguientes valores:

- n_agents ∈ {1, 2, 3, 5, 8, 10, 15, 20, 40, 80, 100}

(estos valores pueden ajustarse según la versión final del código, pero siempre manteniendo el máximo en 100 agentes).

Para cada valor de n_agents se ejecuta el modelo con los parámetros anteriores y se registran las siguientes variables de respuesta:

1. time_used: número de pasos que tomó la simulación hasta detenerse (ya sea por limpiar todas las celdas o por llegar a 1000 steps).

2. final_clean_pct: porcentaje de celdas limpias al final de la simulación.

3. final_total_moves: suma de todos los movimientos realizados por los agentes durante la ejecución.

La función run_experiment del código implementa este procedimiento: crea una instancia de CleaningModel con los parámetros especificados, ejecuta la simulación hasta la condición de paro y devuelve las tres métricas anteriores. Posteriormente se repite el experimento para cada valor de n_agents y se construye un DataFrame con los resultados, que se utiliza para generar las gráficas:

- Tiempo de limpieza vs número de agentes.
- Movimientos totales vs número de agentes.

Con este diseño experimental podemos observar cómo impacta agregar más robots en la velocidad de limpieza y en el "costo" de movimiento del sistema, manteniendo constante el entorno (20×20, 40 % de suciedad inicial, 200 steps como máximo).

5. Resultados

5.1 Resumen numérico

Después de ejecutar varias corridas independientes para cada configuración de número de robots de limpieza, se calcularon los promedios de las métricas obtenidas (tiempo de simulación hasta detenerse, porcentaje final de celdas limpias y movimientos totales del sistema). En la Tabla X se resumen estos valores promedio, que utilizamos como base para el análisis posterior:

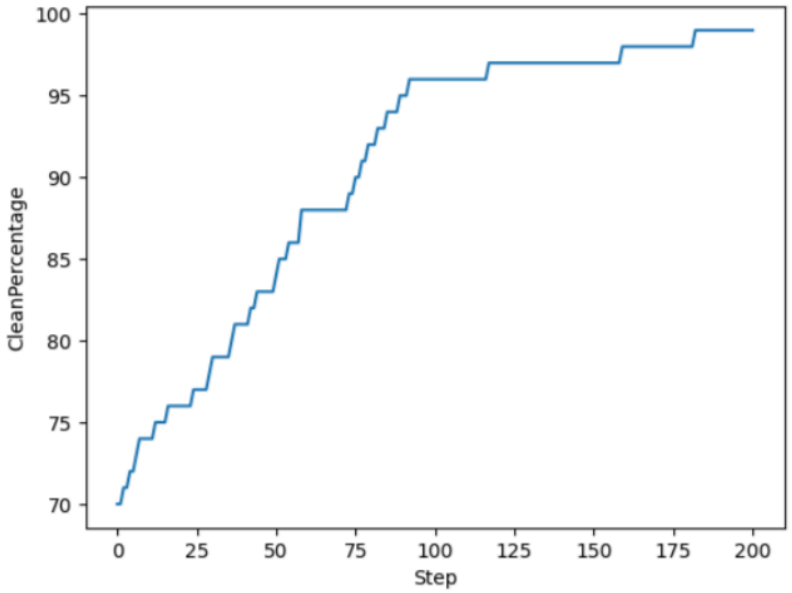
n_agents	Tiempo promedio (steps)	% limpio promedio	Movimientos totales promedio
1	1000.0	88.3	887
2	1000.0	94.8	1 861
3	987.5	97.4	2 813
5	910.0	99.8	4 391
8	696.0	100.0	5 408
10	751.8	100.0	7 358
15	430.0	100.0	6 290

20	344.3	100.0	6 725
40	277.3	100.0	10 930
80	174.3	100.0	13 780
100	195.3	100.0	19 365

5.2 Gráficas

Para visualizar el comportamiento del modelo se utilizó Solara integrado con Mesa. En el código se configuró la interfaz mediante SolaraViz, definiendo los parámetros del modelo (número de agentes, tamaño de la grilla, porcentaje de celdas sucias y tiempo máximo) y un SpaceRenderer para mostrar la grilla. Además, se agregaron componentes de gráfica (CleanPlot y MovesPlot) que leen directamente los datos recolectados por el DataCollector del modelo, generando de forma automática e interactiva las curvas de desempeño.

