NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

REPORT FOR AI6125: MULTI-AGENT SYSTEM

GROUP PROJECT

Build Intelligent Agents for Tileworld Environment

ZENG ZHENG

G211941J

zzeng006@e.ntu.edu.sg

School of Computer Science and Engineering

April 6, 2022

1 Introduction of Tileworld

The Tileworld, initially proposed by Pollack and Ringuette[2], is a widely adopted testbed for multi-agent systems. It is a checkerboard-like grid (Figure 1) with agents (blue squares), tiles (green squares), obstacles (black squares), holes (pink squares), and a fuel station (yellow circle).

To be more specific, an agentis a unit square that has limited visibility (blue dashed squares), can move up, down, left, or right, one unit at a time, and consumes one fuel per move. When an agent runs out of fuel, it dies. Thus, to get a longer life, agents are suggested to refuel at the fuel station, which is also a grid cell. Apart from moving in four directions, an agent can also pick up tiles, and put tiles in the holes to get rewards. The tile also is a unit square that can be carried by an agent, who can carry up to three tiles. Obstacles occupy a non-movable grid cell, and agents cannot occupy the same cell as an obstacle. Therefore, it is necessary to re-route and avoid obstacles when moving towards an objective. Besides, there are holes, which are grid cells that can be filled in by tiles (one tile for one hole).

The objective of this project is to get as many rewards as possible by filling in the holes with tiles in a pre-defined simulation step. And in each step, the agent goes through a "perceive-communicate-plan-act" cycle.

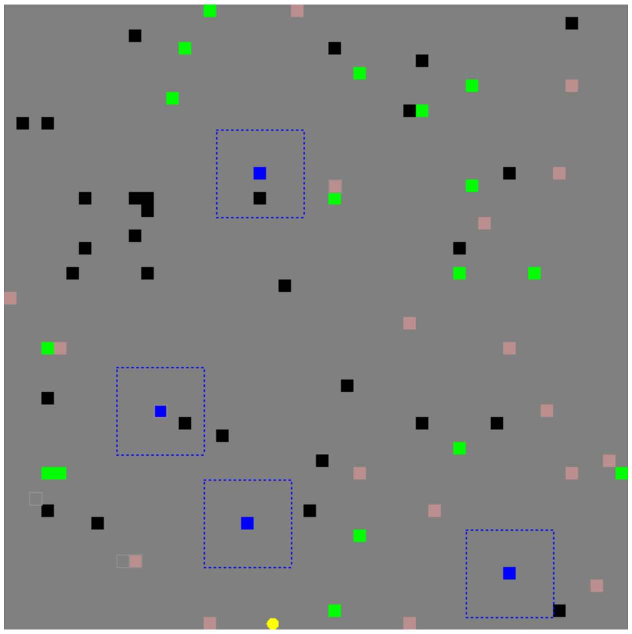
Figure 1: A snapshot of Tileworld game.

2 Team agents Analysis

Our team designed four agent architectures. The first is a MAS based on greedy & inspector agents, where three agents are responsible for collecting tiles and filling holes, while an inspector agent explores the map. And I will describe it in detail in Section 3.

The second is a region-based MAS, where agents work within their assigned areas and help others if they have extra resources.

Besides, there is an agent design with extra communication and utility functions. This agent shares its goal to avoid goal conflicts with others. Also, the agent will choose the best target among the random location, the tile, and the hole by using utility functions.

The last one is a feedback-based MAS, where all agents are benevolent. They look for any requirements in the message and try to give a response to help others. There is also an auction rule, and if agents have a conflicting goal, only the closest one to the destination will retain the goal.

3 Agent Design Description

In my multi-agent system, there are two types of agents who cooperate to find the fuel station, but have different responsibilities once they find it. The **Greedy agent** is responsible for collecting tiles and filling holes in the team to gain points, while the **Inspector agent** is responsible for exploring the map and broadcasting the information gained to other agents. In addition, both types of agents perform better when working together than when acting alone.

As for the design architecture of these two agents, they are basically the same except for the planning layer, so I will first introduce the whole architecture, and then introduce the planning layer of the inspector agent separately to show their difference.

3.1 Agent architecture

The overall architecture of Greedy agent and Inspect agents as Figure 2, which is a **hybrid** architecture based on the classic TOURINGMAHINES architecture[1], which is composed of three subsystems, which are perception subsystem, action subsystem, and a control framework embedding three control layers, which will be implemented as the *sense()*, *think()*, and *act()* functions in our projects. And in every step, an agent goes through a **Sense-Communicate-Plan-Act** cycle.

