

ASIGNMENT: ROBOT_VACUMN

Task 1: Triển khai môi trường mô phỏng

- Mô tả cài đặt:

- Sử dụng mảng 2 chiều để lưu trạng thái sạch/bẩn của từng ô giúp dễ kiểm tra và cập nhật.
- Agent bắt đầu ở vị trí ngẫu nhiên để mô phỏng thực tế.
- Cảm biến va chạm (bumpers) giúp agent không đi ra ngoài phòng.
- Vòng lặp kiểm tra điều kiện dừng khi phòng đã sạch hoặc hết bước để đảm bảo hiệu quả.

- Code:

```
1 def simulation_environment(agent_function, n=5, p=0.2, max_steps=1000, verbose=False):
2     # Initialize room: True = dirty, False = clean
3     room = np.random.rand(n, n) < p
4     # Agent position
5     x, y = random.randint(0, n-1), random.randint(0, n-1)
6     actions_taken = 0
7
8     for step in range(max_steps):
9         # Sensors
10        bumpers = {
11            "north": y == 0,
12            "south": y == n-1,
13            "west": x == 0,
14            "east": x == n-1
15        }
16        dirty = room[y, x]
17        # Agent decides action
18        action = agent_function(bumpers, dirty)
19        actions_taken += 1
20        if verbose:
21            print(f"Step {step}: Pos=({x},{y}) Action={action} Dirty={dirty}")
22        # React to action
23        if action == "suck":
24            room[y, x] = False
25        elif action == "north" and not bumpers["north"]:
26            y -= 1
27        elif action == "south" and not bumpers["south"]:
28            y += 1
29        elif action == "west" and not bumpers["west"]:
30            x -= 1
31        elif action == "east" and not bumpers["east"]:
32            x += 1
33        # Check if all clean
34        if not room.any():
35            break
36    return actions_taken
```

- Giải thích:

- **room = np.random.rand(n, n) < p:** Tạo một ma trận kích thước n x n, mỗi ô có xác suất p là bẩn (True), còn lại là sạch (False).
- **x, y = random.randint(0, n-1), random.randint(0, n-1):** Đặt robot vào vị trí ngẫu nhiên trong phòng.
- **actions_taken = 0:** Biến đếm số hành động đã thực hiện.

- **Vòng lặp for step in range(max_steps):** Mỗi bước, robot sẽ thực hiện một hành động cho đến khi phòng sạch hoặc hết số bước cho phép.
- **bumpers:** Tạo cảm biến va chạm cho 4 hướng. Nếu robot ở sát mép phòng thì cảm biến trả về True (không đi tiếp được).
- **dirty = room[y, x]:** Kiểm tra ô hiện tại có bẩn không.
- **action = agent_function(bumpers, dirty):** Gọi hàm agent để quyết định hành động dựa trên cảm biến.
- **actions_taken += 1:** Tăng số lượng hành động.
- **Nếu action == "suck":** Robot làm sạch ô hiện tại.
- **Nếu robot chọn di chuyển (north/south/west/east) và không bị cản,** cập nhật vị trí tương ứng.
- **if not room.any():** Nếu tất cả các ô đều sạch, kết thúc vòng lặp.
- **return actions_taken:** Trả về tổng số hành động đã thực hiện để đánh giá hiệu năng agent.

- Tổng kết :

Agent này giúp mô phỏng quá trình robot hút bụi hoạt động trong phòng, cung cấp cảm biến, xử lý hành động và đo lường hiệu quả làm sạch của agent, đảm bảo tính khách quan khi đánh giá các chiến lược khác nhau.

- Minh chứng môi trường hoạt động với Agent ngẫu nhiên đơn giản:

```
1 steps = simulation_environment(simple_randomized_agent, n=5, p=0.2, max_steps=1000, verbose=True)
2 print(f"Actions taken to clean the room: {steps}")
```

- Kết quả:

