Sorbonne University
Master ANDROIDE / AI2D

FOSYMA PROJECT - DEDALE

FoSyMa Project Report - 08/05/2025

Tarık Ege EKEN – 21110611 – Group N°: 13
GitHub Repository: github.com/EgeEken/PROJET_FOSYMA

May 8, 2025

Contents

1 Introduction

In this report I will present the strategies I implemented for this project, their architecture and methods of exploration, exploitation and communication.

Highlights:

- Exploration and resource search and collecting using `SearchBehaviour`
- Multi-agent collision avoidance with `GetOutOfMyWayBehaviour`
- Knowledge sharing with multiple specialized maps to simplify pathfinding and resource collection
- Expert verification system to ensure complete exploration by all agents
- Autonomous agent behavior with minimal centralized control

2 General Architecture

2.1 UML Diagram

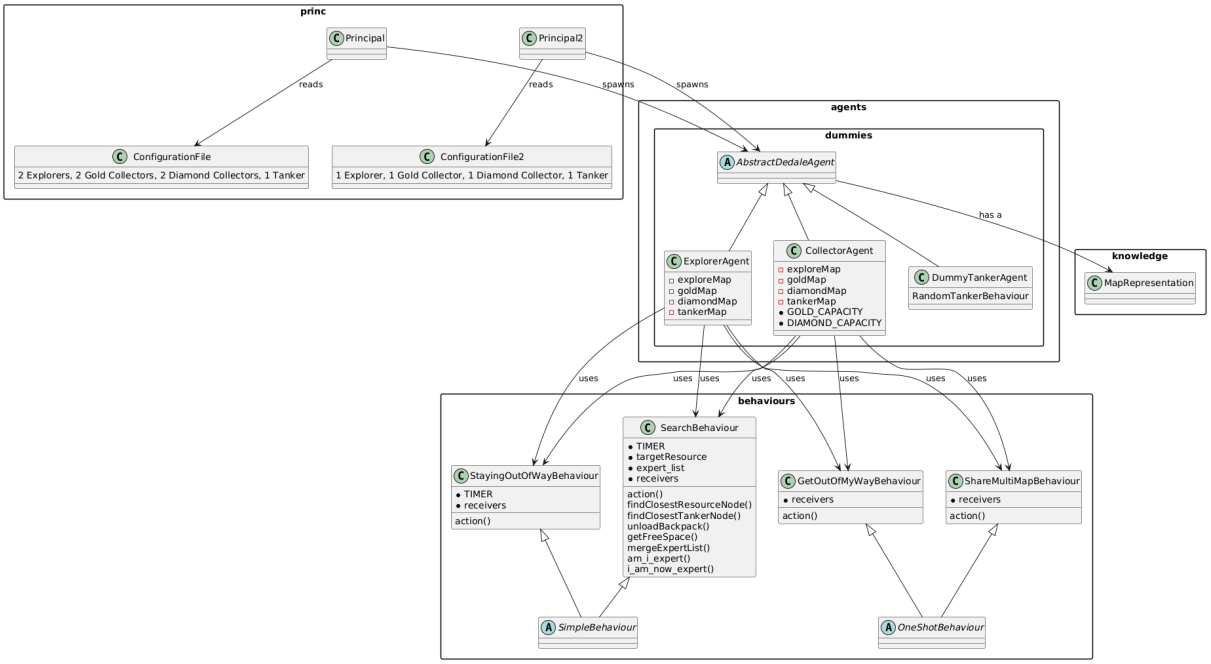

Figure 1: UML Diagram

2.2 Code Structure

The implementation consists of different components:

- **Agents**: Three agent types with specific roles (Explorer, Collector, Tanker)
- **Behaviours**: Four key behaviours controlling agent actions
- **Maps**: Four specialized map representations tracking different environment aspects

- **Communication protocols**: Multiple message types for map sharing and coordination

2.3 Agents

My implementation uses three agent types:

- **ExplorerAgent**: Focuses entirely on map exploration and information sharing
- **CollectorAgent**: Specialized in collecting either gold or diamonds based on initialization parameters and returning them to tankers. But also explores and shares information
- **DummyTankerAgent**: Stationary agent that serves as a storage point for collected resources

Each agent operates autonomously based on its observations and current knowledge state.

2.4 Behaviours

Four main behaviours implement the multi-agent system's functionality:

- **SearchBehaviour**: Primary behaviour for exploration and resource collection, handling path planning, resource pickup, and map updates
- **ShareMultiMapBehaviour**: Communication behaviour for sharing all map representations with all nearby agents
- **GetOutOfMyWayBehaviour**: Collision avoidance by requesting other agents to move when blocking a path
- **StayOutOfWayBehaviour**: Final dormant state for agents that have completed their tasks

2.5 Maps

Each agent maintains four different map representations:

- **exploreMap**: Tracks explored vs unexplored nodes in the environment
- **goldMap**: Records locations where gold has been found and remains (open) or exhausted (closed)
- **diamondMap**: Records locations where diamond has been found and diamonds remain (open) or exhausted (closed)
- **tankerMap**: Marks locations of tanker agents for resource delivery, the tanker's position is not added to any of the other maps, since it can not be walked over, to avoid pathfinding issues

These maps are continuously updated as agents explore and are shared among agents to build a collective knowledge base.