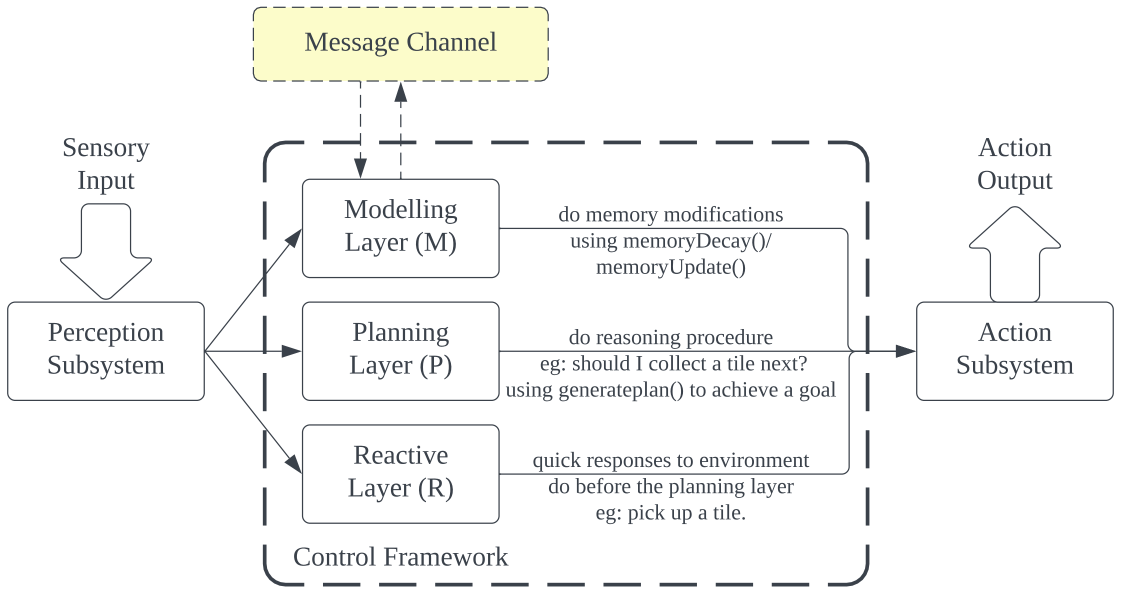
Message Channel
Sensory Input
Perception Subsystem
Modelling Layer (M)
Planning Layer (P)
Reactive Layer (R)
do memory modifications using memoryDecay()/ memoryUpdate()
do reasoning procedure eg: should I collect a tile next? using generateplan() to achieve a goal
quick responses to environment do before the planning layer eg: pick up a tile.
Control Framework
Action Output
Action Subsystem

Figure 2: Agent architecture of Greedy agent and Inspect agent.

The control framework is a typical horizontal layering, consisting of a modeling layer, a planning layer, and a reactive layer. Each layer interfaces directly with the sensory input and action output. In addition, there is an additional **communicate** module, which is put on top of the control framework and will be described in the modeling layer.

3.1.1 Modelling layer

In the modelling layer, agents will model the world, using the information provided by their sensor and by other agents via communication. Then memory modifications will be done by using functions in the memory module to update internal world model. I extended the *TWAgentWorkingMemory* class to enable agents to better model the world. In general, depending on the source of information, we have two ways to process information for memory modification.

1. For information percepted by agents, we use *memoryUpdate()* and *memoryDecay()* functions.
2. For information obtained by communication, we implemented the **memoryMerge()** function to perform memory merging. In this way, changes in objects outside the agent's sensor range will also be detected to keep the agent's memory up to date.

3.1.2 Planning layer

In this layer, agents deliberate about **what intentions** to achieve next such as collecting a tile or filling a hole, based on the current environment state. Normally, the intention is a long-term goal or a set of goals instead of a single act such as picking up a tile. Then agents will use *generatePlan()* function to generate a sequence of actions for action subsystem to execute in order to achieve this goal. The logic reasoning procedure is as Figure 3.

