Group Project

AI6125: MULTI-AGENT SYSTEM

G2202588D JIANG HAOFENG
G2201257D TAN ZHAOZHANG
G2202727A ZHAN SHUO
G2202769J TONG TIAN
G2202544B JU XILAI

Build Intelligent Agents for Tileworld Environment

April 15, 2023

1 Introduction

1.1 Background

Tileworld is a chessboard-like grid shaped environment, we need to design agent-based model to implement in this environment. There are objects includes agents, tiles, obstacles and holes, see Figure 1.

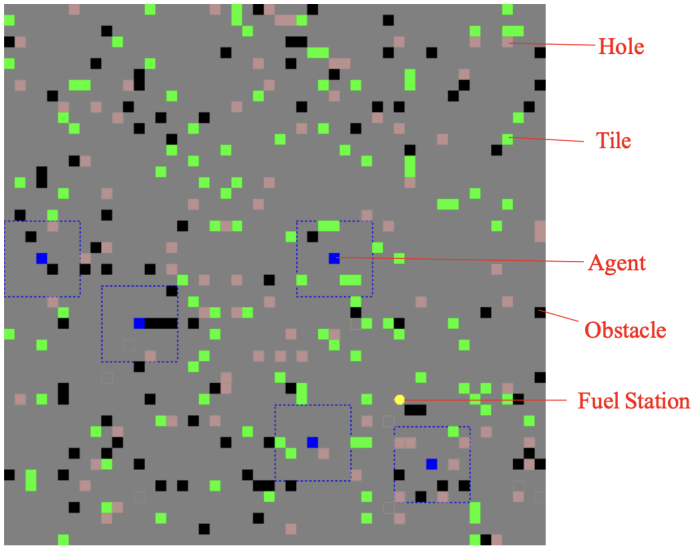


Figure 1: The Tileworld example

To be clear, specific explanations are shown below:

- Agent: An agent is able to move up, down, left and right, one cell at a time, what we need to do is design the agent to get higher scores.
- Tile: A tile is the object that agent carries, can be considered as cargo while the agent can be considered as intelligent trucks to carry the cargo. Occupy a unit square.
- Obstacle: An obstacle is a ”wall” that agent cannot cross or move or stay. Occupy a unit square.
- Holes: A hole is a unit grid cell for agent to put the tile down. Once fill a hole with a tile, then the agent can obtain a score.

1.2 Agent Specifications

1.2.1 Limited View of Agent

The agent can only perceive limited views, or more concrete, the neighbouring cells around him. In the project, we set the limited view size as 3 on four directions, which means the agent can see 7×7 cells with himself centered, see Figure 2.

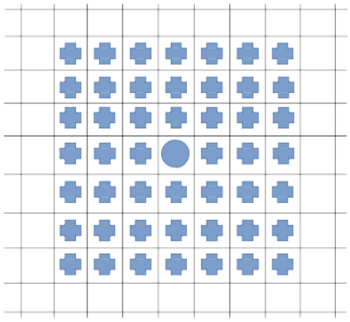


Figure 2: The Visibility of an Agent

1.2.2 Limited Actions

Each agent has limited actions:

- Wait
- Move (Up, Down, Left, Right)
- Pick Up (tiles)
- Drop (tiles to holes)
- Refuel

1.2.3 Limited Capacities

Each agent is able to carry up to 3 tiles at a time, once they find the hole, they can drop the tile (or not, not mandatory).

1.2.4 Limited Fuel

Just like trucks, agents need fuel to move. There is a fuel station in the Tileworld, agents need to refuel themselves before they run out the fuel. Each action consumes one fuel from the agent

1.2.5 Other Functions

Agents can communicate with others anytime, which means they can share their view and what they will do with each other.

