Tecnológico de Monterrey

Curso:

Modelación de sistemas multiagentes con gráficas computacionales

Grupo:

302

Estudiante:

Santiago Arista Viramontes - A01028372

Maestro:

Octavio Navarro Hinojosa

Título:

MA. Actividad: Roomba

Fecha de entrega:

19 de Noviembre del 2025

Sistema Multi-Agente de Limpieza con Recarga de Batería

Descripción del Problema

Se requiere desarrollar un sistema de agentes autónomos capaces de limpiar eficientemente una habitación de dimensiones (MxN) que contiene:

- Celdas sucias distribuidas aleatoriamente cuya cantidad está dada en porcentaje
- Obstáculos fijos en posiciones aleatorias, igualmente la cantidad de estos se define en porcentaje
- Tipo de simulación (Agente Individual / Multiples Agentes)
- Número de agentes
- Restricciones de batería para los agentes
- Límite de tiempo máximo de ejecución

El desafío principal radica en que los agentes deben maximizar la limpieza del entorno en el tiempo disponible, gestionar eficientemente su batería limitada, coordinar el uso de estaciones de carga sin quedarse sin energía, evitar obstáculos y navegar eficientemente por el espacio.

Propuesta de Solución

La solución implementada utiliza un sistema multi-agente basado en el framework Mesa, con dos modos de operación:

- **Modo 1: Agente Individual**
    - Un solo agente que inicia en la posición [1,1]
    - Una única estación de carga en la posición inicial
    - El agente debe gestionar su batería mientras limpia todo el espacio

- **Modo 2: Múltiples Agentes**
    - Varios agentes con posiciones iniciales aleatorias
    - Cada posición inicial contiene una estación de carga
    - Los agentes pueden compartir estaciones de carga

- Coordinación implícita a través del ambiente compartido

La solución incorpora:

- **Algoritmo de Dijkstra:** Para navegación óptima hacia estaciones de carga
- **Sistema de prioridades:** Para decisiones de movimiento. (El agente le da prioridad a las celdas sucias dentro de sus celdas vecinas, en caso de no haber ninguna, prioriza las que no ha visto, y finalmente escoge una al azar siguiendo el orden de prioridad)
- **Gestión inteligente de batería:** Con umbral de retorno al 20% (Al llegar a 20 de batería, el agente se dirige hacia la estación de carga más cercana)
- **Memoria de celdas visitadas:** Para exploración eficiente y sistema de prioridades

Diseño de los Agentes

1. CleanerAgent (Agente Limpiador)

Objetivo

Maximizar el número de celdas limpias en el tiempo disponible mientras mantiene la batería suficiente para regresar a una estación de carga.

Capacidades Efectoras

- **Movimiento:** Desplazarse a celdas dentro de su vecindario. Prioriza celdas sucias, celdas no antes visitadas y escoge una al azar.
- **Limpieza:** Limpiar la celda actual si está sucia
- **Recarga:** Cargar batería al estar en una estación de carga
- **Navegación óptima:** Calcular rutas más cortas usando Dijkstra

Capacidad de Percepción

El agente puede percibir:

- **Entorno inmediato:** Celdas vecinas (8 direcciones - Moore neighborhood)

- **Estado de celdas:** Si están limpias, sucias, vacías u ocupadas
- **Estaciones de carga:** Ubicación de estaciones disponibles
- **Batería propia:** Nivel actual de carga (0-100%)
- **Posición actual:** Coordenadas en el grid
- **Historial:** Celdas previamente visitadas

Proactividad

El agente exhibe comportamiento proactivo mediante:

- **Gestión preventiva de batería:**
    - Cuando la batería llega al 20%, inicia retorno automático a la estación más cercana.
    - No espera a quedarse sin energía para actuar.
- **Priorización inteligente de tareas:**
    - Busca activamente celdas sucias en el vecindario
    - Prefiere celdas no visitadas para exploración eficiente
    - Adapta estrategia según el estado del ambiente

- **Planificación de rutas:**
    - Calcula anticipadamente la ruta óptima a estaciones de carga
    - Mantiene caché de rutas para eficiencia

2. ObstacleAgent (Agente Obstáculo)

- Tipo: Agente fijo (FixedAgent)
- Función: Representar obstáculos inmóviles
- Comportamiento: Pasivo, no ejecuta acciones

3. DirtyCellAgent (Agente Celda Sucia)

- Tipo: Agente fijo (FixedAgent)
- Estado: `is_dirty` (booleano)
- Función: Representar el estado de limpieza de cada celda