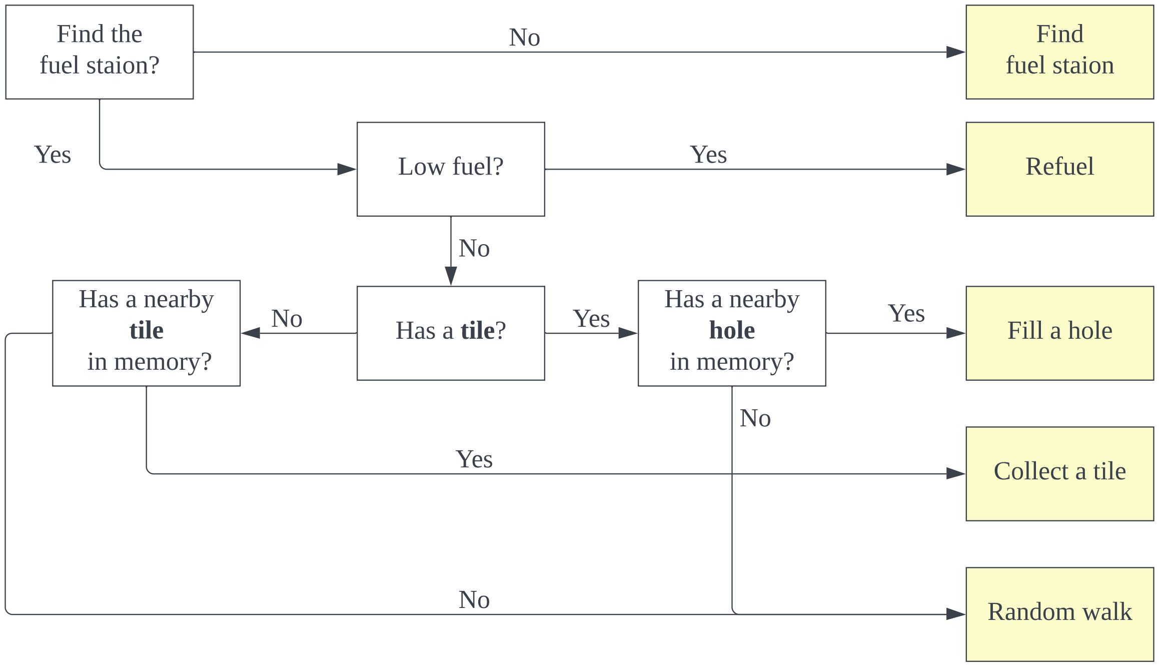

Figure 3: Logic reasoning procedure of Greedy agent.

The agent has five intentions:

1. to **find fuel station** when it is not found by any agents.
2. to **refuel** if the fuel level of agent is low.
3. to **fill a hole** if agent has (is carrying) a tile and looks for a hole in memory which is nearby.
4. to **Collect a tile** if agent has no tiles but looks for a tile in memory which is nearby.
5. to have a **random walk** is triggered by two conditions:
   (a) when the agent has a tile but doesn't have a hole in memory.
   (b) when the agent has no tiles and there is no tile observed in memory.

Besides, about the intention of find the fuel station, here is my searching strategy. As Figure 4, We divide the tile world into four small worlds of the same size and assign each one of them to each agent. During the search, they use their sensor ranges to minimize their steps. Once one agent finds the gas station, he broadcasts this information. So other agents can stop searching and begin to work.

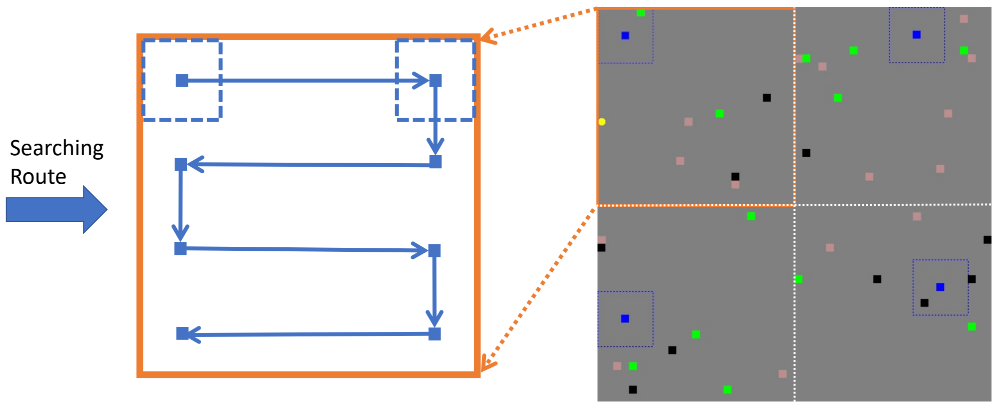

Figure 4: The searching route of agents.