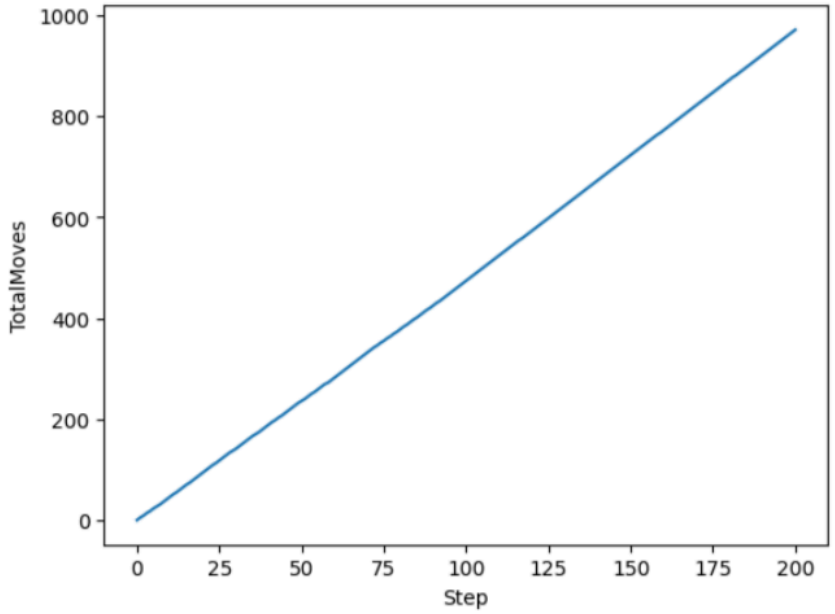
Gráfica 1: Porcentaje de limpieza vs. número de pasos



La primera gráfica muestra el porcentaje de celdas limpias en función del número de pasos (CleanPercentage vs Step). Se observa una curva creciente en forma de “escalera”: el modelo inicia alrededor de 70 % de limpieza y, conforme los robots avanzan, el porcentaje va aumentando hasta acercarse al 100 % cerca del paso 200. Esto confirma que los agentes reactivos van cubriendo progresivamente las

zonas sucias y que, con la configuración usada, el entorno termina prácticamente limpio dentro del límite de tiempo.

Gráfica 2: Movimientos totales promedio vs. número de pasos.



La segunda gráfica representa el número total de movimientos acumulados de todos los agentes (TotalMoves vs Step). La curva crece casi de manera lineal desde 0 hasta alrededor de 1000 movimientos, lo que indica que en prácticamente cada paso la mayoría de los robots realiza desplazamientos. Esta gráfica permite interpretar el “costo” global en movimientos (y, en un escenario físico, en energía) asociado a lograr el nivel de limpieza observado en la primera gráfica. Juntas, ambas visualizaciones muestran cómo evoluciona la limpieza y qué esfuerzo total requiere el sistema multiagente para alcanzarla.

6. Análisis y discusión

En general, observamos que al aumentar el número de agentes de 1 a 3 y 5, el porcentaje promedio de limpieza se incrementa de manera importante: de alrededor de 88 % con un solo robot a casi 100 % con cinco robots. Con pocos agentes, cada uno debe recorrer una proporción muy grande de la habitación y, aunque la simulación llega al límite de 1000 pasos, todavía quedan celdas sucias (especialmente para 1 y 2 robots).

A partir de cierto punto (en nuestro caso, desde 8 agentes en adelante), el modelo logra 100 % de celdas limpias en todas las corridas y el tiempo promedio de limpieza disminuye de forma notable: pasa de ~910 pasos con 5 agentes a ~696 con 8 y ~277 con 40. Esto confirma la idea de que al agregar más robots el trabajo se reparte y la limpieza se completa en menos pasos.

Sin embargo, también se observan rendimientos decrecientes: al comparar, por ejemplo, 40, 80 y 100 agentes, la mejora en tiempo ya no es tan dramática (de ~277 a ~174 y ~195 pasos), mientras que el número total de movimientos del sistema sigue creciendo mucho. Es decir, agregar más agentes sigue ayudando, pero cada agente adicional aporta cada vez menos a reducir el tiempo total de limpieza.

En cuanto al número total de movimientos, se aprecia que aumenta con el número de agentes, ya que hay más robots moviéndose en paralelo en cada paso. Aunque la simulación termina en menos steps, el esfuerzo global del sistema (movimientos totales) puede mantenerse alto o incluso crecer de forma significativa debido a la superposición de trayectorias y a que varios agentes recorren zonas similares. Desde la perspectiva de un sistema físico, esto implicaría un mayor consumo de energía y desgaste cuando se usan muchos robots.

Desde la perspectiva de sistemas multiagente, estos resultados muestran cómo un conjunto de agentes con reglas totalmente locales y reactivas (sin memoria ni comunicación explícita) puede generar comportamientos globales complejos. El simple hecho de variar el número de agentes altera el patrón de exploración de la habitación y, por lo tanto, el desempeño global del sistema. En nuestro caso, se identifica claramente un umbral mínimo de agentes para lograr la limpieza completa, así como una zona en la que aumentar el número de robots ya no mejora mucho el tiempo, pero sí incrementa el costo global en movimientos.