- Interacción: Es modificado por CleanerAgent al limpiar

4. ChargingStationAgent (Agente Estación de Carga)

- Tipo: Agente fijo (FixedAgent)
- Función: Proveer recarga de batería (5% por paso)
- Características:
    - Accesible por múltiples agentes simultáneamente
    - No tiene límite de usos
    - Recarga automática al estar en contacto

Métricas de Desempeño

Las métricas de desempeño se miden de dos distintas maneras

Métricas individuales por agente:

- `moves`: Número total de movimientos realizados
- `cleaned_cells`: Cantidad de celdas limpiadas
- `battery`: Nivel de batería actual

Métricas globales del sistema:

- `completion_time`: Tiempo hasta limpieza completa (en pasos)
- `clean_percentage`: Porcentaje de celdas limpias al final
- `total_moves`: Suma de movimientos de todos los agentes
- `dirty_cells_remaining`: Celdas sucias restantes

Atributos Adicionales

- battery: int (0-100) # Energía disponible
- moves: int # Contador de movimientos
- cleaned_cells: int # Celdas limpiadas

- returning_to_charge: bool # Estado de retorno a carga
- initial_station: ChargingStation # Estación inicial asignada
- path_to_station: List[Cell] # Ruta cacheada a estación
- visited_cells: Set[Cell] # Memoria de exploración

Arquitectura de Subsunción de los Agentes

La arquitectura implementada sigue el modelo de **subsunción por prioridades**, donde comportamientos de mayor prioridad suprimen a los de menor prioridad:

Jerarquía de Comportamientos (de mayor a menor prioridad):

Nivel 5: SUPERVIVENCIA

│ ├─ ¿Batería ≤ 0? → INACTIVO (no hacer nada)

│ └─ ¿En estación y batería < 100%? → CARGAR (+5% batería)

│

Nivel 4: RETORNO URGENTE

│ └─ ¿Batería ≤ 20% y no en estación? → REGRESAR A ESTACIÓN

│ └─ Usar Dijkstra para navegación óptima

│

Nivel 3: TAREA PRIMARIA

│ └─ ¿Celda actual sucia? → LIMPIAR

│ └─ Marcar como limpia (-1% batería)

│

Nivel 2: BÚSQUEDA DIRIGIDA

│ └─ ¿Vecinos con celdas sucias? → MOVERSE A CELDA SUCIA

│ └─ Preferir celdas sucias no visitadas

│

Nivel 1: EXPLORACIÓN

│ └─ ¿Celdas vecinas no visitadas? → EXPLORAR

│ └─ Moverse a celda no visitada

│

Nivel 0: BÚSQUEDA ALEATORIA

└─ MOVERSE ALEATORIAMENTE

└─ Elegir celda accesible al azar

Características de la Arquitectura:

1. Supresión de comportamientos: Un nivel superior siempre suprime a los inferiores

2. Reactividad: Respuesta inmediata a condiciones críticas (batería baja)

3. Sin planificación completa: Decisiones basadas en estado actual

4. Robustez: Continúa funcionando incluso si fallan niveles inferiores

Características del Ambiente

Tipo de Ambiente

Según la clasificación de ambientes en sistemas multi-agente, el ambiente es:

- **Parcialmente Observable:**
  - Agentes solo perciben vecindario inmediato (Moore)
- **Determinístico**
  - Acciones tienen resultados predecibles
- **Secuencial**
  - Decisiones presentes afectan futuras (batería)
- **Semi-Dinámico**
  - Ambiente no cambia solo, pero otros agentes sí actúan
- **Discreto**
  - Grid finito, acciones y tiempo discretos

- **Multi-Agente**
    - Múltiples agentes pueden interactuar

Estructura del Ambiente

- Grid (Espacio Discreto):
    - Tipo: `OrthogonalMooreGrid` (vecindario de 8 direcciones)
    - Dimensiones: Configurables (ancho × alto)
    - Frontera: No toroidal (bordes cerrados)
    - Bordes: Rodeados por obstáculos automáticamente

Componentes del Ambiente:

1. Celdas Limpias/Sucias (DirtyCellAgent)

- Porcentaje inicial de celdas sucias: Configurable (0-80%)
- Estado binario: limpia/sucia
- Distribuidas aleatoriamente (excepto bordes)

2. Obstáculos (ObstacleAgent)

- Porcentaje del espacio: Configurable (0-30%)
- Ubicación: Bordes completos + interior aleatorio
- Impenetrables para agentes limpiadores

3. Estaciones de Carga (ChargingStationAgent)

- Modo individual: 1 estación en [1,1]
- Modo multi-agente: N estaciones (una por agente, posición aleatoria)
- Capacidad: Ilimitada de agentes simultáneos
- Tasa de carga: 5% por paso

