```python
import random
import time

class ThreeRoomVacuumEnvironment:
    def __init__(self):
        self.rooms = ['A', 'B', 'C']
        self.room_status = {room: random.choice(['Dirty', 'Clean']) for room in self.rooms}
        self.agent_location = random.choice(self.rooms)
        self.score = 0
        self.time_elapsed = 0

    def display(self):
        print(f"Time: {self.time_elapsed}s | Score: {self.score}")
        for room in self.rooms:
            status = "Dirty" if self.room_status[room] == 'Dirty' else "Clean"
            agent = "*" if room == self.agent_location else " "
            print(f"[{room}]{agent} {status}")
        print("-------------------------")

    def update_score(self, action):
        # Scoring rules:
        # -1 for moving
        # +25 for cleaning
        # -10 per dirty room (penalty for leaving rooms dirty)
        if action == "Move":
            self.score -= 1
        elif action == "Clean":
            self.score += 25

        # Penalty for dirty rooms
        dirty_count = sum(1 for status in self.room_status.values() if status == 'Dirty')
        self.score -= dirty_count * 10

    def step(self):
        # Agent perceives current room status
        current_room_status = self.room_status[self.agent_location]

        # Agent decides action
        if current_room_status == 'Dirty':
            action = "Clean"
            self.room_status[self.agent_location] = 'Clean'
        else:
            action = "Move"
            # Move to a random adjacent room (for simplicity)
            adjacent_rooms = [r for r in self.rooms if r != self.agent_location]
            self.agent_location = random.choice(adjacent_rooms)

        # Update score based on action
        self.update_score(action)
        self.time_elapsed += 1
        self.display()
        time.sleep(1)  # Pause for 1 second between actions

# Run the environment
def run_three_room_environment(steps=20):
    env = ThreeRoomVacuumEnvironment()
    print("Initial State:")
    env.display()
    for _ in range(steps):
        env.step()
    print(f"Final Score: {env.score}")

run_three_room_environment()
```

```python
import random
import time
from collections import deque

class NRoomVacuumEnvironment:
    def __init__(self, n=4):
        if n < 2:
            raise ValueError("Number of rooms must be at least 2")

        self.rooms = [chr(65 + i) for i in range(n)]  # Room names: A, B, C, ...
        self.room_status = {room: random.choice(['Dirty', 'Clean']) for room in self.rooms}
        self.agent_location = random.choice(self.rooms)
        self.score = 0
        self.time_elapsed = 0
        self.visited = deque(maxlen=n)  # Track recently visited rooms to avoid ping-pong

    def display(self):
        print(f"Time: {self.time_elapsed}s | Score: {self.score}")
        for room in self.rooms:
            status = "Dirty" if self.room_status[room] == 'Dirty' else "Clean"
            agent = "*" if room == self.agent_location else " "
            print(f"[{room}]{agent} {status}")
        print("-------------------------")

    def update_score(self, action):
        # Scoring rules:
        # -1 for moving
        # +25 for cleaning
        # -10 per dirty room (penalty for leaving rooms dirty)
        if action == "Move":
            self.score -= 1
        elif action == "Clean":
            self.score += 25

        # Penalty for dirty rooms
        dirty_count = sum(1 for status in self.room_status.values() if status == 'Dirty')
        self.score -= dirty_count * 10

    def step(self):
        # Agent perceives current room status
        current_room_status = self.room_status[self.agent_location]

        # Agent decides action
        if current_room_status == 'Dirty':
            action = "Clean"
            self.room_status[self.agent_location] = 'Clean'
        else:
            action = "Move"
            # Move to the dirtiest adjacent room not recently visited
            candidates = []
            for room in self.rooms:
                if room != self.agent_location and room not in self.visited:
                    candidates.append((self.room_status[room], room))

            if not candidates:  # All rooms recently visited
                candidates = [(self.room_status[room], room)
                              for room in self.rooms if room != self.agent_location]

            # Prefer dirtier rooms
            candidates.sort(reverse=True)  # 'Dirty' > 'Clean'
            self.visited.append(self.agent_location)
            self.agent_location = candidates[0][1]

        # Update score based on action
        self.update_score(action)
        self.time_elapsed += 1
```

```
        self.display()
        time.sleep(1)  # Pause for 1 second between actions

# Run the environment
def run_n_room_environment(n=4, steps=30):
    env = NRoomVacuumEnvironment(n)
    print(f"Initial State ({n} rooms):")
    env.display()
    for _ in range(steps):
        env.step()
    print(f"Final Score: {env.score}")

run_n_room_environment(n=5)  # Example with 5 rooms

class RationalNRoomVacuumEnvironment(NRoomVacuumEnvironment):
    def __init__(self, n=4):
        super().__init__(n)
        self.clean_cycles = 0

    def step(self):
        # Check if all rooms are clean
        if all(status == 'Clean' for status in self.room_status.values()):
            self.clean_cycles += 1
            if self.clean_cycles >= len(self.rooms):
                print("All rooms clean for a full cycle. Agent stops.")
