Lab 3_B: Implementing an Adaptation to the Discrete Smart Surface benchmark

To solve this lab, you must break down complex problems into manageable components and understand how they interact. Specifically, you will acquire familiarity with implementing a complex problem from a research paper.

This lab holds six goals:

1. Activate your software design and development skills.
2. Encourage you to be more intentional about your workflows: from the software architecture process to the development one.
3. Boost your problem-solving, software design, and development skills using OOP in Java.
4. Push you to be intentional about:
    a. Separation of concerns.
    b. Modular programming.
5. Keep adding new entries to your Dev Journal.
6. Get familiar with a research paper within the area of multi-agent systems.

Coding Policy

1. A note on using Grammar Checker tools such as Grammarly to answer textual questions: It is fine to use those as long as you DO NOT make use of generative AI features.
2. You will read and follow our Academic Integrity Policy.
3. You will follow our course's course coding policy.
4. ***Nope, you are not allowed to use AI -Powered Code Completion* Tools** in this assignment.
5. Read: Can You Explain Your Code?

Instructions

1) You are not allowed to split the task across team members: Everybody MUST collaboratively work on both parts.
2) Teams of max three people.
3) Remember to compile and document your thoughts into your Dev Journal.
4) A note on using Grammar Checker tools such as Grammarly to answer textual questions: It is fine to use those as long as you DO NOT make use of generative AI features.
5) What to hand? A pdf with your responses.

In Part A, you reflected on the architecture. Now that you have your reflections and software architecture in hand, you will implement a solution to the problem described below.

Problem Description.

- You will work with a simplification of the Discrete Smart Surface benchmark (reference: Independent reinforcement learners in cooperative Markov games: a survey regarding coordination)
- We described the benchmark here: Situated Learners in a Sequential Decision-Making Setting. This Programming Challenge's problem description is actually a simplification of that work.

Problem in details:

1. There is a 2D grid of N x M situated agents (see figure 1 below). At each time step, all agents individually pick actions to move an object.
    a. Agents individually pick their actions and feed them to the environment.
    b. *A* is the set of actions available to each agent, and A = {left, right, up, down, still}.
    c. For the purposes of this assignment, agents pick their actions randomly.
2. To accomplish the task, agents must successfully coordinate actions so that the object moves from the initial position, depicted by i in the image below, to the goal or terminal location g.
    a. A trial starts from placing the object at the initial position i and finishes once the object's top left reaches the goal state g. The object does not rotate and cannot leave the surface.
    b. A motion of the 2D object that is placed on the 2D grid's surface is determined by the Environment using a weighted sum of the agents' actions.
    c. Agents are dynamically categorized into three groups: agents by the object, agents under the object, and remaining agents. Each group has a weight that impacts the motion dynamics: wb, wu, ws, respectively, and {wb = 1, wu = 5, ws = 0}. If there is more than one action with the highest number, the environment chooses randomly among those. That process results in the *applied_action*.
3. The environment applies the *applied_action* so that the object moves from the *current_state* to the *next_state.* (If the *applied_action* = "stay still", even though the object does not move, there is a state transition.)
4. *E* is the width of the grid's right and left borders, which are penalty areas. Borders have equal width and height, such as 1 × N each.
    a. If the object reaches a penalty area, all agents continuously get a punishment of = -10 until the object moves to a non-penalty area. Otherwise, agents continuously receive a score of -1, independently of the object's motion. Finally, once the object reaches the goal/terminal location, all agents receive a score of 100, and the trial ends.
5. *S* is the set of states. Each possible object's position defines a unique state; hence, there are (M −m+1)(N −n+1) states, see figure 2.