3.1.3 Reactive layer

In addition to modelling and planning, there is a reaction layer, which usually has some priority over the deliberation layer, and in my design it is no exception. In this layer, the agent will make some quick reactions (simple actions comply with the **if-then rules**) to the current environment, and the if-then rules are as follows:

1. If there is a fuel station where the agent is standing, then refuel.
2. If there is a tile where the agent is standing, then pick up a tile.
3. If the agent has (is carrying) at least one tile and there is a hole where the agent is standing, then fill a hole.

3.1.4 Communication module

In this module, I extended the *Message* class, and the message will be any objects apart from *String* variables. Agents can send and receive messages about the fuel station location, send messages of their locations and perceived objects, and receive messages of other agents' locations and objects to refresh their memories. Agents send messages in the *communicate* stage, and receive messages in the *think* stage.

3.2 Inspector agent

Motivation I designed the Inspector agent to address the problem of goal conflict, which is a bothersome phenomenon when I conduct experiments with the Greedy agent. As the name implies, greedy agents always target the closest tiles and holes to them. But the problem is that when they share their memories and have a poor exploration strategy, they always gather together to achieve this goal when there are not enough available resources (tiles and holes in their memories). This is not efficient for agents to explore the world, because there are many resources far away from them, but the agents are not aware of them.

What Inspector do This agent is responsible for searching the map on a fixed route (see the blue arrow in Figure 5), recording the tiles and holes perceived by that agent, and broadcasting this information to other Greedy agents, so they have more conflict-free targets in the absence of resources around them. And this could avoid agents from grouping together and improve the whole performance.

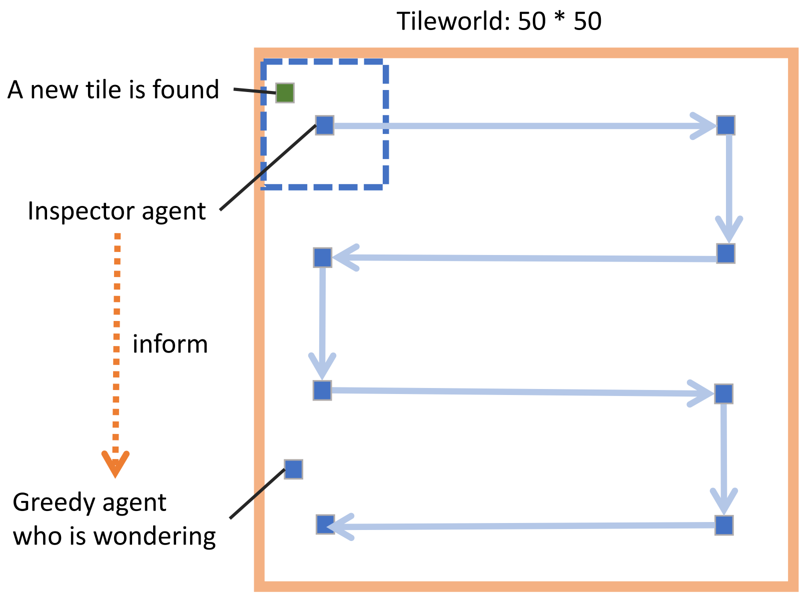

Figure 5: The role and the fixed route of inspector (orange and green arrows represnt how Greedy agents cooperate with Insepctor Agent; blue arrows represent the route of Inspector agent.

The reasoning process of the inspector agent (as shown in Figure 6) is less complicated than the Greedy agent because the inspector has only two intentions, i.e., to find the gas station and to explore the map, depending on whether the team finds the gas station or not.

3.3 Implementation

Here, I briefly mention the code of the project I implemented. The modified codes include:

1. MyAgent extends TWAgent;

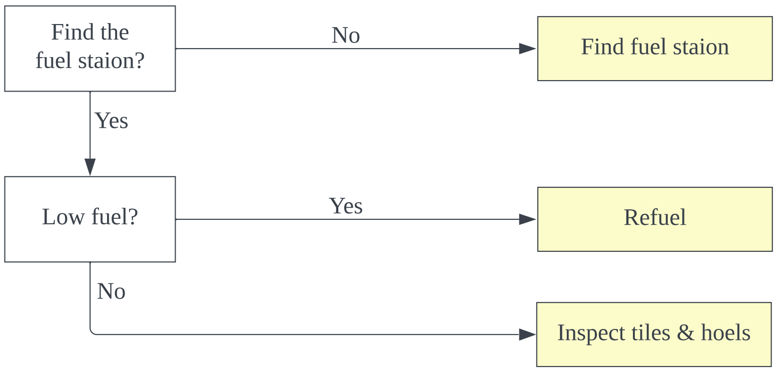

Figure 6: Logic reasoning procedure of Inspect agent.