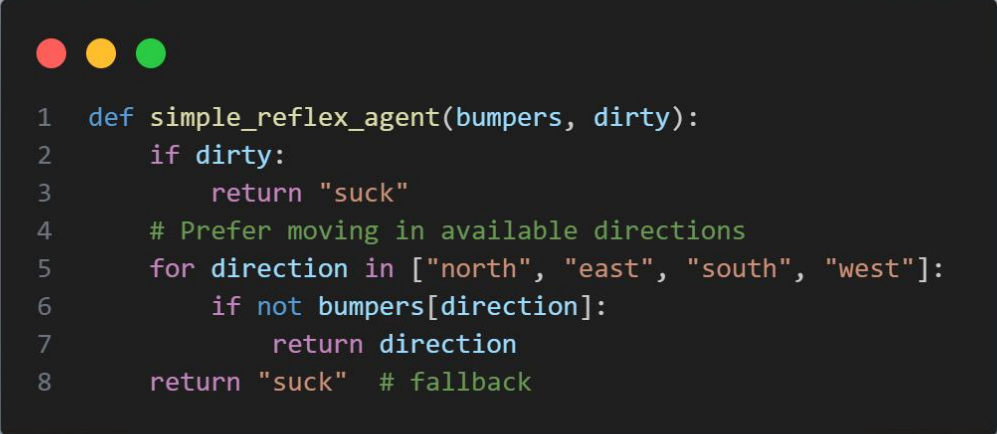
```
Step 0: Pos=(2,3) Action=suck Dirty=False
Step 1: Pos=(2,3) Action=east Dirty=False
Step 2: Pos=(3,3) Action=east Dirty=False
Step 3: Pos=(4,3) Action=west Dirty=True
Step 4: Pos=(3,3) Action=west Dirty=False
Step 5: Pos=(2,3) Action=north Dirty=False
Step 6: Pos=(2,2) Action=suck Dirty=False
Step 7: Pos=(2,2) Action=suck Dirty=False
Step 8: Pos=(2,2) Action=west Dirty=False
Step 9: Pos=(1,2) Action=west Dirty=False
Step 10: Pos=(0,2) Action=south Dirty=True
Step 11: Pos=(0,3) Action=west Dirty=False
Step 12: Pos=(0,3) Action=north Dirty=False
Step 13: Pos=(0,2) Action=south Dirty=True
Step 14: Pos=(0,3) Action=suck Dirty=False
Step 15: Pos=(0,3) Action=east Dirty=False
Step 16: Pos=(1,3) Action=south Dirty=False
Step 17: Pos=(1,4) Action=east Dirty=True
Step 18: Pos=(2,4) Action=south Dirty=False
Step 19: Pos=(2,4) Action=south Dirty=False
Step 20: Pos=(2,4) Action=suck Dirty=False
Step 21: Pos=(2,4) Action=west Dirty=False
Step 22: Pos=(1,4) Action=north Dirty=True
Step 23: Pos=(1,3) Action=west Dirty=False
Step 24: Pos=(0,3) Action=north Dirty=False
...
Step 398: Pos=(4,4) Action=south Dirty=True
Step 399: Pos=(4,4) Action=south Dirty=True
Step 400: Pos=(4,4) Action=suck Dirty=True
Actions taken to clean the room: 401
```

Task 2: Agent phản xạ đơn giản

- Mô tả cài đặt:

- Agent ưu tiên làm sạch nếu ô hiện tại bẩn, vì mục tiêu là làm sạch phòng nhanh nhất.
- Nếu không bẩn, agent chọn hướng bất kỳ không bị cản để di chuyển, tránh lặp lại hành động vô ích.

- Code:



```
1 def simple_reflex_agent(bumpers, dirty):
2     if dirty:
3         return "suck"
4     # Prefer moving in available directions
5     for direction in ["north", "east", "south", "west"]:
6         if not bumpers[direction]:
7             return direction
8     return "suck" # fallback
```

- Giải thích:

- **if dirty: return "suck":** Nếu cảm biến báo ô hiện tại đang bẩn (dirty == True), agent sẽ ưu tiên thực hiện hành động hút bụi ("suck") để làm sạch ngay lập tức. Đây là hành vi hợp lý vì mục tiêu chính là làm sạch phòng càng nhanh càng tốt.
- **for direction in ["north", "east", "south", "west"]: ...** :Nếu ô hiện tại đã sạch, agent sẽ kiểm tra lần lượt các hướng di chuyển: bắc, đông, nam, tây.
- **if not bumpers[direction]: return direction:** Với mỗi hướng, agent kiểm tra cảm biến va chạm (bumpers). Nếu hướng đó không bị cản (giá

trị là False), agent sẽ chọn di chuyển theo hướng đó. Điều này giúp agent tránh va vào tường và luôn ưu tiên di chuyển sang ô mới.

