# GROUP PROJECT

## AI6125: Multi-agent System

*Build Intelligent Agents for Tileworld Environment*

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**
**NANYANG TECHNOLOGICAL UNIVERSITY**

## 1 OBJECTIVES

The purpose of this project is to:

- Make students familiar with a popular toolkit for the modelling and simulation of agent-based systems.
- Provide students the opportunity to experiment with different agent designs for a popular agent-based testbed.

## 2 SPECIFICATION

The aim of this exercise is to use the MASON agent toolkit (Java-based tool) to implement an agent which can inhabit and perform in the provided Tileworld system. You are free to implement any type of agent you desire using any algorithms you would like. This will be conducted in teams of 3-4 students, with each team member designing and implementing one agent. Your agent team (3-4 agents) will then perform together in the same Tileworld.

### 2.1 MASON

MASON is a widely used, open-source, cross-platform, agent-based modelling and simulation toolkit that was originally developed at George Mason University. It is written in Java and has extensive online tutorials and examples. Before the implementation, it is recommended that you spend some time going through the manual of MASON (Section 1.1 and Section 2.1-2.12) yourself. The manual and other related materials can be obtained from the following link: https://cs.gmu.edu/~eclab/projects/mason/. By reading the recommended sections of the manual, you should have a basic understanding of the architecture of MASON and its working flow.

### 2.2 Tileworld

In this project, you will use Tileworld as a canonical reference for an agent-based model to implement. Tileworld is a chessboard-like grid on which there are agents, tiles, obstacles, holes, and a fuel station. See Figure 1 for an example.

- An **agent** is able to move up, down, left or right, one cell at a time.
- A **tile** is a unit square which can be carried by the agent.
- An **obstacle** occupies a grid cell and is immovable, agents cannot occupy the same cell as an obstacle.
- A **hole** is a grid cell which can be filled in by a tile.
- The **fuel station** is a grid cell where agents can refuel.
- The **reward** is obtained by: when the tile is moved on top of the hole cell, the tile and hole cell disappear, which leaves a blank cell. This means a hole is filled and the agent gets a reward.

In this project, you have to develop an agent for the Tileworld domain. We have uploaded two papers about the Tileworld domain to NTULearn, i.e. “A history of the Tileworld agent testbed” and “Introducing the Tileworld: Experimentally evaluating agent architectures”. You may study them to gain a better understanding about the domain.

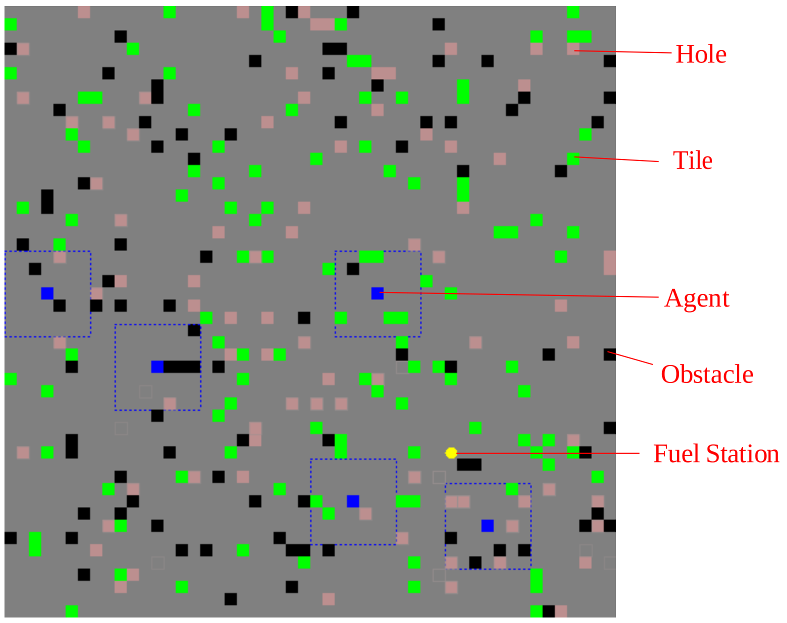

Figure 1: The Tileworld example

### 2.3 Implementing the Agent

The source file of the Tileworld implementation can be downloaded from NTULearn. To compile and run the Java program, you need to meet the following conditions:

- Install Java JDK and the JDK version should be 1.8.
- Install the Java3D library (version 1.5). You can download the library file here: https://www.oracle.com/java/technologies/javase/java-3d.html and install the library following: https://download.java.net/media/java3d/builds/release/1.5.1/README-download.html.
- Download the MASON_14.jar file from NTULearn and use this jar file as the external library for compiling and running. Note that the latest MASON.20.jar file from MASON website is not compatible with the given Tileworld implementation, and thus should not be used.

