Curso:

Modelación de Sistemas Multiagentes con Gráficas Computacionales

Grupo: 302

Estudiante:

Fausto Izquierdo Vejar - A01785221

Maestro:

Octavio Navarro

Título:

MA. Actividad: Roomba

Fecha de entrega:

22 de noviembre 2025

Simulación de Limpieza con Agentes Roomba

1. Problema y Propuesta de Solución

Se necesita simular un sistema de robots roombas de limpieza que deben limpiar un espacio con celdas sucias distribuidas aleatoriamente, evitando obstáculos y tomando en cuenta su batería limitada para poder operar hasta que el espacio esté limpio

- Algoritmo de Dijkstra para calcular rutas óptimas
- Se calcula constantemente la distancia a estación de carga
- Arquitectura de subsunción: Comportamientos con prioridades para decisiones “inteligentes”
- Conocimiento parcial: Cada agente conoce solo su estación inicial pero puede usar cualquiera

2. Diseño de los Agentes

RoombaAgent

Tiene el objetivo de limpiar todas las celdas sucias manteniendo suficiente batería para regresar a cargar.

Capacidades Efectoras:

- Movimiento cardinal (arriba, abajo, izquierda, derecha)
- Limpieza de celdas sucias (costo: 1% batería)
- Recarga en estaciones (5% por paso)

Percepción:

- Posición actual en el grid
- Detección de otros agentes en celda actual (suciedad o estación de carga)
- Nivel de batería (0-100%)
- Conocimiento global de posiciones de obstáculos y celdas sucias

- Ubicación de estación de carga

Proactividad:

- Calcula rutas óptimas antes de moverse
- Chequeo constante de batería
- Busca activa de objetivo más cercano

Métricas Individuales:

- Movimientos totales realizados
- Celdas limpiadas por este agente
- Tiempo total cargando batería
- Nivel actual de batería

3. Arquitectura de Subsunción

Los agentes tienen 4 niveles de prioridad:

- Nivel 1 (Mayor Prioridad): Cargar Batería
    - Condición: Estar en estación de carga con batería < 100%
    - Acción: Cargar batería
    - Justificación: Sin batería el agente no funciona
- Nivel 2: Limpiar Celda Actual
    - Condición: Estar en celda sucia
    - Acción: Limpiar la celda
    - Justificación: Eficiencia - si ya estamos aquí, limpiar inmediatamente
- Nivel 3: Regresar por Batería Baja
    - Condición: Batería insuficiente para regresar (margen de seguridad del 10%)
    - Acción: Calcular y seguir ruta a estación de carga
    - Justificación: Prevenir quedarse sin batería lejos de estación
- Nivel 4: Navegar a Celda Sucia
    - Condición: Batería disponible y existen celdas sucias
    - Acción: Encontrar celda sucia más cercana (Dijkstra) y seguir ruta óptima
    - Justificación: Cumplir objetivo principal de limpieza

4. Características del Ambiente

- Accesibilidad: Parcialmente observable (agentes solo conocen su estación inicial)
- Determinismo: Determinista
- Temporalidad: Secuencial (decisiones afectan estados futuros)
- Dinamismo: Semi-dinámico (otros roombas modifican el ambiente)
- Espacio: Discreto (grid con posiciones enteras)
- Agentes: Multi-agente competitivo/colaborativo
- Grid: Grid de Mesa, permite múltiples agentes por celda, no toroidal

Agentes Fijos:

- DirtyCell: Representa suciedad, eliminada al ser limpiada
- Obstacle: Bloquea movimiento, no puede atravesarse
- ChargingStation: Punto de recarga, compartible entre agentes

Configuración Inicial:

- Agentes spawnean en posiciones aleatorias
- Cada agente tiene una ChargingStation en su posición inicial
- Suciedad distribuida según porcentaje configurado
- Obstáculos distribuidos según porcentaje configurado

5. Estadísticas Recolectadas (se imprimen al terminar la simulación)

Métricas Globales (Nivel Modelo):

- Dirty Cells: Número de celdas sucias restantes
- Average Battery: Promedio de batería de todos los agentes
- Total Movements: Suma de movimientos de todos los agentes
- Cells Cleaned: Celdas limpiadas
- Roombas: Número de agentes activos

Métricas Individuales (Nivel agente):