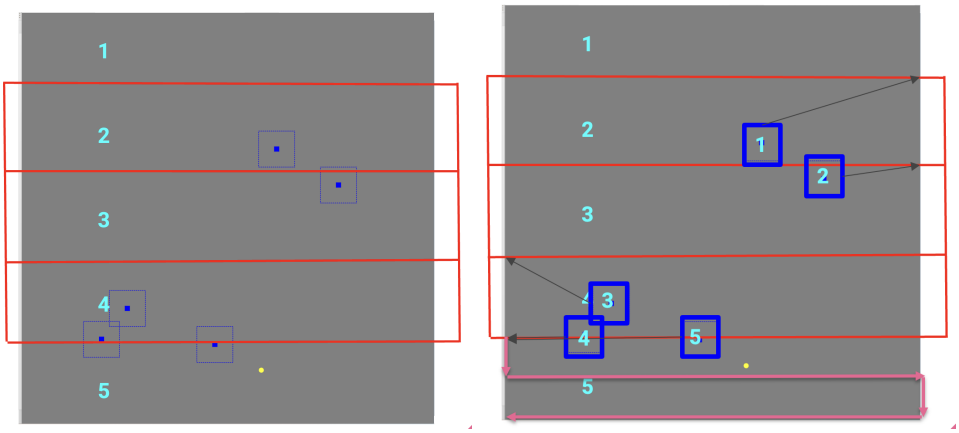
2 Agents Design

To be clear, we adopt rule-based system to design our agents, we separate our systems to three parts: Fuel-Station Finding & Voyage Agent, Communication & Global Memory, and specific Work Implementation.

2.1 Fuel-Station Finding & Voyage Agent

2.1.1 Fuel-Station Finding

Finding the fuel station at very beginning is vital. If agents failed to find fuel station, the overall score will be extremely low because no agents will survive to the end. That is why we put fuel station finding process as our first step.



(a) Separation of Tileworld (b) Arrangements

Figure 3: First step of Fuel-station Finding

We could separate this task into basically three tasks. The first one is to separate the map into 5 exclusive areas efficiently as we have 5 agents, Figure 3a. We guarantee the width of each area is the multiple of 7 except the last area, so that most of agents won't scan overlapped area to improve the efficiency. And once the agent finish its own responsible area, it could start to work normally.

Then the second task is to match each agent with one point of each area. How to minimize the distance between agents' original position and starting points of scanning? In fact, this is one NP-Hard problem while the number of agents is limited and fixed, which is five. So we could solve the problem by fully arranging all the combinations, calculating and comparing the total distance between all agents and corresponding best starting point of each area.

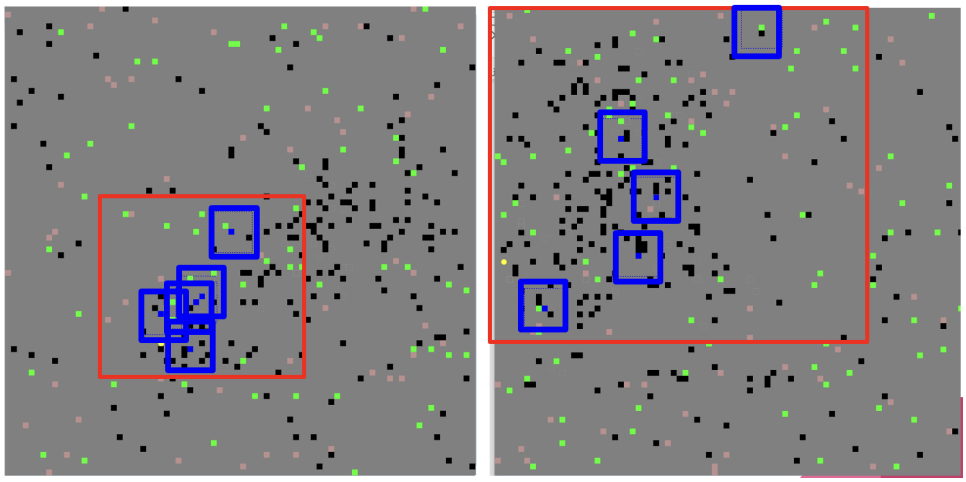
And the last task is to scan the area with U shaped path. See Figure 3b pink arrow lines. Once one agent finds the fuel station. It will broadcast to other agents about the location of fuel station. And all agents stop scanning and start to work normally.

2.1.2 Voyage Agent

Through checking the performance of our normal agents working together on GUI atmosphere, we found a phenomenon frequently happened. Because of our refuel algorithm, the agents will arrive at fuel station in close period which means they will perceive similar or even the same atmosphere after refueling so that they would make similar movements based on same thinking logic. This is the example(left figure). So in this case, they will go through overlapped area which reduces the efficiency. See Figure 4a.