3 Strategy

3.1 Exploration

The behaviour `SearchBehaviour` handles both the exploration and exploitation for both `ExplorerAgent`s and `CollectorAgent`s. Each agent goes to the closest open node in their `exploreMap` using Dijkstra's Algorithm to find the best route.

1. **Observe**: Calls `observe()` to observe adjacent nodes and check the current node's resource status
2. **Receive**: Receives messages sent by other agents in range
3. **Update**: Updates nodes and edges in the 4 `MapRepresentation`s
4. **Send**: Sends the updated maps to other agents in range
5. **Traverse**: Finds the closest open node in the `exploreMap` using Dijkstra's Algorithm and moves to it

Merits: Straightforward, fast exploration
Drawbacks: The order of nodes chosen for exploration is not necessarily the most efficient one in the long run
Complexity: Dijkstra's Algorithm is $O(N + E)$ time, $O(N)$ memory

3.2 Exploitation

The behaviour `SearchBehaviour` handles the resource collection for `CollectorAgent`s. Each collector agent has one type of resource they target, and aims to bring back to the closest tanker agent they know of.

1. **Target Detection**: Checks current node for target resource
2. **Resource Collection**: Attempts to open lock and pick up resources if present
3. **Backpack Management**: Monitors backpack capacity and looks for a
4. **Delivery**: Navigates to tanker and unloads collected resources

Merits: Resource-specialized agents; efficient pathfinding to resources and tankers
Drawbacks: Agents may compete for the same resources without coordination
Complexity: Also uses Dijkstra's Algorithm for pathfinding, $O(N + E)$ time, $O(N)$ memory

3.3 Information Sharing

Agents broadcast all 4 of their maps using `ShareMultiMapBehaviour` every time they take action. This allows agents to rapidly exchange information without requiring extensive communication measures, but comes at the cost of a lot of redundant messages. Each received map is then merged with the agent's current map before making a new decision.

Merits: Simple to implement, resilient to message loss.
Drawbacks: High bandwidth; no incremental updates. Does not wait for an ack to send

maps, which causes a lot of message loss
Cost: $O(4 * (|V| + |E|))$ bytes per broadcast. (4 maps)

3.4 Collision / Blocking avoidance

When agents encounter another agent on their path, they send a GET-OUT-OF-MY-WAY (GOOMW) message, using `GetOutOfMyWayBehaviour`, which signals the agent they detected to move out of the way. On the event that an agent that has been requested to move is unable to do so due to having no alternative edges to move through, they send back a GOOMW message to the agent they received it from. They can also send it to another agent that is blocking their path while trying to get away from the first agent's path, creating a chain reaction that prevents blocking due to too many agents piled up.

Merits: Cheap messages, no need to broadcast to all nearby agents like `ShareMultiMapBehaviour`, only one message sent at a time
Drawbacks: Latency between updates; potential for redundant effort.
Cost: 6 bytes per broadcast. (3 characters per message, sent to only one agent at a time)

3.5 Expert Verification

Explorer and Collector agents that have finished their tasks of exploration, and have an `exploreMap` that has no open nodes left are deemed "experts", and they randomly move around the map sharing their `expert_list` with other agents, which they merge to make sure everyone else is also an expert, when an agent's `expert_list` is completed, meaning that it knows every other agent is an expert, it is deemed a super expert, and it stops moving around, going in a sort of idle state controlled by `StayOutOfWayBehaviour`.

Merits: Verifies that all other maps know the whole map, ensuring it is safe to stop exploring and sharing information
Drawbacks: Random walk following the end of exploration to find non-expert agents relies on luck.

3.6 Final Dormant State (StayOutOfWayBehaviour)

As exploration and exploitation are completed, and every agent has been made sure to be experts, there is still a chance that there are experts out there that need to finish emptying their backpacks, in which case, a fully inactive agent might block their path toward a tanker. So super expert agents are instead placed in a dormant state using `StayOutOfWayBehaviour`, which only listens for GOOMW messages, and moves out of the way if requested, and do nothing else.

Merits: Prevents blocking by inactive agents, with no cost of movement or communication
Drawbacks: Depending on the goal of "efficiency", it might be considered more efficient to simply move to a distant corner of the topology under the assumption that the agent won't be in anyone's way, rather than wait to be moved, but this isn't necessarily the case

4 Synthesis, Critique and Possible Improvements

My implementation manages to successfully explore any given topology I tested it out on, and gather and store resources effectively. But unfortunately there are some issues that I have been unable to fix in time:

- Due to some obscure concurrency problem, during collision avoidance, agents make multiple moves during the same time agents in the rest of the map make one move, I was unable to fix this bug after hours of debugging, and even a complete overhaul of the collision avoidance system
- The strategy is not particularly well prepared to confront a golem/adversarial mechanism
- There is a lot of message loss, since the agents do not verify if another agent is indeed in range with a smaller message before sending all 4 maps which results in very expensive messages being sent every turn by every agent, and most of them getting lost
- There isn't a centralized decision-making process, or explicit area assignment, or role shifts between agents, each agents acts autonomously, making their decisions based on their situation and observations, rather than taking orders. This has its pros and cons however, it also makes them require less strategic communication in principle
- Due to the partly luck based nature of the collision avoidance and expert verification systems, there is a chance that verification takes a long time stuck in semi-loops going back and forth between two agents before one of them manages to break out

Improving the communication system and perhaps a more centralized strategy could make my implementation more effective.