- **return "suck" # fallback:** Nếu tất cả các hướng đều bị cản (agent bị kẹt ở góc), agent sẽ thực hiện lại hành động hút bụi. Đây là phương án dự phòng để đảm bảo agent luôn thực hiện một hành động hợp lệ.

- Tổng kết:

Hàm này mô phỏng một agent phản xạ đơn giản:


Luôn làm sạch nếu ô hiện tại bẩn.

Nếu ô sạch, di chuyển sang hướng bất kỳ không bị cản.

Nếu bị kẹt, tiếp tục hút bụi (dù ô đã sạch).

Nhược điểm duy nhất là: chưa tối ưu vì không ghi nhớ trạng thái phòng hay vị trí đã đi qua, chỉ hoạt động với môi trường đơn giản

- Minh chứng agent phản xạ đơn giản hoạt động với môi trường:



```
1 steps = simulation_environment(simple_reflex_agent, n=5, p=0.2, max_steps=1000, verbose=True)
2 print(f"Actions taken to clean the room: {steps}")
```

- Kết quả:

```
Step 0: Pos=(2,2) Action=north Dirty=False
Step 1: Pos=(2,1) Action=north Dirty=False
Step 2: Pos=(2,0) Action=east Dirty=False
Step 3: Pos=(3,0) Action=east Dirty=False
Step 4: Pos=(4,0) Action=suck Dirty=True
Step 5: Pos=(4,0) Action=south Dirty=False
Step 6: Pos=(4,1) Action=suck Dirty=True
Step 7: Pos=(4,1) Action=north Dirty=False
Step 8: Pos=(4,0) Action=south Dirty=False
Step 9: Pos=(4,1) Action=north Dirty=False
Step 10: Pos=(4,0) Action=south Dirty=False
Step 11: Pos=(4,1) Action=north Dirty=False
Step 12: Pos=(4,0) Action=south Dirty=False
Step 13: Pos=(4,1) Action=north Dirty=False
Step 14: Pos=(4,0) Action=south Dirty=False
Step 15: Pos=(4,1) Action=north Dirty=False
Step 16: Pos=(4,0) Action=south Dirty=False
Step 17: Pos=(4,1) Action=north Dirty=False
Step 18: Pos=(4,0) Action=south Dirty=False
Step 19: Pos=(4,1) Action=north Dirty=False
Step 20: Pos=(4,0) Action=south Dirty=False
Step 21: Pos=(4,1) Action=north Dirty=False
Step 22: Pos=(4,0) Action=south Dirty=False
Step 23: Pos=(4,1) Action=north Dirty=False
Step 24: Pos=(4,0) Action=south Dirty=False
...
Step 997: Pos=(4,1) Action=north Dirty=False
Step 998: Pos=(4,0) Action=south Dirty=False
Step 999: Pos=(4,1) Action=north Dirty=False
Actions taken to clean the room: 1000
```

Task 3: Agent phản xạ dựa trên mô hình

- Mô tả cài đặt:

Agent dựa trên mô hình sẽ lưu lại vị trí hiện tại của mình và trạng thái sạch/bẩn của từng ô trong phòng. Agent sẽ di chuyển theo một quy tắc nhất định (ví dụ: đi theo từng hàng, zig-zag) để đảm bảo quét sạch toàn bộ phòng. Nhờ việc ghi nhớ các ô đã đi qua và đã làm sạch, agent này có thể tối ưu số bước di chuyển và tránh lặp lại các hành động không cần thiết.