That is why voyage agents come out. They are different with normal agents. After they refuel, they won't directly work immediately.This is the example(right) Instead, they would go to a random place which means different agents would probably go to different direction to explore different area. See Figure 4b.

Besides, we found that, as the step number increases, there are more and more obstacles around the fuel station. So it would be efficient to make some agents explore further area instead of immediately work normally after refueling. This is the intuition of making voyage agent, which is, as steps number goes up, the starting working point after refueling is farther.



(a) Overlap Problem without Voyage Agent (b) Voyage Agent

Figure 4: Voyage Agent Comparison

2.2 Communication & Global Memory

2.2.1 Communication

First we talk about communication. For each step, each agent has a 7×7 view and a target. Since we do not care about complexity in this project, we make all the message public to every agent. That is, broadcast every message. Each agent sends messages in the following four ways: *tile x y*; *hole x y*; *blank x y*; and *target x y*. *x&y* means the location of the object. The tile, hole and blank are related to the global memory, and the target is about dealing with target conflict.

2.2.2 Global Memory

We use a 2D array, or a $X\_size \times Y\_size$ table of strings to store the tiles and holes which sensed by other agents. The reason we use strings to store memory is that the message itself is a string. It is more convenient for us to implement. If it is a tile or hole message, we fill the entry; if there is a "blank" message, make the corresponding entry "NULL".

2.2.3 Target Conflict Problem

There is a problem that still exists: target conflict reduction. Remember the messages include the targets from each agent at this current step. This yields our contract, which is "First-come-first-serve", If one agent has a target registered, the nearby region cannot be targeted by other agents. Take Figure 5 as an example. Once an agent targets at this tile, and registered first, it broadcasts the location of this tile, therefore, another agent will know that this tile has been targeted, and it should not target any object whose distance is less than or equal to 3(units, best performance under 50×50 and 70×70 size). Thus, the agent will choose another target instead. This method saves a lot of steps for agents, especially accompanied with our voyage agent design.

2.3 Work Implementation

2.3.1 Fuel Estimation

During the agent-working period, we need to compare the Manhattan distance between the agent and fuel station with the Fuel Level, in order to find fuel station before agents ran out of fuel.

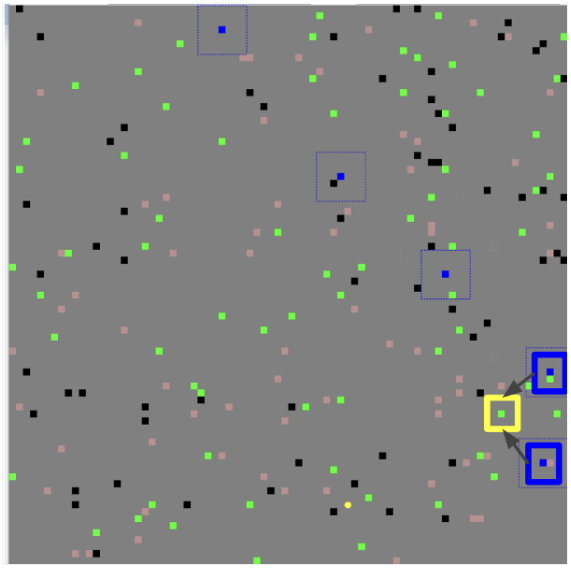


Figure 5: Target Conflict Problem

When the fuel value > distance, the agent continues the tile-hole work. But when the fuel level equals to this distance, the destination of agents changes to the fuel station. However ,we know that there are some obstacles in this map and the obstacles might cause the agent to take extra steps, and the extra steps may cause the agent ran out of fuel. To avoid such condition, we set a buffer value B.

$$F \geq D + B$$

The F is the fuel value and the D is the Manhattan distance between the agent and fuel station. And we find an interesting thing that the amount of obstacles will increase as the system operates, which means the agent need more extra steps when their have run 3000 or 4000 steps compare to at first. so It is reasonable to set buffer value B to be proportional to steps.

And after some experiments, we find when the buffer value equals to steps/100. the system performs very well. But it might not the best value, and it can be developed in the future.

