Tecnológico de Monterrey

Actividad Roomba

Santiago Coronado Hernández A01785558

Prof. Octavio Navarro Hinojosa

Modelación de sistemas multiagentes con gráficas computacionales

Gpo. 302

18 de Noviembre del 2025

Problema que se está resolviendo, y la propuesta de solución.
El problema que se está resolviendo es la limpieza de una área utilizando roombas (randomAgents) mientras evitan obstáculos y ven su batería. El objetivo es maximizar la limpieza del entorno en el menor tiempo posible, considerando restricciones como la batería limitada y la necesidad de recargar.

La propuesta de solución consiste en un modelo basado en agentes que interactúan en un grid. Los agentes tienen comportamientos específicos como moverse aleatoriamente, buscar basura, evitar obstáculos, recargar batería en estaciones y limpiar celdas. El modelo utiliza una arquitectura de simulación basada en el framework Mesa, que permite la visualización y recolección de estadísticas.

El diseño de los agentes (objetivo, capacidad efectora, percepción, proactividad, métricas de desempeño, etc.).

- Objetivo:
  - RandomAgent: Limpiar la mayor cantidad de basura posible mientras gestiona su batería.
  - ObstacleAgent: Representar obstáculos estáticos en el entorno.
  - TrashAgent: Representar basura que debe ser limpiada.
  - RechargeStationAgent: Proveer un lugar donde los agentes puedan recargar su batería.
- Capacidad efectora:
  - RandomAgent:
    - Moverse a celdas vecinas (explorar).
    - Limpiar basura en su celda actual.
    - Recargar batería en estaciones.
    - Calcular rutas hacia estaciones de recarga o basura usando algoritmos como Dijkstra.
  - ObstacleAgent: No tiene acciones.
  - TrashAgent: Puede desaparecer cuando es limpiado.
  - RechargeStationAgent: No tiene acciones.
- Percepción:
  - RandomAgent:
    - Detecta el estado de las celdas vecinas (si están vacías, tienen basura, obstáculos o estaciones de recarga).
    - Puede calcular rutas hacia objetivos específicos (basura o estaciones de recarga).
  - Los demás agentes no tienen percepción activa.
- Proactividad:
  - RandomAgent:

        - Decide entre explorar, limpiar, recargar o buscar estaciones de recarga según su estado actual (nivel de batería, presencia de basura, etc.).
    - Los demás agentes son reactivos o estáticos.
- Métricas de desempeño:
    - RandomAgent:
        - Cantidad de basura limpiada.
        - Número de recargas realizadas.
        - Pasos realizados.
    - Modelo:
        - Porcentaje de celdas limpias.
        - Número total de movimientos realizados por los agentes.

La arquitectura de subsunción de los agentes.

- **Nivel 0**
    - Supervivencia de los random agents
    -
```
"""
Determines the new action it will take, and then moves
"""
if self.dead:
    return

if self._battery > 0:
```
    -
```
else:
    self.die()
```
    - Garantizar que las roombas que estén vivas puedan hacer el resto de acciones, mientras que las muertas ya no hagan nada (o bueno si la última roomba viva se muere, se acaba la simulación, si aun hay vivas, nada más se quedan ahí para estorbar a las demás)
- **Nivel 1**
    - Modo de supervivencia (crisis)
    -
```
    # Check if battery is low
    elif self._battery <= 35:
        self.in_crisis = True
        self.crisis()
    else:
        self.explore()
```
    - Si la pila esta en 35 o menos, la roomba se enfoca en volver a una estación de carga que no esté ocupada (aquí se utiliza dijkstra para ver la mejor manera de ir a una estación de carga)
- **Nivel 2**
    - Recarga de energía

- ```
  # Check if on recharge station
  elif any(isinstance(a, RechargeStationAgent) for a in self.cell.agents):
      if self.in_crisis or self._battery < 100:
          self.recharge()
      else:
          self.explore()
  ```
  - Al estar en una estación de carga, el agente se recarga hasta tener 100 % de pila (si por casualidad termina en una estación de carga, igual se carga para aprovechar el hecho de que está ahí).
- **Nivel 3**
  - Limpieza
  - ```
    # Check if on trash
    if any(isinstance(a, TrashAgent) for a in self.cell.agents):
        self.clean()
    ```
  - Si el roomba está en una celda sucia, empieza a limpiar
- **Nivel 4**
  - Exploración
  - ```
    else:
        self.explore()
    ```
  - Si no esta en ningun otra acción, entonces explora
  - La acción de explorar es algo compleja, ya que aquí es donde también se busca basura con un BFS (que utiliza dijkstra hacia una celda en especifico) y sus vecinos.