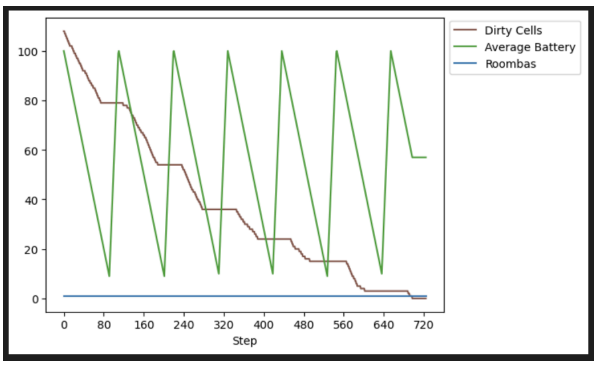
- Movements: Contador de movimientos realizados
- Cells Cleaned: Celdas limpiadas por este agente
- Times Charged: Pasos totales cargando

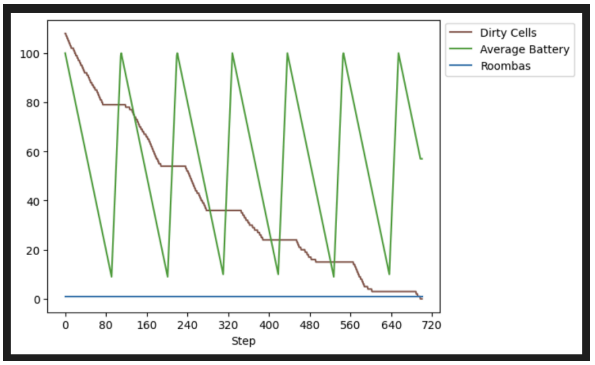
Los datos se visualizan en tiempo real en las gráficas de líneas para métricas globales

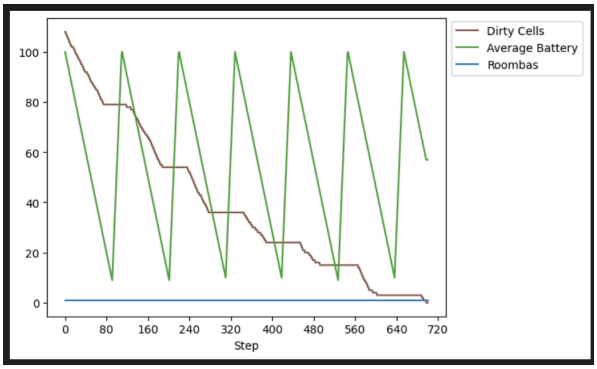
Representación visual en grid con colores según estado:

- Verde: Batería > 60%
- Amarillo: Batería 20-60%
- Rojo: Batería < 20%

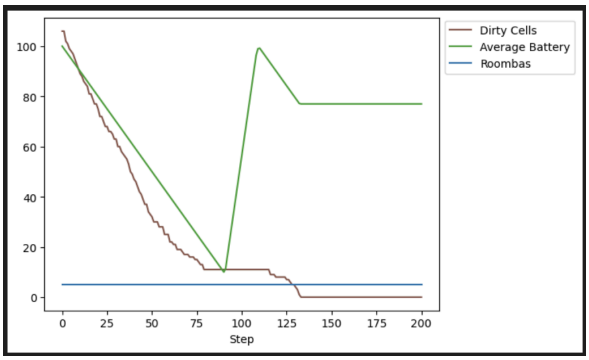
1 agente (3 intentos)