- Code:

```
1 class ModelBasedAgent:
2     def __init__(self, n):
3         self.n = n
4         self.x = 0
5         self.y = 0
6         self.direction = "east" # start moving east
7         self.visited = set()
8     def agent_function(self, bumpers, dirty):
9         pos = (self.x, self.y)
10        self.visited.add(pos)
11        if dirty:
12            return "suck"
13        # Zig-zag pattern: move east until wall, then south, then west, then south, repeat
14        if self.direction == "east":
15            if not bumpers["east"]:
16                self.x += 1
17                return "east"
18            elif not bumpers["south"]:
19                self.y += 1
20                self.direction = "west"
21                return "south"
22        elif self.direction == "west":
23            if not bumpers["west"]:
24                self.x -= 1
25                return "west"
26            elif not bumpers["south"]:
27                self.y += 1
28                self.direction = "east"
29                return "south"
30        # If stuck, try any available move
31        for d in ["north", "east", "south", "west"]:
32            if not bumpers[d]:
33                if d == "north": self.y -= 1
34                if d == "south": self.y += 1
35                if d == "east": self.x += 1
36                if d == "west": self.x -= 1
37            return d
38        return "suck"
```


- Giải thích:

- **__init__(self, n):** Khởi tạo agent với kích thước phòng $n \times n$.
 - **self.x, self.y:** Vị trí hiện tại của agent (bắt đầu ở góc trên trái).
 - **self.direction:** Hướng di chuyển hiện tại, khởi đầu là "east" (đi sang phải).
 - **self.visited:** Tập hợp các ô đã đi qua, giúp agent ghi nhớ lịch sử di chuyển.
- **agent_function(self, bumpers, dirty):** Hàm quyết định hành động dựa trên cảm biến và trạng thái agent.
 - **pos = (self.x, self.y):** Lưu vị trí hiện tại vào tập visited.
 - **Nếu ô hiện tại bẩn (dirty == True), agent sẽ hút bụi ("suck").**
 - **Nếu không bẩn, agent di chuyển theo chiến lược zig-zag:**
 - ◆ **Nếu đang đi sang phải (direction == "east"):**
 - **Nếu chưa bị cản phải (not bumpers["east"]),** di chuyển sang phải.
 - **Nếu bị cản phải nhưng chưa bị cản dưới (not bumpers["south"]),** di chuyển xuống dưới và đổi hướng sang trái.
 - ◆ **Nếu đang đi sang trái (direction == "west"):**
 - **Nếu chưa bị cản trái (not bumpers["west"]),** di chuyển sang trái.
 - **Nếu bị cản trái nhưng chưa bị cản dưới (not bumpers["south"]),** di chuyển xuống dưới và đổi hướng sang phải.

- **Nếu bị kẹt (mọi hướng đều bị cản),** agent thử di chuyển theo bất kỳ hướng nào không bị cản (ưu tiên theo thứ tự north, east, south, west).
- **Nếu vẫn không di chuyển được,** agent thực hiện hút bụi ("suck") như phương án dự phòng.

- Ý nghĩa chiến lược zig-zag:

Agent sẽ đi hết một hàng, xuống hàng tiếp theo, rồi đi ngược lại, lặp lại cho đến khi hết phòng.

Nhờ ghi nhớ vị trí và hướng, agent tránh lặp lại các ô đã đi qua, tối ưu hóa số bước di chuyển.

- Tổng kết:

Agent này thông minh hơn agent phản xạ đơn giản vì có khả năng ghi nhớ và lập kế hoạch di chuyển.

Phù hợp với phòng lớn, hình dạng phức tạp, hoặc có vật cản.

Đảm bảo mọi ô đều được kiểm tra và làm sạch, giảm thiểu số bước thừa.

- Minh chứng Agent dựa theo mô hình hoạt động với môi trường:

```
1 agent = ModelBasedAgent(n=5)
2 def agent_func(bumpers, dirty):
3     return agent.agent_function(bumpers, dirty)
4 steps = simulation_environment(agent_func, n=5, p=0.2, max_steps=1000, verbose=True)
5 print(f"Actions taken to clean the room: {steps}")
```