Cada comportamiento tiene condiciones específicas que ven su activación, su desarrollo (toma de decisiones y algoritmos a utilizar) y su conclusión.

Características del ambiente.

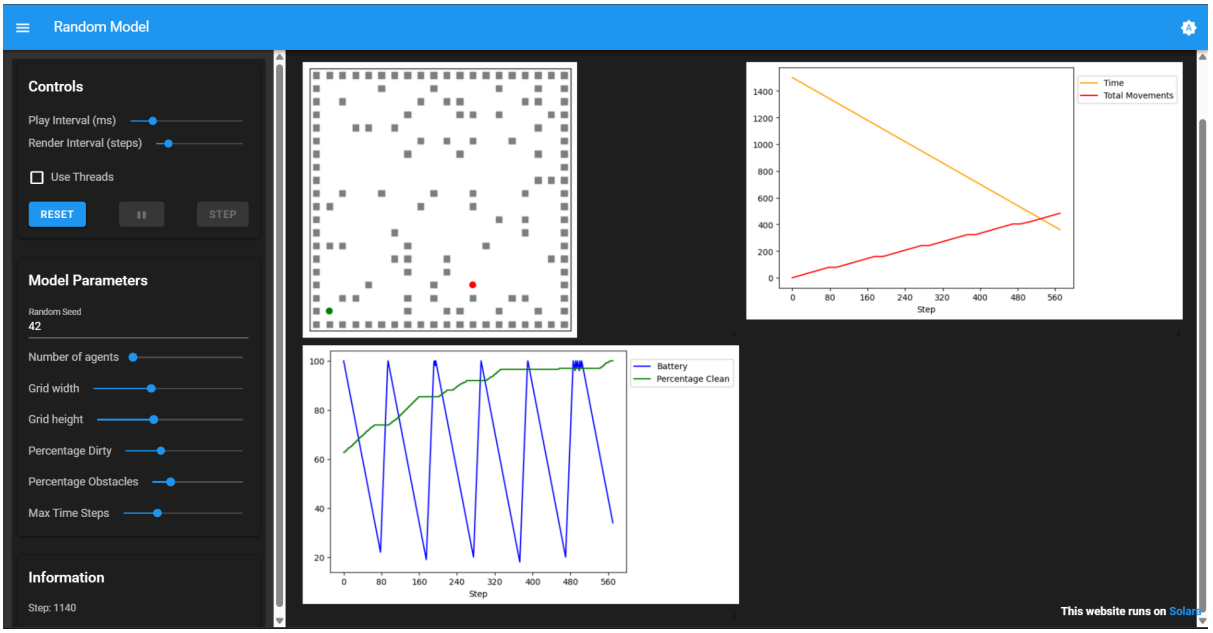
- Parcialmente accesible
  - Las roombas conocen sus vecinos en cada paso y el camino que van recorriendo
- No determinista
  - Los mismos parámetros pueden generar diferentes resultados
- No Episódico/ Secuencial
  - Porque las roombas dependen del paso anterior
- Estático
  - El ambiente solo cambia si la roomba hace una acción (limpiar, mover, morir)
- Discreto
  - Hay un número finito de posibles acciones que se pueden tomar en cada paso
    ```

Las estadísticas recolectadas en las simulaciones.

Simulación 1 (Un roomba)

Caso Normal:

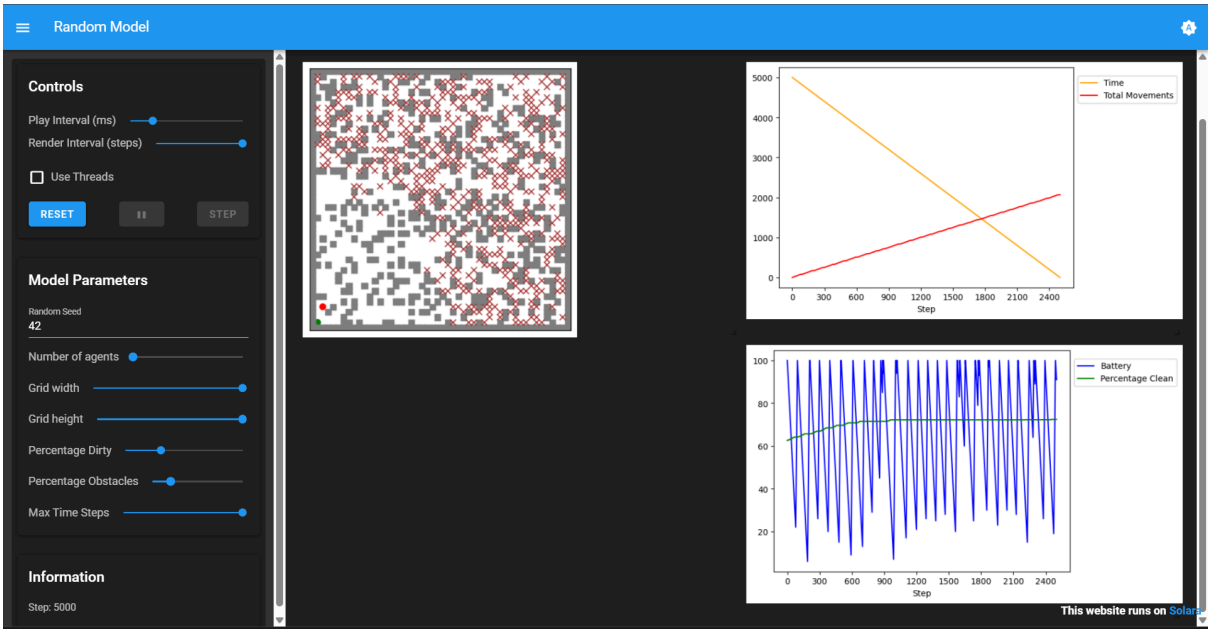
- 1 agente
- 20 celdas de largo
- 20 celdas de altura
- 30 % de celdas sucias
- 20 % de obstáculos
- 1500 pasos máximos



En este caso, sí se logró limpiar todo el grid, pero le tomó mucho tiempo, también hizo muchos movimientos innecesarios, como se ve en la gráfica, esto es porque se queda atorado en ciertas áreas, especialmente si están medio encerradas por obstáculos.

Caso Extremo:

- 1 agente
- 50 celdas de largo
- 50 celdas de altura
- 30 % de celdas sucias
- 20 % de obstáculos
- 5000 pasos máximos

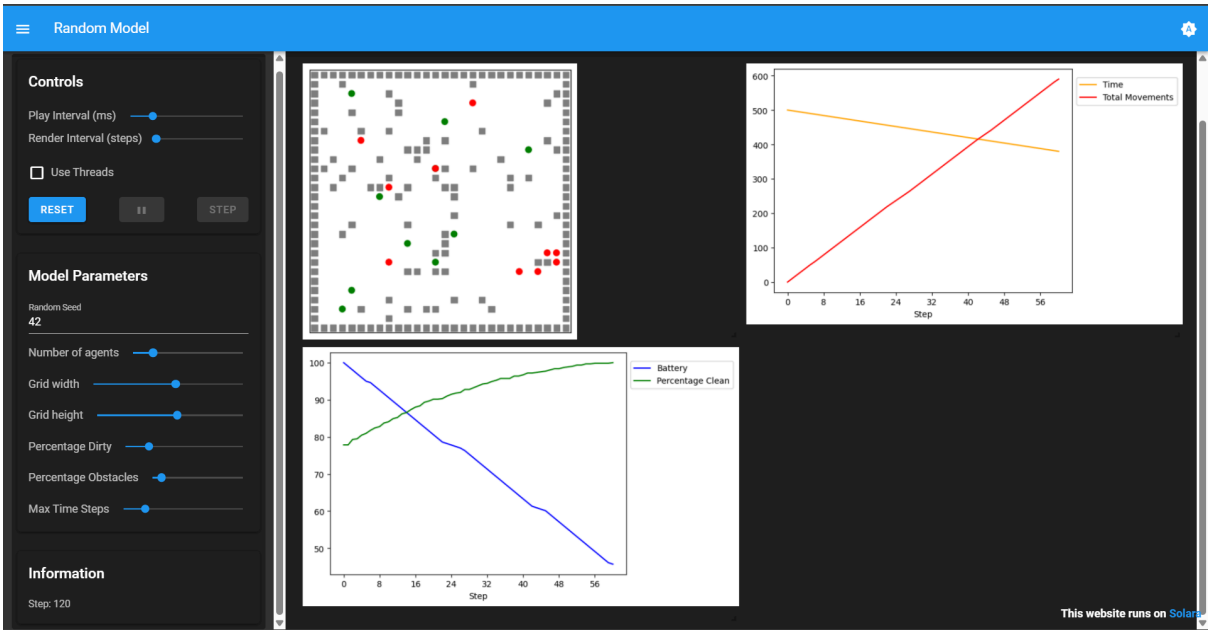


Ya con grids más grandes es mucho más difícil que un roomba limpie todo el grid, y ya que el roomba comete el mismo error que en el ejemplo anterior, se vuelve imposible limpiar toda el área en un tiempo aceptable.