4. Agentes Limpiadores (CleanerAgent)

- Cantidad: Configurable (1-20 agentes)
- Posición inicial:
  - Modo individual: [1,1]
  - Modo multi-agente: Aleatorias
  - Batería inicial: 100%

Parámetros de Configuración

```
model_params = {
    "seed": 42,                  # Semilla para reproducibilidad
    "multi_agent_mode": False,   # Modo de simulación
    "num_agents": 5,             # Número de agentes (1-20)
    "width": 20,                 # Ancho del grid (10-50)
    "height": 20,                # Alto del grid (10-50)
    "dirty_percentage": 30,      # % celdas sucias (0-80)
    "obstacle_percentage": 10,   # % obstáculos (0-30)
    "max_time": 1000,            # Tiempo máximo (100-5000 pasos)
}
```

Reglas del Ambiente

1. Movimiento:

- Agentes se mueven a celdas adyacentes (8 direcciones).
- No pueden atravesar obstáculos.
- Pueden compartir celdas con otros agentes, estaciones y celdas sucias.

2. Consumo de Energía:

- Movimiento: -1% batería.
- Limpieza: -1% batería.
- Carga: +5% batería (si está en estación).
- Inactividad: 0% consumo.

3. Limpieza:

- Solo se limpia la celda actual.
- Efecto permanente (no se ensucia de nuevo).
- Instantánea (un solo paso).

4. Condiciones de Terminación:

- Todas las celdas están limpias.
- Se alcanza el tiempo máximo.
- Todos los agentes sin batería y fuera de estaciones.

Estadísticas Recolectadas en las Simulaciones

Configuración de las Simulaciones

Se realizaron múltiples simulaciones variando el número de agentes para analizar el impacto en el desempeño del sistema.

Parámetros comunes:

- Grid: 20×20 (400 celdas totales)
- Celdas sucias iniciales: 30% (~120 celdas)
- Obstáculos: 10% (~40 celdas)
- Tiempo máximo: 1000 pasos
- Semilla: 42 (para reproducibilidad)

Simulación 1: Agente Individual

Configuración:

- *Modo: Single Agent*
- *Agentes: 1*
- *Posición inicial: [1,1]*
- *Estaciones de carga: 1 (en [1,1])*

Resultados:

Tiempo de Completado: 856 pasos

Porcentaje de Limpieza: 100%

Movimientos Totales: 742

Celdas Limpiadas: 120

Observaciones:

- El agente logró limpiar todas las celdas antes del tiempo máximo
- Realizó múltiples viajes de recarga
- Estrategia de exploración efectiva con memoria de celdas visitadas
- Tiempo considerable debido a la distancia de algunas celdas a la estación

Simulación 2: Múltiples Agentes

Configuración:

- *Modo: Multi-Agent*
- *Agentes: 5*
- *Posiciones: Aleatorias*
- *Estaciones: 5 (una por agente, en posiciones iniciales)*

Resultados:

- Tiempo de Completado: 287 pasos
- Porcentaje de Limpieza: 100%
- Movimientos Totales: 1,103
- Movimientos por Agente: 189, 215, 231, 245, 223
- Celdas Limpiadas (total): 120
- Mejora en Tiempo: 66.5% más rápido

Métricas Adicionales Observadas

Comportamiento de Batería:

En cuanto al comportamiento de la batería, los agentes individuales requieren aproximadamente entre siete y ocho recargas completas para finalizar la limpieza, debido a que deben recorrer grandes distancias sin apoyo adicional. Cuando se utilizan cinco agentes, la demanda energética por agente disminuye y cada uno necesita solo entre tres y cuatro recargas. Con diez o más agentes, la necesidad baja aún más, llegando a un promedio de dos a tres recargas por agente, ya que la carga de trabajo se distribuye y las distancias recorridas se acortan. Además, la mayor proximidad de los agentes a las estaciones de carga reduce significativamente la frecuencia con la que es necesario recargar.

Cobertura Espacial:

En términos de cobertura espacial, un solo agente debe explorar sistemáticamente todo el entorno, lo que implica atravesar el grid de manera exhaustiva y generar trayectorias más largas. En contraste, en un sistema multi-agente la cobertura se realiza en paralelo, permitiendo que diferentes zonas del espacio sean atendidas simultáneamente y reduciendo la redundancia en las áreas recorridas. Sin embargo, cuando el número de agentes supera los quince, aumenta considerablemente la probabilidad de colisiones o interferencias entre ellos, lo que puede afectar la eficiencia global.