#### 2.3.1 Agent Specifications

Each student in a team needs to design one agent. The designed agent must adhere to the following specifications:

- The agent can perceive only limited neighbouring cells. Mathematically, given the agent's position $(x, y)$ and an object's position $(X, Y)$, the object can be sensed only when $max(abs(x - X), abs(y - Y)) \leq 3$. See Figure 2 for an example.
- The agent has a limited set of actions that it can take. That is, {WAIT, MOVE_UP, MOVE_DOWN, MOVE_LEFT, MOVE_RIGHT, PICK_UP, DROP, REFUEL}.
- The agent can carry up to 3 tiles at a time.
- Every movement consumes one fuel from the agent. Once there is zero fuel, the agent can no longer move and will become stuck.
- The agent performs a Sense-Communicate (optional)-Plan-Act cycle once per time step.

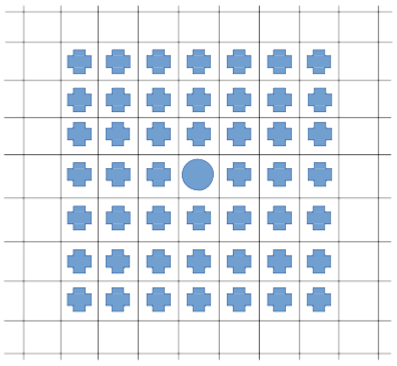
Figure 2: The visibility of an agent

- Your agent should inherit from the TWAgent class provided.
- Your agents may communicate via message passing. As all agents belong to the same team, all messages are broadcasted to every agent. However, an agent can still specify the intended recipient(s) in the messages. Also, there is no communication range limit.

### 2.3.2 Modules Needed to be Implemented

To meet the above agent specifications and obtain a high reward in the game, you are expected to implement the following functions:

- Planning module: in every step, the agent senses the objects around it and updates its memory. Based on the updated memory, the agent plans its action of current step. You can refer to the existing codes in “tileworld.planners” package and implement your own planner. A good planner is critical for obtaining a high reward.
- Memory module (optional): the agent stores the previously sensed information in its memory and does planning based on its memory. Therefore, the memory module can affect the performance of the agent. The “TWAgentWorkingMemory.java” file implements a basic memory module. You may create your own memory module by extending the “TWAgentWorkingMemory” class.
- Communication module (optional): in every time step, each agent can broadcast a message after it senses the environment. Currently, the “Message” class encodes each message. You can create your own message class by extending the “Message” class and use your own design to encode information. In the planning phase, your agent may use the messages from other agents for better planning. To retrieve all messages broadcasted in the current time step, just call the “getMessages()” method of the “TWEnvironment” class.

Note that to implement your own agent, you should create a new agent class by extending the “TWAgent” class. In your own agent class, you can override the “communicate()”[1], “think()” and “act()” methods. You should **NOT** override the other methods of “TWAgent”, e.g. “sense()” (and related “TWAgentSensor” class), “putTileInHole()” etc. Also, you should **NOT** modify the “environment” package of the simulator and the “increaseReward()” method of “TWEnvironment” class can only be called in the “putTileInHole()” method of “TWAgent” class. Any violations will lead to the downgrade of your final score.

[1]Note that in the “communicate()” method, the “receiveMessage()” method of the “TWEnvironment” class needs to be called so that the message can be added into the broadcasted message list of the environment.

# 3 ASSESSMENT

The assessment has two components:

- A report for each team member (maximum 2000 words and 8 pages including images/plots) outlining the agent you designed and its implementation. The report is 40% of the whole course assessment. The report should include:
  - A background section on Tileworld.
  - Analysis of different designs of the agents your team have tried and developed (this part can be the same for different reports from the same group).
  - A detailed description of the agent you have developed and tested. In this part, you should specify the connections between your agent and the whole team, e.g. the role your agent plays.
  - The results you have achieved for different agent designs.
- A team based 20-minute presentation and demonstration of your agents. This will form part of a competition. The presentation and demonstration are 20% of the whole course assessment.

## 3.1 Experiments

Tileworld is characterized by a configurable environment. The number of tiles, holes and obstacles created in every time step can be specified. Each object also has a lifetime, i.e. the number of time steps that the object will persist for. The configurations are specified in the "Parameters.java" file. The competition will use 3 different environment configurations, two of which will be known to you and one of which will be unknown to you.

- **Configuration One:**
  - Environment Size: $50 \times 50$ cells
  - Average Object Creation Rate: NormalDistribution($\mu$=0.2, $\sigma$=0.05)
  - Lifetime: $100$
- **Configuration Two:**
  - Environment Size: $80 \times 80$ cells
  - Average Object Creation Rate: NormalDistribution($\mu$=2, $\sigma$=0.5)
  - Lifetime: $30$
- **Configuration Three:**
  - Environment Size: $? \times ?$ cells
  - Average Object Creation Rate: ?($\mu$=?, $\sigma$=?)
  - Lifetime: ?

The following parameters are fixed in all cases:

- Total Time Steps : 5000
- Amount of fuel at beginning: 500

We will run 10 experiments for each configuration.  For each experiment, we will use a different random seed.

**Deadline**

Presentation and Demonstration will be in Week 12 and Week 13.
The due date of the report is: Sunday, 11:59:59PM of Week 13.
Note that FIVE(5) marks will be deducted for the delay report submission of each calendar day.