MALAWI UNIVERSITY OF BUSINESS AND APPLIED SCIENCES (MUBAS)

SCHOOL OF APPLIED SCIENCES
DEPARTMENT OF COMPUTING AND INFORMATION TECHNOLOGY

Presented to:

Dr A. Taylor

Presented by:

DANIEL MSAMILA(BIT/22/SS/020)
VITUMBIKO KAMANGA (BIT/22/SS/012)
ABDUL CARRIM LAIBU (BIT/22/SS/017)
CALEB PHIRI (BIT/22/SS/029)

Module:

ARTIFICIAL INTELIGENCE (AIT-401)

Assignment:

Vacuum cleaner Intelligent Agent Report.

Due date:

28TH FEBRUARY, 2026.

Table of Contents

1. Agent Design Overview

This project implements an AI agent modelled after the classic Vacuum Cleaner World described in Russell & Norvig's Artificial Intelligence: A Modern Approach. The agent operates in a two-room environment (Room A and Room B), perceives the cleanliness state of its current room, decides whether to clean or move, and tracks performance over time.

The implementation uses the following:

- **Python's asyncio framework:** this is used to simulate concurrent operations, including the agent's decision loop,
- **Randomizer:** that periodically dirties rooms randomly.
- **Live dashboard**: rendering the current world state.

1.1. Environment Description

The environment consists of exactly two rooms: **A** and **B**.

Each room can be in one of the two states:

- clean (no dirt present)
- dirty (dirt present, requiring cleaning)

The environment is dynamic: the **randomizer module** periodically sets rooms to dirty or clean based on probabilistic rules.

The dirty is randomly set to a room where the agent is not currently in.

The use of a randomizer simulates real-world conditions where cleaned areas can become dirty again.

1.2. Agent Architecture

The agent is implemented as a Model-Based Reflex Agent. Unlike a simple reflex agent that reacts only to current perceptions, this agent maintains an internal state of the world, tracking each room's state.

Component	Class / File	Responsibility
Environment	environment.py	Manages room states, provides async-safe access via locks
Agent	agent.py	Perceives, decides, and acts; maintains internal belief model
Actions	actions.py	Low-level clean, move, and rest operations (legacy sync version)
Metrics	metrics.py	Tracks rewards, penalties, clean count, goal count
Randomizer	randomizer.py	Stochastically changes environment state at intervals
Dashboard	dashboard.py	Renders real-time world state and metrics to the terminal
Main	main.py	Entry point; launches all async tasks concurrently

2. Rationality of the Agent

To assess rationality, we evaluate the agent against the four criteria of the PEAS framework (Performance, Environment, Actuators, Sensors).

2.1. PEAS Analysis

PEAS Element	Description
Performance Measure	+10 per room cleaned, +50 for achieving both-clean goal, -1 per move, -5 time penalty per cycle
Environment	Two-room world (A and B), dynamic: rooms can become dirty at any time
Actuators	Clean current room, Move to other room, Rest (idle)
Sensors	Perceive current room's cleanliness state (dirty / clean)

2.1.1. Rationality Assessment

Yes, the robot is rational

2.1.2. Evidence of Rational Behaviour

- **Perception-driven action:** The agent always perceives the current room before acting. If dirty, it cleans; if clean, it moves. This is the optimal strategy given the available sensors and a two-room environment.
- **Performance maximization:** Cleaning earns +10 reward and avoids leaving dirt. Moving costs -1 but is necessary to discover and address dirt in the other room. The agent minimizes unnecessary moves by only moving when the current room is already clean.
- **Goal recognition:** The agent checks whether both rooms are clean at the start of each action loop. When the goal state is achieved, it stops the cleaning phase and claims the +50 bonus reward, avoiding wasted effort.

- **Model-based belief:** The agent maintains an internal model of room states. This allows it to reason about the world beyond its immediate sensor reading, which is important given the dynamic environment.

2.1.3. Limitations to Full Rationality

- **Partial observability:** The agent can only perceive the room it is currently in. It has no direct knowledge of the other room's state and must move to discover it, incurring a movement penalty.
- **Dynamic environment:** The randomizer can dirty a room the agent just cleaned or is not currently in. The agent's belief model does not proactively update for changes in rooms it cannot see, meaning the internal model may become stale.
- **No predictive planning:** The agent does not plan ahead it reacts cycle-by-cycle. A more rational agent could use the randomizer's probability distribution to inform when or how often to revisit rooms.
- **Fixed rest duration:** After each cleaning cycle the agent rests for 5 seconds regardless of environment state, incurring a time penalty even if rooms are already dirtying up again.