- Kết quả:

```
Step 0: Pos=(4,3) Action=south Dirty=False
Step 1: Pos=(4,4) Action=suck Dirty=True
Step 2: Pos=(4,4) Action=west Dirty=False
Step 3: Pos=(3,4) Action=suck Dirty=True
Step 4: Pos=(3,4) Action=west Dirty=False
Step 5: Pos=(2,4) Action=west Dirty=False
Step 6: Pos=(1,4) Action=west Dirty=False
Step 7: Pos=(0,4) Action=north Dirty=False
Step 8: Pos=(0,3) Action=south Dirty=False
Step 9: Pos=(0,4) Action=east Dirty=False
Step 10: Pos=(1,4) Action=east Dirty=False
Step 11: Pos=(2,4) Action=east Dirty=False
Step 12: Pos=(3,4) Action=east Dirty=False
Step 13: Pos=(4,4) Action=north Dirty=False
Step 14: Pos=(4,3) Action=south Dirty=False
Step 15: Pos=(4,4) Action=west Dirty=False
Step 16: Pos=(3,4) Action=west Dirty=False
Step 17: Pos=(2,4) Action=west Dirty=False
Step 18: Pos=(1,4) Action=west Dirty=False
Step 19: Pos=(0,4) Action=north Dirty=False
Step 20: Pos=(0,3) Action=south Dirty=False
Step 21: Pos=(0,4) Action=east Dirty=False
Step 22: Pos=(1,4) Action=east Dirty=False
Step 23: Pos=(2,4) Action=east Dirty=False
Step 24: Pos=(3,4) Action=east Dirty=False
...
Step 997: Pos=(4,4) Action=north Dirty=False
Step 998: Pos=(4,3) Action=south Dirty=False
Step 999: Pos=(4,4) Action=west Dirty=False
Actions taken to clean the room: 1000
```

Task 4: So sánh hiệu năng các agent

- Mô tả:

- Sử dụng pandas và matplotlib để trực quan hóa kết quả, giúp dễ dàng nhận xét và so sánh.
- Chạy nhiều lần (100 lần) để lấy kết quả trung bình, giảm ảnh hưởng của ngẫu nhiên.

- Code:

```
1 def evaluate_agent(agent_func, n, p=0.2, runs=100):
2     results = [simulation_environment(agent_func, n=n, p=p, max_steps=10000) for _ in range(runs)]
3     return np.mean(results)
4
5 sizes = [5, 10, 100]
6 results = {"Randomized": [], "Simple Reflex": [], "Model-Based": []}
7
8 for size in sizes:
9     # Randomized agent
10    results["Randomized"].append(evaluate_agent(simple_randomized_agent, size))
11    # Simple reflex agent
12    results["Simple Reflex"].append(evaluate_agent(simple_reflex_agent, size))
13    # Model-based agent
14    agent = ModelBasedAgent(n=size)
15    def agent_func(bumpers, dirty):
16        return agent.agent_function(bumpers, dirty)
17    results["Model-Based"].append(evaluate_agent(agent_func, size))
18
19 # Create DataFrame for table
20 df = pd.DataFrame(results, index=[f"{s}x{s}" for s in sizes])
21 print(df)
```

- Giải thích:

1. Hàm evaluate_agent

```
1 def evaluate_agent(agent_func, n, p=0.2, runs=100):
2     results = [simulation_environment(agent_func, n=n, p=p, max_steps=10000) for _ in range(runs)]
3     return np.mean(results)
```

- **Mục đích:** Chạy thử nghiệm agent nhiều lần (mặc định 100 lần) trên phòng kích thước $n \times n$, mỗi lần khởi tạo phòng và vị trí agent ngẫu nhiên.
- **Kết quả:** Trả về giá trị trung bình số hành động cần thiết để làm sạch phòng (hiệu năng agent).

2. Khởi tạo các biến

```
1 sizes = [5, 10, 100]
2 results = {"Randomized": [], "Simple Reflex": [], "Model-Based": []}
```

sizes: Danh sách các kích thước phòng cần đánh giá (5x5, 10x10, 100x100).

results: Tạo dictionary để lưu kết quả cho từng loại agent.

3. Vòng lặp đánh giá từng agent trên từng kích thước phòng

```
1 for size in sizes:
2     # Randomized agent
3     results["Randomized"].append(evaluate_agent(simple_randomized_agent, size))
4     # Simple reflex agent
5     results["Simple Reflex"].append(evaluate_agent(simple_reflex_agent, size))
6     # Model-based agent
7     agent = ModelBasedAgent(n=size)
8     def agent_func(bumpers, dirty):
9         return agent.agent_function(bumpers, dirty)
10    results["Model-Based"].append(evaluate_agent(agent_func, size))
11
```

- **Randomized agent:** Đánh giá agent chọn hành động ngẫu nhiên.
- **Simple reflex agent:** Đánh giá agent phản xạ đơn giản (ưu tiên hút bụi, nếu sạch thì di chuyển).