And when the agent goes to the fuel station, it can also do the tile-hole work and does not need to take extra steps. In Figure 6, we can assume a rectangle between the agent and the fuel station, and the agent can get to the tile or hole in this rectangle without taking extra steps, it can also get to the fuel station before running out of fuel.

2.3.2 Agent's Target Priority

During the working process, the agents' priority is determined by the number of tiles carried.

When the agent is working, we can separate it in 3 conditions by the number of tiles it carried, like Table 1.

carried tiles	0	1 or 2	3
target of agents	closest tile	closest tile or hole	closest hole

Table 1: Agent's Target Priority

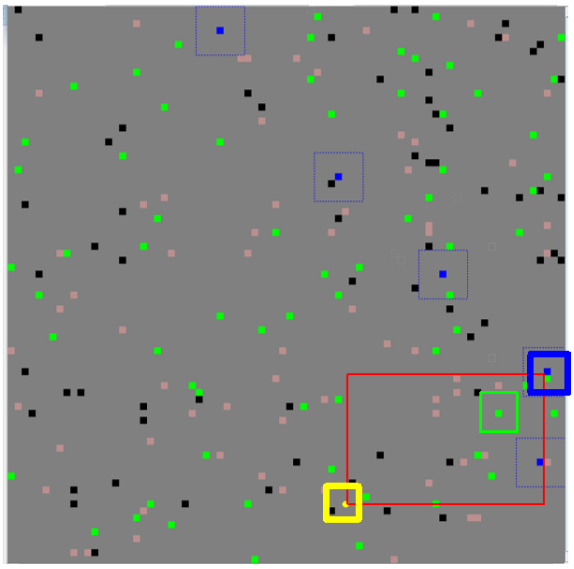


Figure 6: Fuel Estimation

2.3.3 Memory Loss

When the agent's memory contains no tiles or holes, we call it "lose its memory".At that time, it lose the target it want to go, so we have another method to fix this problem.

And it can also be divide into two conditions.

It loses its own memory and it lose the shared memory(it need to be clarified, and it is a very rare condition which means that there is no tile or hole in the 5 agents' memories).

When it loses own memory, it will use the shared memory and find the closest tile or hole in it(obey the rules we have showed before).

But if it loses shared memory, then we need to justify the condition: if its fuel level less than 2 times of the Manhattan distance between the agent and fuel station, the agent will go to the fuel station. if not, it will random go some where(because this condition last very short time, and it won't affect the final score a lot.)

3 Experiments

In order to find the best number of voyage agents in each environment, we experiment on below 3 scenarios:

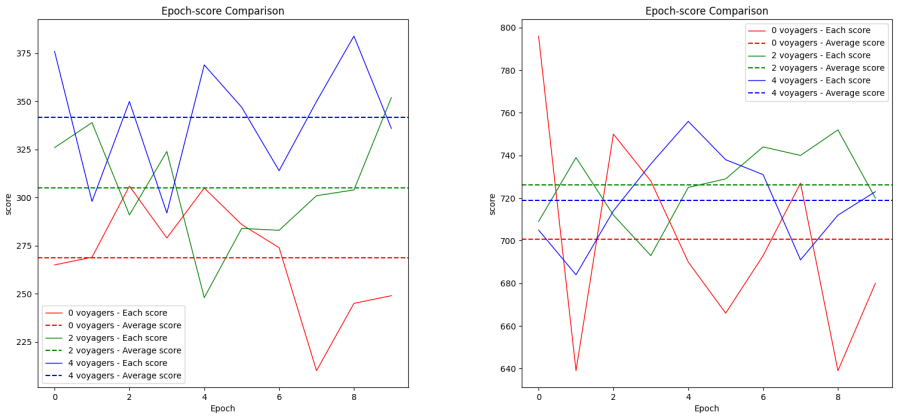
- 5 normal agents, 0 voyage agents
- 3 normal agents, 2 voyage agents
- 1 normal agent, 4 voyage agents

3.1 50X50 cells

We test our model on the environment size 50x50 cells in 10 epochs, the average object creation rate: Normal Distribution ($\mu = 0.2, \sigma = 0.05$), Lifetime: 100. According to Table 2 and Figure 7a, the optimal average score, which is 341.6, occurs when there are 4 voyage agents with 1 normal agent. The large ratio of voyagers probably results from the small scale (50x50), and having more voyagers are beneficial, in terms of sacrificing some steps for the privilege of dispersed agent locations.