3. Simulation

This implementation corresponds to the Hard difficulty level: the agent is initialized with a fixed starting state (Room A, both rooms dirty), but the environment changes dynamically at set intervals while the agent moves, via the randomizer module.

3.1. Initial Conditions

Parameter	Initial Value
Agent starting room	Room A

Room A state	dirty
Room B state	dirty
Agent phase	CLEANING
Internal model (belief)	{'A': None, 'B': None} — unknown until perceived

3.2. Dynamic Environment Rules

The randomizer runs concurrently with the agent and applies the following logic every 2 seconds:

- It targets the room the agent is NOT currently in (to avoid interfering with active cleaning).
- If the agent is in CLEANING phase: 60% probability the other room becomes dirty, 40% clean.
- If the agent is in RESTING phase: 30% probability the other room becomes dirty, 70% clean.

This simulates a realistic dynamic environment where previously cleaned areas can deteriorate, and the agent must continuously re-evaluate and act.

3.3. Agent Decision Loop

Each iteration of the agent's act() method follows this logic:

```
perceive current room state
  if dirty:
    clean() → set state to 'clean', update belief, +10 reward
  else:
```

move() → switch current_room, -1 reward

every 10 seconds:

if both_clean(): reward_goal() (+50), break early

else: apply time_penalty (-5), rest 5 seconds, report metrics

3.4. Sample Simulation Trace

The following table shows a representative trace of the agent's behaviour across several time steps in a dynamic environment:

Time (s)	Agent Room	Room A	Room B	Action Taken	Reward
0	A	dirty	dirty	Perceive A: dirty → Clean A	+10
1	A	clean	dirty	Perceive A: clean → Move to B	-1
2	B	clean	dirty	Perceive B: dirty → Clean B	+10
3	B	clean	clean	both_clean() → Goal achieved!	+50
4	B	clean	clean	Resting phase begins (5s)	-5
9	B	dirty*	clean	New cycle: Perceive B: clean → Move A	-1
10	A	dirty	clean	Perceive A: dirty → Clean A	+10

3.5. Metrics & Reward Tracking

The Metrics class accumulates rewards and counters throughout the simulation. The dashboard displays live values and a summary is printed to the console at the end of each cleaning cycle:

```
===== METRICS =====
Cleans: 4
Goals: 2
Total Reward: 113
==================
```

The reward structure incentivizes the agent to clean efficiently, achieve the both-clean goal state, minimize moves, and avoid wasting time aligning individual action rewards with the overall performance measure.

4. Code Structure Summary

The codebase is fully asynchronous, using Python 3's asyncio for concurrent execution of the agent loop, randomizer, and dashboard. All shared state (room cleanliness) is protected by an asyncio.Lock to prevent race conditions.

4.1. Key Design Decisions

- **Async-first**: Using asyncio allows the agent, randomizer, and dashboard to run concurrently without threading complexity. The agent's sleep calls yield control, allowing other coroutines to run.
- **Lock-protected environment**: Every read and write to room state uses an async lock, ensuring consistency across concurrent tasks.

- **Separation of concerns**: Each file has a single, well-defined responsibility. The agent does not directly manipulate environment internals it always goes through the Environment API.
- **Model-based belief**: The agent's internal_model dictionary provides a simple but extensible belief state. It is updated after every clean action and after every perceive call.

4.2. Running the Simulation

The dashboard will clear the terminal and refresh every second, displaying room states, agent position, phase, belief state, and current metrics in real time.

5. Summary

This project successfully implements a Model-Based Reflex Agent solving the Vacuum Cleaner World at the Hard difficulty level. The key achievements are:

- A fully functional, concurrent simulation using Python asyncio.
- A dynamic environment that changes independently of the agent, creating a realistic and challenging scenario.
- A rational agent that perceives, updates its belief model, and acts to maximize a well-defined performance measure.
- A live terminal dashboard for real-time observation of agent behaviour and metrics.

The agent demonstrates core principles of AI rational agency as described in the textbook, while also illustrating the practical challenges of partial observability and environmental non-stationarity in real-world agent design.