- **Model-based agent:** Đánh giá agent dựa trên mô hình (ghi nhớ vị trí, di chuyển zig-zag).

4. Tạo bảng kết quả bằng pandas

```
1 df = pd.DataFrame(results, index=[f"{s}x{s}" for s in sizes])
2 print(df)
```

- **Tạo DataFrame:** Biến kết quả thành bảng, mỗi dòng là một kích thước phòng, mỗi cột là một loại agent.
- **In bảng:** Hiển thị số hành động trung bình của từng agent trên từng kích thước phòng.

- Kết quả:

	Randomized	Simple Reflex	Model-Based
5x5	431.43	9900.01	7903.94
10x10	2910.60	10000.00	9505.57
100x100	10000.00	10000.00	10000.00

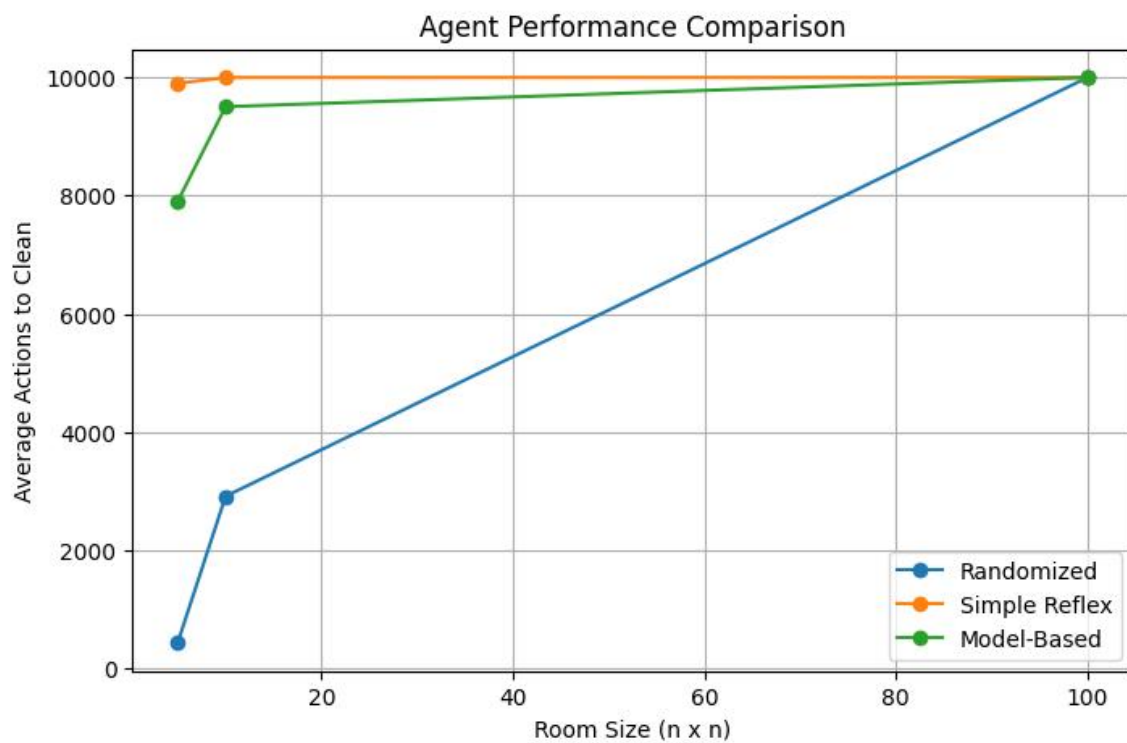
- Agent dựa trên mô hình (Model-Based) thường hiệu quả nhất, đặc biệt khi kích thước phòng tăng lên.
- Agent ngẫu nhiên (Randomized) kém hiệu quả nhất do không có chiến lược rõ ràng.
- Agent phản xạ đơn giản (Simple Reflex) tốt hơn ngẫu nhiên nhưng vẫn có thể lãng phí hành động.

