Analysis of the Developed Solution

We chose a multi-agent system (MAS) for our greenhouse simulation because the nature of the problem required distributed, modular, and flexible control. In the greenhouse scenario there are many independent entities each needing to monitor conditions, detect anomalies, and possibly act. A MAS lets each entity be an autonomous agent, with its own responsibilities. Instead of one monolithic controller trying to do everything, agents can specialize. Some monitor environmental conditions, others classify plant health, others execute corrective actions. This division of labor leads to parallel execution, scalability, and more realistic simulation behavior. In dynamic, uncertain environments a multi-agent architecture is especially suitable.

When designing the system, we considered a set of key variables that influence how agents should behave and how the simulation would perform:

- **Scale of the greenhouse (number of plants/zones)**:
  As the number of plants increases, workload rises. MAS allows distributing that workload among many agents so performance remains acceptable even at larger scales.
- **Diversity of tasks (detection, classification, action, monitoring)**:
  Different tasks have different computational or timing requirements. Some tasks demand quick reactions to events and others require reasoning or decision-making. By assigning suitable architectures per agent the system remains efficient and effective.
- **Dynamic and unpredictable environment**:
  Anomalies or environmental changes may happen at any time. Agents must respond locally and autonomously but also coordinate when needed. This requires autonomy + communication + coordination.
- **Need for coordination and communication**:
  When multiple agents contribute partial or their actions affect shared resources, coordination is crucial. The MAS framework supports communication and collaboration among agents, enabling coherent global behavior even though agents act individually.
- **Performance/resource constraints**:
  Given finite computing resources, the architecture must balance complexity with efficiency. By using specialized agents and mixing reactive + reasoning agents, we optimize resource use while maintaining functionality.

As scale or complexity increases, the benefits of MAS become more noticeable; but at the same time communication overhead and coordination complexity also grow. The chosen design helps balance those trade-offs.

For visualization, we used a simple but effective graphic design resembling a greenhouse layout. This design allows clear representation of spatial structure and provides visual cues to highlight agent states, plant health or anomalies, and agent activity. This makes it easier to observe, debug, and interpret what happens during simulation. The design refrains from photorealistic detail and instead favors clarity and abstraction.

The final solution brings several advantages. Modularity and specialization make maintenance and extension easier. So its scalability and flexibility mean adding more plants, agents, or behaviors is possible without redesigning the whole system. Hybrid agent architectures let the system respond quickly to events while still enabling intelligent decisions. Decentralization ensures robustness and visualization facilitates understanding and debugging.

Yet, there remain limitations. Multi-agent systems add complexity in design and coordination. Like defining agents, their roles, their communication protocols requires more effort than a simpler system. As the number of agents increases, communication overhead and potential conflicts may degrade performance or lead to unpredictable emergent behaviors. Because the environment is simplified and abstracted, some real-world subtleties are lost. The simulation remains a model, not a full real greenhouse. Also, debugging and validation are harder.

To mitigate those drawbacks, some modifications could help. Adopt a hybrid architecture selectively. Reserve reasoning agents only for tasks that truly need them, while keeping most agents reactive. Implement even more efficient communication to minimize overhead; possibly add a supervisory agent to monitor global state, catch conflicts or undesired emergent behavior and build a test suite of simulation scenarios to validate robustness, test coordination, and detect bugs.

Reflection of the project

At the beginning of the semester I was planning to learn how to use Unity for building simulations and Blender how to create my own Models. As well to understand different ways to simulate a system of “intelligent” actors.

Over these weeks I designed multiple agents with defined roles. Mapping the area, scanning and classifying tomatoes, and harvesting/transporting healthy or infected produce. I built cooperation, communication, task-assignment, collision avoidance, and realistic behaviors into the system. I also saw the simulation handle realistic problems like disease-driven crop waste.

Beyond the technical work, I started to think about system design, modularity, agent cooperation, trade-offs between complexity and functionality, abstraction versus realism. I realized that building a simulation is as much about architecture, assumptions and design decisions, as about code.