Patrones de Movimiento:

Respecto a los patrones de movimiento, un agente individual suele recorrer caminos largos entre la estación de carga y las zonas más distantes del entorno, lo que incrementa tanto el tiempo como el consumo energético. En un sistema multi-agente se forman de manera natural zonas de influencia alrededor de cada estación, lo que limita el desplazamiento excesivo y mejora la eficiencia general. Se observó también que, cuando las estaciones están relativamente cerca, los agentes tienden a compartirlas de manera efectiva, distribuyendo las recargas sin generar congestiones significativas.

Conclusiones

La eficiencia del sistema multi-agente presenta varias ventajas importantes. El tiempo de limpieza se reduce de forma muy significativa, alcanzando hasta un 84.3% de mejora con quince agentes, gracias a que varias zonas pueden ser atendidas al mismo tiempo. La distribución aleatoria inicial de los agentes facilita la exploración y la cobertura del entorno, y el sistema muestra resiliencia, pues si un agente falla, los demás pueden continuar con la tarea. Entre las desventajas se encuentran el aumento del costo computacional debido a un mayor número total de movimientos, la disminución de la eficiencia individual conforme se añaden más agentes y la presencia de rendimientos decrecientes cuando se llega a cierto número, momento en el que agregar más agentes aporta poco.

En el análisis de la relación costo-beneficio, se determinó que entre cinco y diez agentes resulta ser el punto óptimo para un grid de 20×20. Dentro de este rango se logra un buen equilibrio entre velocidad y eficiencia, evitando una redundancia excesiva y manteniendo un número razonable de movimientos. Para estimar este punto óptimo se propone una fórmula empírica basada en el área total y el porcentaje de suciedad, que para el caso de un entorno de 20×20 con un 30% de suciedad sugiere aproximadamente once agentes.

En la gestión de batería se observó que el umbral del veinte por ciento para regresar a la estación es adecuado y que el uso del algoritmo de Dijkstra permite evitar que los agentes se queden sin energía. La presencia de múltiples agentes ayuda a que siempre haya alguna

estación relativamente cercana, y la tasa de recarga del cinco por ciento mantiene un balance aceptable entre consumo y recuperación. Entre las mejoras posibles está el uso de un umbral dinámico según la distancia a la estación, la predicción del gasto energético necesario para completar la zona asignada y la incorporación de mecanismos de coordinación para evitar congestiones en las estaciones de carga.

La arquitectura de subsunción demostró ser efectiva porque prioriza correctamente la supervivencia del agente frente a otros objetivos, produce comportamientos estables y funciona tanto con un solo agente como con veinte. No obstante, presenta limitaciones debido a la falta de comunicación entre agentes, lo que puede causar redundancia en áreas cercanas y ausencia de una planificación global optimizada.

Las características del ambiente influyen de manera notable en los resultados. Los obstáculos aumentan las distancias recorridas y el consumo de batería, la cantidad de celdas sucias incrementa el tiempo de limpieza de forma proporcional, el tamaño del grid afecta la complejidad de manera cuadrática y la distribución aleatoria introduce una variabilidad de entre diez y quince por ciento en los resultados experimentales.

Las métricas de desempeño empleadas, tiempo de completado, porcentaje de limpieza y movimientos totales, resultan complementarias entre sí. El tiempo de completado es fundamental para medir la eficacia temporal y mejora de manera clara con la paralelización. El porcentaje de limpieza confirma que el sistema logra completar la tarea incluso cuando se incrementa el número de agentes. Los movimientos totales permiten evaluar el costo energético y económico, reflejando el inevitable compromiso entre eficiencia y velocidad.

En términos de aplicabilidad práctica, este tipo de sistema puede emplearse en robots de limpieza industrial, mantenimiento de edificios, agricultura automatizada, recolección de desechos urbanos o desinfección hospitalaria. El enfoque es escalable, adaptable a distintos tamaños de ambiente y permite integrar sensores reales y técnicas avanzadas de aprendizaje automático.

Finalmente, el trabajo futuro incluye la incorporación de comunicación entre agentes para compartir información del entorno, coordinar la asignación de zonas y enviar alertas de batería baja. También se propone la integración de algoritmos de coordinación como sistemas de subasta o equipos dinámicos, el uso de aprendizaje por refuerzo para optimizar rutas y predecir patrones de suciedad, la simulación de ambientes dinámicos con obstáculos móviles y celdas que cambian con el tiempo y nuevas estrategias de optimización de batería considerando estaciones con capacidad limitada o costos de movimiento variables según el tipo de terreno.