Nhận xét: Khi phòng càng lớn, sự khác biệt về hiệu quả giữa agent dựa trên mô hình và các loại agent khác càng rõ rệt, cho thấy vai trò của việc lập kế hoạch và ghi nhớ trạng thái.

- Ngoài ra, còn có đồ thị biểu diễn:

Code:

```
1 plt.figure(figsize=(8,5))
2 for agent_type in results:
3     plt.plot(sizes, results[agent_type], label=agent_type, marker='o')
4 plt.xlabel('Room Size (n x n)')
5 plt.ylabel('Average Actions to Clean')
6 plt.title('Agent Performance Comparison')
7 plt.legend()
8 plt.grid(True)
9 plt.show()
```



Task 5: Độ bền vững

-Giải thích về độ bền vững của các agent

1. Nếu phòng có kích thước không xác định:

- Agent ngẫu nhiên và phản xạ đơn giản vẫn hoạt động nhưng hiệu quả thấp, dễ bỏ sót hoặc lặp lại các bước.
- Agent dựa trên mô hình có thể khám phá phòng, ghi nhớ các vị trí đã đi qua, xây dựng bản đồ dần dần nên thích nghi tốt hơn.

2. Nếu khu vực làm sạch có hình dạng bất thường:

- Agent ngẫu nhiên và phản xạ đơn giản dễ bỏ sót các khu vực xa hoặc lãng phí hành động.
- Agent dựa trên mô hình nếu ghi nhớ các ô đã đi qua sẽ thích nghi tốt, đảm bảo làm sạch toàn bộ khu vực.

3. Nếu phòng có vật cản:

- Tất cả agent cần xử lý cảm biến va chạm. Agent ngẫu nhiên và phản xạ đơn giản có thể bị kẹt hoặc va chạm lặp lại.
- Agent dựa trên mô hình có thể ghi nhớ vị trí vật cản để tránh lặp lại va chạm, di chuyển tối ưu hơn.

4. Nếu cảm biến bụi không hoàn hảo (10% sai lệch):

- Agent có thể bỏ sót ô bẩn hoặc làm sạch ô đã sạch. Agent ngẫu nhiên và phản xạ đơn giản không có cơ chế kiểm tra lại.
- Agent dựa trên mô hình có thể kiểm tra lại các ô nghi ngờ nhiều lần để giảm số ô còn bẩn.

5. Nếu cảm biến va chạm không hoàn hảo (10% sai lệch):

- Agent có thể thử di chuyển vào vị trí không hợp lệ, dễ bị kẹt hoặc lãng phí hành động.

- Agent dựa trên mô hình có thể học từ các lần di chuyển thất bại, ghi nhớ vị trí thực tế của tường/vật cản để tránh lặp lại lỗi.

Advanced: Mô phỏng cảm biến bụi không hoàn hảo

Code:

```

1  def simulation_environment_imperfect_dirt(agent_function, n=5, p=0.2, max_steps=1000, verbose=False):
2      room = np.random.rand(n, n) < p
3      x, y = random.randint(0, n-1), random.randint(0, n-1)
4      actions_taken = 0
5      for step in range(max_steps):
6          bumpers = {
7              "north": y == 0,
8              "south": y == n-1,
9              "west": x == 0,
10             "east": x == n-1
11         }
12         # 10% chance dirt sensor is wrong
13         true_dirty = room[y, x]
14         if random.random() < 0.1:
15             dirty = not true_dirty
16         else:
17             dirty = true_dirty
18         action = agent_function(bumpers, dirty)
19         actions_taken += 1
20         if verbose:
21             print(f"Step {step}: Pos=({x},{y}) Action={action} Dirty={dirty}")
22         if action == "suck":
23             room[y, x] = False
24         elif action == "north" and not bumpers["north"]:
25             y -= 1
26         elif action == "south" and not bumpers["south"]:
27             y += 1
28         elif action == "west" and not bumpers["west"]:
29             x -= 1
30         elif action == "east" and not bumpers["east"]:
31             x += 1
32         if not room.any():
33             break
34     return actions_taken, np.sum(room) # actions taken, uncleaned squares

```

Giải thích:

- Mỗi lần agent kiểm tra cảm biến bụi, có 10% khả năng nhận kết quả sai (nếu ô đang bẩn thì báo sạch, nếu sạch thì báo bẩn).
- Agent thực hiện hành động dựa trên cảm biến này, có thể bỏ sót ô bẩn hoặc làm sạch ô đã sạch.

