INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Reporte - M1 Actividad

Equipo: NA **Grupo:** 301

Integrantes:

Diego Vega Camacho - A01704492

Profesor:

Pedro Oscar Pérez Murueta

12 de Noviembre , 2023

class **VaccumAgent (Agent)**

Primero que nada tenemos:

```
    def __init__(self, id, model, x, y):
        super().__init__(id, model)
        self.position = (x, y)
```

Donde inicializamos las propiedades y asignamos las variables. Posteriormente tenemos:

```
def step(self):
```

En donde tenemos la comprobación de la condición de parada:

```
        if model.dirty_cells.sum() == 0:
            model.running = False
```

La obtención de la posición actual:

```
        x_pos = self.position[0]
        y_pos = self.position[1]
```

La limpieza de la celda actual:

```
        if model.dirty_cells[x_pos][y_pos] == 1:
            model.dirty_cells[x_pos][y_pos] = 0
```

Y finalmente el movimiento a una celda aleatoria:

```
        possible_cells = self.model.grid.get_neighborhood(self.position, moore=True,
include_center = False)
        new_position = self.random.choice(possible_cells)
        cellmate = self.model.grid.get_cell_list_contents([new_position])

        if cellmate:
            return

        self.position = new_position
        self.model.grid.move_agent(self, self.position)
```

Este agente se mueve aleatoriamente y limpia cualquier celda en la que se encuentre si está sucia. También, detiene la simulación si no hay más celdas sucias en el entorno.

def **get_vaccums (model)**

```
def get_vaccums(model):
    return np.asarray([agent.position for agent in model.schedule.agents])
```

Está función recopila las posiciones de todos los agentes que son instancias de la clase VaccumAgent dentro del modelo y devuelve estas posiciones en un array NumPy

class **VaccumModel (Model)**

Primero tenemos el método init, el cual inicializa el modelo con los parámetros: width y height (dimensiones del entorno), num_agents (número de agentes vaccum) y dirty_cells_percentage (porcentaje de celdas sucias). Y a su vez se inicializan los parámetros: schedule (orden en el que los agentes realizan sus acciones), grid (representación del entorno), datacollector (recolección de datos del modelo), dirty_cells (matriz de estado de limpieza de celdas), dirty_cells_percentage (porcentaje de celdas sucias) y running (estado de ejecución del entorno)

```
    def __init__(self, width, height, num_agents, dirty_cells_percent):
        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(width, height, torus = False)
        self.datacollector = DataCollector(model_reporters={"vaccums": get_vaccums})
        self.dirty_cells = np.zeros((width, height))
        self.dirty_cells_percent = dirty_cells_percent
        self.running = True
```

De igual forma, el mismo método crea agentes y celdas sucias:

```
        # Create agents and place them in the grid
        for i in range(num_agents):
            x = 1
            y = 1
            agent = VaccumAgent(i, self, x, y)
            self.schedule.add(agent)
            self.grid.place_agent(agent, (x, y))

        # Create dirty cells in the grid
        dirty_cells = 0
        while dirty_cells < int(width * height * (dirty_cells_percent / 100)):
            x = int(np.random.rand() * width)
```

```
            y = int(np.random.rand() * height)
            if self.dirty_cells[x][y] == 0:
                self.dirty_cells[x][y] = 1
                dirty_cells += 1
```

Posteriormente el método step hace que el modelo avance:

```
def step(self):
        if self.running == False:
            return

        self.datacollector.collect(self)
        self.schedule.step()
```

Esta clase define un modelo de simulación donde los agentes tipo VaccumAgent se mueven en un entorno representado por una rejilla, limpian celdas sucias y la simulación avanza en pasos sucesivos.

def **clean_cells (model, iterations, num_vaccums)**

Esta función realiza una serie de iteraciones sobre el modelo de VaccumModel

```
def clean_cells(model, iterations, num_vaccums):
 print ("\n")
 print("Number of vaccums: ", num_vaccums)

 for iteration in iterations:
   time = tm.time()

   print("\n")
   print("Iteration: ", iteration)

   for i in range(iteration):
       model.step()

       if model.running == False:
          break

   dirty_cell = model.dirty_cells.sum()
   dirty_cell_percentage = dirty_cell / (WIDTH * HEIGHT) * 100

   print("Dirty cells: ", dirty_cell)
```

```
    print("Time: ", tm.time() - time)
```

Este mismo ejecuta el avance del modelo en cada iteración y registra el número de celdas sucias que quedan y el tiempo que tarda cada iteración en ejecutarse.

def **clean_all_cells (model, num_vaccums)**

De igual forma, está función ejecuta el modelo de simulación hasta que todas las celdas estén limpias

```
def clean_all_cells(model, num_vaccums):
 time = tm.time()
 steps = 0

 print("\n")
 print("Number of vaccum: ", num_vaccums)

 while model.running == True:
     model.step()
     steps += 1

 print("Steps to clean all cells: ", steps)

 total_time = tm.time() - time
 print("Time: ", total_time)

 return [total_time, steps]
```

Registra el tiempo total que tomó y la cantidad de pasos que se ejecutaron para lograr la limpieza completa del entorno.