Simulación 2 (Varios roombas)

Caso Normal:

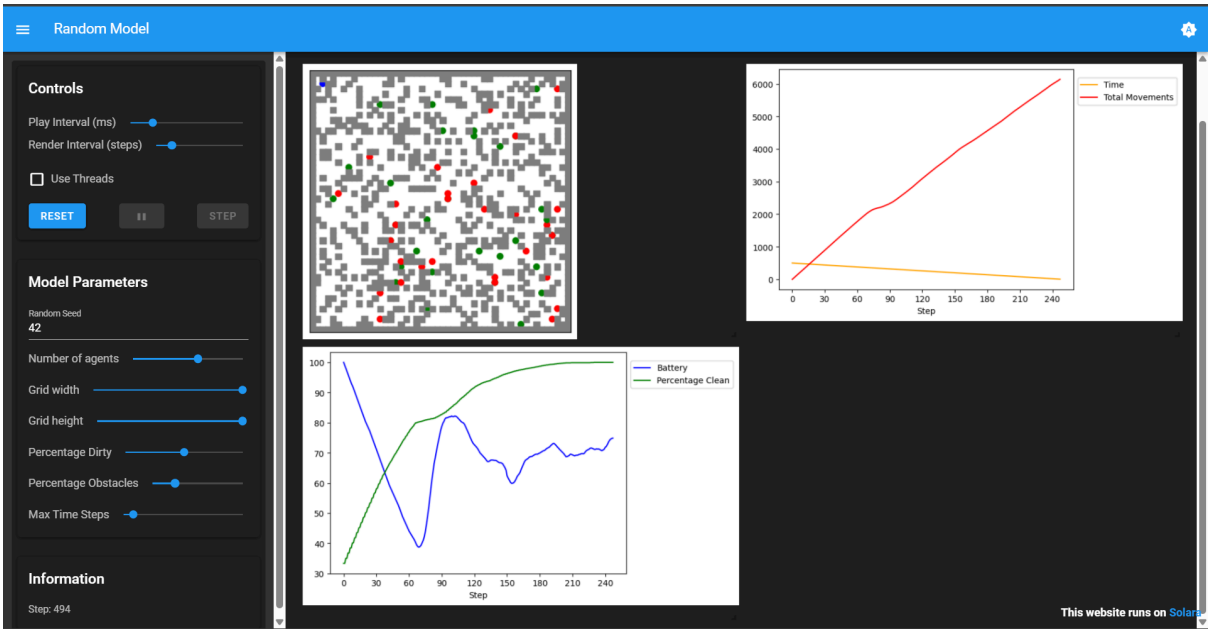
- 10 agentes
- 28 celdas de largo
- 28 celdas de altura
- 20 % de celdas sucias
- 10 % de obstáculos
- 500 pasos máximos



Ya cuando hay más roombas, es más fácil limpiar el grid, especialmente porque pueden cubrir un área más grande (entonces una roomba no tiene que ir hasta el otro extremo del grid para limpiar lo que está allá, sino que otra roomba más cercana se puede encargar de eso).

Caso Extremo:

- 30 agentes
- 50 celdas de largo
- 50 celdas de altura
- 50 % de celdas sucias
- 25 % de obstáculos
- 500 pasos máximos



Incluso con el grid al máximo, todo el área se puede limpiar en un promedio de 500 pasos, a veces más, a veces menos, el error o la situación que se mencionó en la simulación de un roomba también afecta a esta simulación, por lo que podría ser más efectiva.

Conclusiones.

En conclusión, creo que me falta agregar que cuando una roomba se regresa a una estación de carga, vaya a un extremo de las celdas que ya conoce, o que vaya a la celda más lejana de la estación de carga que conozca, aparte que sería bueno usar dijkstra para llegar a esa celda de la mejor manera posible, para así poder llegar más lejos cada viaje.