                return False
        else:
            self.clean_cycles = 0

        # Original step logic
        current_room_status = self.room_status[self.agent_location]

        if current_room_status == 'Dirty':
            action = "Clean"
            self.room_status[self.agent_location] = 'Clean'
        else:
            action = "Move"
            candidates = []
            for room in self.rooms:
                if room != self.agent_location and room not in self.visited:
                    candidates.append((self.room_status[room], room))

            if not candidates:
                candidates = [(self.room_status[room], room)
                              for room in self.rooms if room != self.agent_location]

            candidates.sort(reverse=True)
            self.visited.append(self.agent_location)
            self.agent_location = candidates[0][1]

        self.update_score(action)
        self.time_elapsed += 1
        self.display()
        time.sleep(1)
        return True

def run_rational_agent(n=4, max_steps=50):
    env = RationalNRoomVacuumEnvironment(n)
    print(f"Initial State ({n} rooms):")
    env.display()
    for _ in range(max_steps):
        if not env.step():
            break
    print(f"Final Score: {env.score}")

run_rational_agent(n=4)
```

```python
import random
import time
from collections import deque

class NRoomVacuumEnvironment:
    def __init__(self, n=4):
        if n < 2:
            raise ValueError("Number of rooms must be at least 2")

        self.rooms = [chr(65 + i) for i in range(n)]  # Room names: A, B, C, ...
        self.room_status = {room: random.choice(['Dirty', 'Clean']) for room in self.rooms}
        self.agent_location = random.choice(self.rooms)
        self.score = 0
        self.time_elapsed = 0
        self.visited = deque(maxlen=n)  # Track recently visited rooms to avoid ping-pong

    def display(self):
        print(f"Time: {self.time_elapsed}s | Score: {self.score}")
        for room in self.rooms:
            status = "Dirty" if self.room_status[room] == 'Dirty' else "Clean"
            agent = "*" if room == self.agent_location else " "
            print(f"[{room}]{agent} {status}")
        print("-------------------------")

    def update_score(self, action):
        # Scoring rules:
        # -1 for moving
        # +25 for cleaning
        # -10 per dirty room (penalty for leaving rooms dirty)
        if action == "Move":
            self.score -= 1
        elif action == "Clean":
            self.score += 25

        # Penalty for dirty rooms
        dirty_count = sum(1 for status in self.room_status.values() if status == 'Dirty')
        self.score -= dirty_count * 10

    def step(self):
        # Agent perceives current room status
        current_room_status = self.room_status[self.agent_location]

        # Agent decides action
        if current_room_status == 'Dirty':
            action = "Clean"
            self.room_status[self.agent_location] = 'Clean'
        else:
            action = "Move"
            # Move to the dirtiest adjacent room not recently visited
            candidates = []
            for room in self.rooms:
                if room != self.agent_location and room not in self.visited:
                    candidates.append((self.room_status[room], room))

            if not candidates:  # All rooms recently visited
                candidates = [(self.room_status[room], room)
                              for room in self.rooms if room != self.agent_location]

            # Prefer dirtier rooms
            candidates.sort(reverse=True)  # 'Dirty' > 'Clean'
            self.visited.append(self.agent_location)
            self.agent_location = candidates[0][1]

        # Update score based on action
        self.update_score(action)
        self.time_elapsed += 1
```

```
        self.display()
        time.sleep(1)  # Pause for 1 second between actions

# Run the environment
def run_n_room_environment(n=4, steps=30):
    env = NRoomVacuumEnvironment(n)
    print(f"Initial State ({n} rooms):")
    env.display()
    for _ in range(steps):
        env.step()
    print(f"Final Score: {env.score}")

run_n_room_environment(n=5)  # Example with 5 rooms

class RationalNRoomVacuumEnvironment(NRoomVacuumEnvironment):
    def __init__(self, n=4):
        super().__init__(n)
        self.clean_cycles = 0

    def step(self):
        # Check if all rooms are clean
        if all(status == 'Clean' for status in self.room_status.values()):
            self.clean_cycles += 1
            if self.clean_cycles >= len(self.rooms):
                print("All rooms clean for a full cycle. Agent stops.")