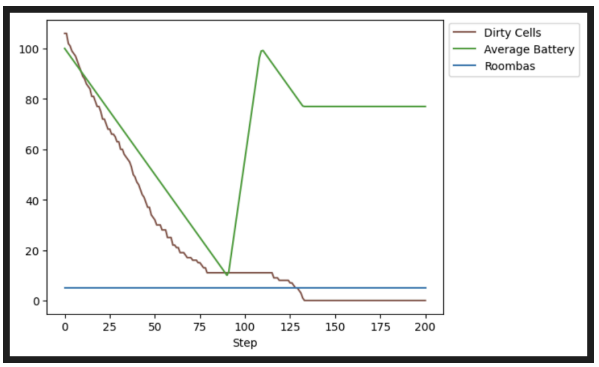


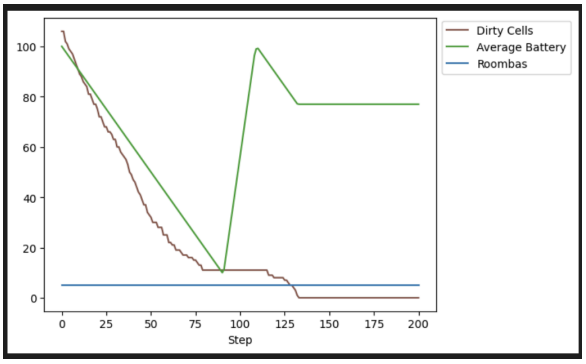




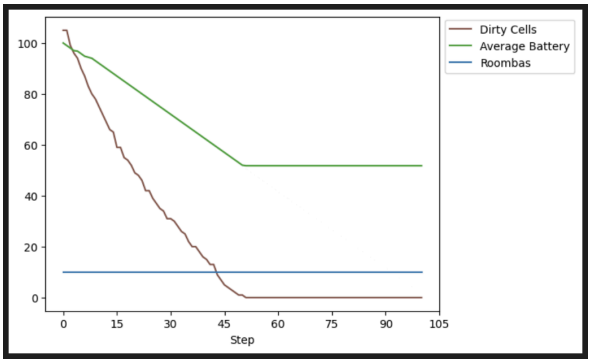
5 agentes (3 intentos)

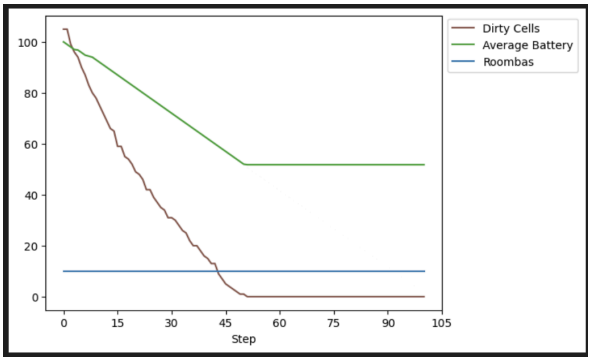


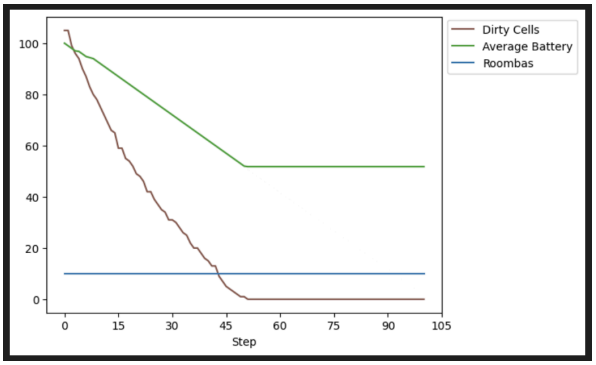




10 agentes (3 intentos)







6. Conclusiones

El diseño tiene fortalezas que lo hacen bastante eficiente para resolver el problema de limpieza. Usar el algoritmo de Dijkstra garantiza que los agentes siempre encuentren rutas

óptimas, para minimizar el consumo de batería y el tiempo de ejecución. La arquitectura de subsunción define las prioridades de los roombas de forma clara. Esta estructura hace que el sistema sea escalable, y soporte múltiples agentes trabajando al mismo tiempo con estaciones de carga compartidas, y el sistema de métricas facilita analizar el rendimiento a nivel individual y global. Sin embargo, el modelo tiene limitaciones; los agentes no cuentan con comunicación entre ellos, por lo que puede haber esfuerzos duplicados al dirigirse a la misma celda sucia. Adicionalmente, su conocimiento del ambiente es limitado ya que solo recuerdan la ubicación de su estación inicial sin aprender sobre otras disponibles (aunque sí tienen conocimiento completo de celdas sucias y obstáculos). También podría mejorarse la planificación de recorrido, pues éste se basa únicamente en el objetivo más cercano sin optimizar secuencias completas de limpieza (greedy).

Los resultados muestran patrones según la configuración: con un solo agente hay una alta eficiencia individual pero baja velocidad total, mientras que configuraciones de 5-10 agentes logran un buen balance entre velocidad y eficiencia, y configuraciones mayores a 10 agentes no fueron puestas a prueba, pues tendrían un rendimiento decreciente por la competencia por objetivos. La arquitectura de subsunción es efectiva para problemas de exploración con recursos limitados (ej: batería), donde la implementación de rutas óptimas y chequeo constante de batería permite una solución eficaz, aunque las principales mejoras deberían enfocarse en implementar comunicación entre agentes y planificación de rutas que optimicen la eficiencia global del sistema.