- Hàm trả về số hành động đã thực hiện và số ô còn bẩn sau khi kết thúc (uncleaned squares).

Ý nghĩa:

- Kết quả cho thấy khi cảm biến không hoàn hảo, số ô còn bẩn có thể tăng lên nếu agent không có chiến lược kiểm tra lại các ô nghi ngờ.
- Agent dựa trên mô hình có thể cải tiến bằng cách kiểm tra lại các ô đã đi qua nhiều lần để giảm số ô còn bẩn, còn agent đơn giản sẽ dễ bỏ sót hoặc lãng phí hành động.

More Advanced Implementation

Ý tưởng nâng cao

- 1. Vật cản:** Có thể thêm một ma trận boolean để đánh dấu các ô là vật cản. Agent cần kiểm tra cảm biến va chạm và ghi nhớ vị trí các vật cản để tránh lặp lại va chạm.
- 2. Agent khám phá phòng không biết kích thước:** Agent sẽ di chuyển theo một quy tắc (ví dụ: luôn đi đến ô chưa được kiểm tra gần nhất, giống như tìm kiếm theo chiều sâu). Agent cần lưu lại các ô đã đi qua và xây dựng bản đồ phòng dần dần.
- 3. Agent dựa trên utility:** Mỗi ô có xác suất bị bẩn lại, agent cần học các xác suất này và ưu tiên làm sạch các ô có xác suất cao. Utility của trạng thái là số ô sạch hiện tại. Agent sẽ tối ưu hóa utility trong một số bước giới hạn.

- Code:

```
1 def simulation_environment_with_obstacles(agent_function, n=5, p=0.2, obstacle_p=0.1, max_steps=1000, verbose=False):
2     room = np.random.rand(n, n) < p
3     obstacles = np.random.rand(n, n) < obstacle_p
4     x, y = random.randint(0, n-1), random.randint(0, n-1)
5     actions_taken = 0
6     for step in range(max_steps):
7         bumpers = {
8             "north": y == 0 or obstacles[y-1, x] if y > 0 else False,
9             "south": y == n-1 or obstacles[y+1, x] if y < n-1 else False,
10            "west": x == 0 or obstacles[y, x-1] if x > 0 else False,
11            "east": x == n-1 or obstacles[y, x+1] if x < n-1 else False
12        }
13        dirty = room[y, x]
14        action = agent_function(bumpers, dirty)
15        actions_taken += 1
16        if verbose:
17            print(f"Step {step}: Pos=({x},{y}) Action={action} Dirty={dirty}")
18        if action == "suck":
19            room[y, x] = False
20        elif action == "north" and not bumpers["north"]:
21            y -= 1
22        elif action == "south" and not bumpers["south"]:
23            y += 1
24        elif action == "west" and not bumpers["west"]:
25            x -= 1
26        elif action == "east" and not bumpers["east"]:
27            x += 1
28        if not room.any():
29            break
30    return actions_taken
```

Giải thích:

- Mỗi ô trong phòng có xác suất `obstacle_p` trở thành vật cản, agent không thể đi qua các ô này.
- Khi agent kiểm tra cảm biến va chạm (bumper), nếu hướng di chuyển gặp tường hoặc vật cản thì cảm biến báo `True`, agent không thể di chuyển theo hướng đó.
- Agent thực hiện hành động như bình thường, nhưng sẽ bị hạn chế bởi các vật cản, có thể bị kẹt hoặc phải đi vòng.
- Hàm trả về số hành động đã thực hiện để làm sạch phòng.

Ý nghĩa:

- Môi trường có vật cản làm tăng độ khó cho agent, đòi hỏi agent phải có chiến lược ghi nhớ vị trí vật cản để tránh lặp lại va chạm và tối ưu hóa đường đi.

- Agent đơn giản dễ bị kẹt hoặc lãng phí hành động, agent dựa trên mô hình sẽ hiệu quả hơn nếu biết ghi nhớ và tránh các vị trí bị cản.