                return False
        else:
            self.clean_cycles = 0

        # Original step logic
        current_room_status = self.room_status[self.agent_location]

        if current_room_status == 'Dirty':
            action = "Clean"
            self.room_status[self.agent_location] = 'Clean'
        else:
            action = "Move"
            candidates = []
            for room in self.rooms:
                if room != self.agent_location and room not in self.visited:
                    candidates.append((self.room_status[room], room))

            if not candidates:
                candidates = [(self.room_status[room], room)
                              for room in self.rooms if room != self.agent_location]

            candidates.sort(reverse=True)
            self.visited.append(self.agent_location)
            self.agent_location = candidates[0][1]

        self.update_score(action)
        self.time_elapsed += 1
        self.display()
        time.sleep(1)
        return True

def run_rational_agent(n=4, max_steps=50):
    env = RationalNRoomVacuumEnvironment(n)
    print(f"Initial State ({n} rooms):")
    env.display()
    for _ in range(max_steps):
        if not env.step():
            break
    print(f"Final Score: {env.score}")

run_rational_agent(n=4)
```

```python
import random
import time
from collections import deque

class NRoomVacuumEnvironment:
    def __init__(self, n=4):
        if n < 2:
            raise ValueError("Number of rooms must be at least 2")

        self.rooms = [chr(65 + i) for i in range(n)]  # Room names: A, B, C, ...
        self.room_status = {room: random.choice(['Dirty', 'Clean']) for room in self.rooms}
        self.agent_location = random.choice(self.rooms)
        self.score = 0
        self.time_elapsed = 0
        self.visited = deque(maxlen=n)  # Track recently visited rooms to avoid ping-pong

    def display(self):
        print(f"Time: {self.time_elapsed}s | Score: {self.score}")
        for room in self.rooms:
            status = "Dirty" if self.room_status[room] == 'Dirty' else "Clean"
            agent = "*" if room == self.agent_location else " "
            print(f"[{room}]{agent} {status}")
        print("-------------------------")

    def update_score(self, action):
        # Scoring rules:
        # -1 for moving
        # +25 for cleaning
        # -10 per dirty room (penalty for leaving rooms dirty)
        if action == "Move":
            self.score -= 1
        elif action == "Clean":
            self.score += 25

        # Penalty for dirty rooms
        dirty_count = sum(1 for status in self.room_status.values() if status == 'Dirty')
        self.score -= dirty_count * 10

    def step(self):
        # Agent perceives current room status
        current_room_status = self.room_status[self.agent_location]

        # Agent decides action
        if current_room_status == 'Dirty':
            action = "Clean"
            self.room_status[self.agent_location] = 'Clean'
        else:
            action = "Move"
            # Move to the dirtiest adjacent room not recently visited
            candidates = []
            for room in self.rooms:
                if room != self.agent_location and room not in self.visited:
                    candidates.append((self.room_status[room], room))

            if not candidates:  # All rooms recently visited
                candidates = [(self.room_status[room], room)
                              for room in self.rooms if room != self.agent_location]

            # Prefer dirtier rooms
            candidates.sort(reverse=True)  # 'Dirty' > 'Clean'
            self.visited.append(self.agent_location)
            self.agent_location = candidates[0][1]

        # Update score based on action
        self.update_score(action)
        self.time_elapsed += 1
```

```
        self.display()
        time.sleep(1)  # Pause for 1 second between actions

# Run the environment
def run_n_room_environment(n=4, steps=30):
    env = NRoomVacuumEnvironment(n)
    print(f"Initial State ({n} rooms):")
    env.display()
    for _ in range(steps):
        env.step()
    print(f"Final Score: {env.score}")

run_n_room_environment(n=5)  # Example with 5 rooms

class ReflexVacuumAgent:
    def __init__(self, room_count):
        self.room_count = room_count
        self.model = {chr(65 + i): 'Unknown' for i in range(room_count)}  # Internal model
        self.last_action = None
        self.location = None

    def decide_action(self, current_room, current_status):
        self.location = current_room
        self.model[current_room] = current_status

        # Reflex rules
        if current_status == 'Dirty':
            return 'Clean'
        else:
            # Use model to decide where to go next
            # Prioritize rooms marked as dirty in model
            for room, status in self.model.items():
                if room != current_room and status == 'Dirty':
                    return f"Move to {room}"

            # If no known dirty rooms, explore randomly
            return f"Move to {random.choice([r for r in self.model.keys() if r != current_room])}"

class ReflexVacuumEnvironment(NRoomVacuumEnvironment):
    def __init__(self, n=4):
        super().__init__(n)
        self.agent = ReflexVacuumAgent(n)

    def step(self):
        current_room = self.agent_location
        current_status = self.room_status[current_room]

        action = self.agent.decide_action(current_room, current_status)

        if action == 'Clean':
            self.room_status[current_room] = 'Clean'
        elif action.startswith('Move to'):
            new_room = action.split()[-1]
            self.agent_location = new_room

        self.update_score(action.split()[0])
        self.time_elapsed += 1
        self.display()
        time.sleep(1)
        return True

def run_reflex_agent(n=4, steps=30):
    env = ReflexVacuumEnvironment(n)
    print(f"Reflex Agent with Model ({n} rooms):")
    env.display()
    for _ in range(steps):
```

```
        if not env.step():
            break
    print(f"Final Score: {env.score}")

run_reflex_agent(n=4)
```