Number of voyage agents	Worst single score	Best single score	Average score
0	210	306	268.8
2	248	352	305.2
4	292	384	**341.6**

Table 2: Experiment results on 50x50 cells



(a) 50x50 cells

(b) 80x80 cells

Figure 7: Comparison graph

3.2 80X80 cells

We test our model on the environment size 80x80 cells in 10 epochs, the average object creation rate: Normal Distribution ($\mu = 2, \sigma = 0.5$), Lifetime: 30. According to Table 3 and Figure 7b, the optimal average score, which is 726.3, occurs when there are 2 voyage agents with 3 normal agents. The map is so large (80x80) that the more steps cost by voyagers are nontrivial and should be considered. If voyagers are too few, the locations between each agent are likely to be close, resulting in low scores (639 in our case). Therefore, we finally have similar numbers of voyagers and normal agents.

Number of voyage agents	Worst single score	Best single score	Average score
0	639	796	700.8
2	693	752	**726.3**
4	684	756	719.0

Table 3: Experiment results on 80x80 cells

4 Conclusion & Future Vision

4.1 Conclusion

In our project, we mainly use the rule-based system to achieve high scores. In the 50x50 experiment, we get a relatively low scores **341.6** because of the small maps(more obstacles in the same region). But in the 80x80 experiment, we get average score **726.3**, which has less obstacles and lifetime during the whole process. However, during the presentation, we get the map size 100x100, with the configurations μ_{tile} and σ_{tile} much smaller than the 80x80 environment. The large map and the distribution lead to few and scattered objects, especially tiles, making it hard for agents to search for the targets.

With reference to 80x80 experiment, having 2 voyagers is the optimal ratio. Therefore, we also set 2 voyage agents and 3 normal agents during the presentation, finally getting average score **105.5**. See Figure 8.

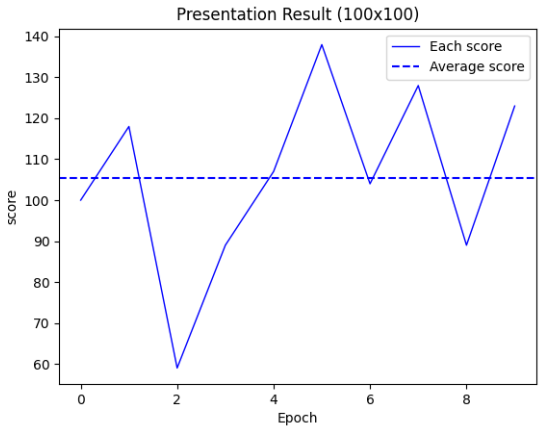


Figure 8: Presentation Result

4.2 Future Vision

4.2.1 Objects Global Memory

Now we have some directions for improvements about our method. The global memory only records the information about the sensors on agents, which means that the tiles and holes in global memory will not disappear unless the agents travel around the neighbouring areas. However, the lasting time of tiles and holes can be recorded in the global memory so we can get the real-time information about tiles and holes.

4.2.2 Partition during implement process

There are still some problems when the agents run out of fuel. All of the agents carry the same amount of fuel, so they will start to find the fuel station nearly at the same time, leading to congestion in the same area.

We can just divide the region into five parts during working process, which will avoid congestion, and it will also change the communication. Moreover, mutual exclusion in perceptual region when sensing might be better in such conditions.

4.2.3 Rule-based System VS RL System

Rule-based System	Reinforcement Learning System	
Easy to be implemented	Efficient	High-Dimension during dynamic situations

Table 4: Rule-based System VS RL System

At first we consider to implement with two methods: Rule-based method and Reinforcement Learning method. Naturally we pay much attention to the reinforcement learning method, which will be more efficient if designed wisely.If the obstacles are fixed, we can use some classical reinforcement learning methods like Q-learning. But the dynamic environment makes it hard to be implemented. So we finally choose the Rule-based method.