2. MyInspectorAgent extends TWAgent;
3. MyMemory extends TWAgentWorkingMemory;
4. MyMessage extends Message;
5. DefaultTWPlanner modification using AstarPathGenerator.

4 My Designs and Experiment Results

I have tried several agent designs, so firstly, I will introduce their main ideas and their differences, then I will show the performances of these agents and analysis the results.

Simple reactive + limited communication Agents only share the location of the fuel station and have no intention but some quick reactions to the environment in the planning layer.

Simple hybrid + limited communication Agents also share fuel station information but followed the hybrid architecture as Figure 2 for the control framework. And there are four intentions, i.e., refuel, collect tiles, fill holes, and random walk.

Greedy hybrid + limited communication This agent architecture is almost identical to the previous one. However, the difference is that this design has an additional intention, i.e., **find fuel station**.

Greedy hybrid + communication This architecture is based on the previous one, but at the communication level, apart from the fuel station information, they also the tiles and holes they have recently discovered. When resources are unevenly distributed on the map (i.e., the same resources are clustered in the same place), the information sharing will help improve overall efficiency.

Friendly hybrid + communication In this architecture, agents try to be friendly and find resources that will not be targeted by others to avoid goal conflict situations. If they always go for the nearest tile or hole, which is likely to be someone else's target as well, so the agent will waste a lot of time inevitably. And the logic of a Friendly agent is as follows:

1. Get ten nearest tiles/holes from this agent, and for every tile/hole:

2. (a) Get the positons of other agents.
   (b) Comparing distances from the resource to all agents, including the agent itself.
   (c) If there is at least one other agent who is closer to the resource than this agent, it means that goal conflict is probable. So This agent will behave friendly and give up this resource to others.
3. If there is at least one resource left, the agent set the nearest tile/hole as a goal.

Greedy & Inspector hybrid combination + communication This architecture has been elaborated before, see Section 3.2.

4.1 Experiment results analysis

There are 2 different environment configurations for experiments. For configuration one, the siez of environment is 50×50, the average object creation rate fits a normal sidtribution ($\mu = 0.2, \delta = 0.05$), and the lifetime of objects are 10. As for the second configuration, the siez of environment is 80×80, the average object creation rate fits a normal sidtribution ($\mu = 2, \delta = 0.5$), and the lifetime of objects are 30.

We will run 10 experiments for each configuration. For each experiment, we will use a different random seed, the total time steps is 5000, the amount of fuel at beginning is 500, and we will have 4 agents in the environment. We calculate the average of the results of the 10 experiments as our final performance for this agent design. And The performances of six agent designs in 2 different environment configurations is as Table 1.

Table 1: The average of ten results of four agents in two different environment configurations

Exp.	Architecture Name	Configuration One	Configuration Two
1	Simple reactive + limited communication	3.0	8.3
2	Simple hybrid + limited communication	121.6	109.9
3	Greedy hybrid + limited communication	208.1	336.5
4	Greedy hybrid + communication	210.8	**354.4**
5	Friendly hybrid + communication	190.4	330.0
6	Greedy & Inspector hybrid(3, 1) + communication	**271.0**	305.0

From Table 1, we find that agents of hybrid architectures perform much better than simple reactive ones (Exp 1). Also, comparing the figures for Exp 2 and Exp 3, we can that finding the fuel station is vitally important for agents. Similarly, from the figures of Exp 3 and Exp 4, we find that memory sharing (the information of tiles and holes found by other agents) is slightly helpful for greedy agents since the average score increases from 336.5 to 354.4 for configuration two.

However, from the figures for Exp 4 and Exp 5, we find the friendly agents are not very helpful, we think it is because the algorithm I designed is not good enough. Finally, from the data of Exp 4 and Exp 6, we find that the Inspector agent improves the performance of configuration 1, but reduces the performance of configuration 2. I believe this is because the latter has a much higher object creation rate and a much shorter lifetime than the former. Therefore, greedy agents always have enough resources around them, and inspectors are less useful, but reduce the number of workers.

In conclusion, architecture 4 agents perform best in the first environment (271.0), while the architecture 6 agents perform best in configuration two (354.4).

References

[1] Innes A Ferguson. *TouringMachines: An architecture for dynamic, rational, mobile agents.* Tech. rep. University of Cambridge, Computer Laboratory, 1992.

[2] Martha E Pollack and Marc Ringuette. "Introducing the Tileworld: Experimentally evaluating agent architectures". In: *AAAI*. Vol. 90. 1990, p183–189.