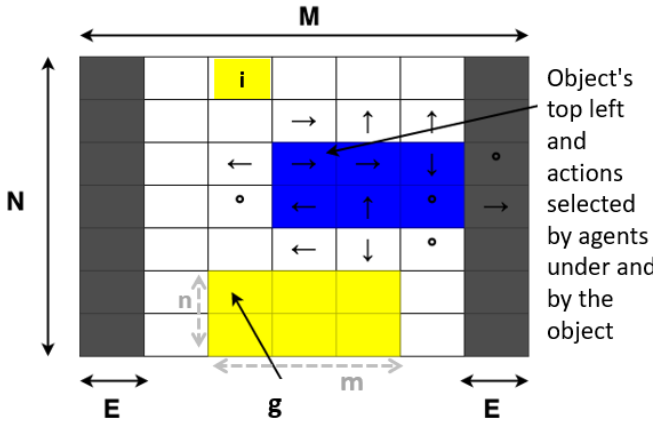

Figure 1. Each cell corresponds to a situated agent. The agents' task: to move the object from an initial position i into the goal state g (which defines the terminal state). The 3 × 2 object is color-coded in blue. Arrows inside the grid correspond to actions selected by agents under or by the object (circles for 'stay still'). The leftmost and rightmost columns are penalty areas, which agents also occupy.

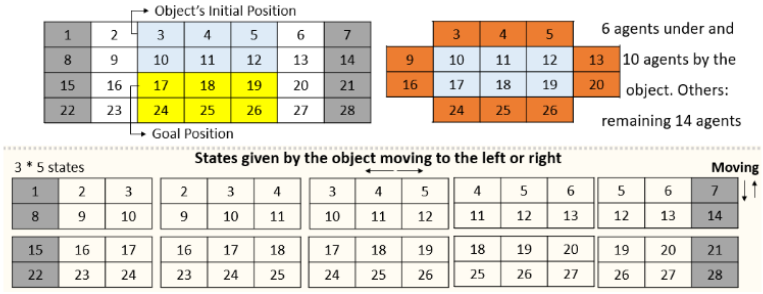

Figure 2. Upper left: a toy example; numbers within cells represent a situated agent. Upper right: a state in which 16 agents' actions play a role in the transition function F; remaining 14 agents are ignored ($w_s = 0$). Lower level: 15 possible states given by the grid and the object's dimensions (intermediary row omitted).

Figure 2, the lower part shows the states that follow as the object moves: five columns sideways by three rows in the up/down directions. Within the states, we show the corresponding agents that would be under the object. At each time step, the object's top-left provides s^t, the state s at the time step t.

Figure 2, upper right, depicts a state with the following agents under the object: 10-12 and 17-19. Now, see the interplay between state and agent role/importance: in this state, only 16 out of 28 agents play a role (agents by and under the object only). Hence, Figure 2 helps notice that the number of agents by the object varies as the object moves: for this example, 5 agents if at the corners, 7 if at the top or bottom states, 8 sideways, and 10 otherwise – whereas the number of agents under the object remains constant.

Implementation Details

1. You will implement the game dimensions as presented in Figure 1.
2. Since this is not an RL class, the agents will pick their actions randomly (equiprobable strategy for up, down, right, left).
3. Here are the agents' weights to impact the object's motions: wb = 2, wu = 5, ws = 1.
4. Make sure to make a clear distinction between agent and environment. Be clear about differentiating the agent's concerns from the environment's concerns.
5. At each time step, your program will print: the object's location and the number of agents that chose each of the four actions (*all* agents), and then according to their category (under, by, others). Once the object reaches the goal location, your program must print a greeting message.
6. To help you test your program, it will first ask the user to select one of two options: run the program in its default mode (according to the instructions above) or the fast mode.
    a. In the fast mode, everything is the same as above, except that agents under the object always select the action "down", whereas all the other agents pick their actions randomly as usual.

What to hand?

You will upload a pdf file only. In the pdf, the team will answer the questions below:

1. (1.0) You will watch the video: Programming Distributed Systems with Mae Milano, and then reflect and answer: what are the connections between this lab and the video?
2. (1.0) You will add screenshots of your program's output for the fast mode. You will also explain the sequence of images (what the output means).
3. (1.0) Reflect and answer: how does your program work?
4. (1.0) How did you dynamically identify what agents are under the object?
5. (1.0) What is the difference between the default and the fast modes?

6. (1.0) If you were to improve the agents, what would you do?
7. (1.0) In what ways did working on the software architecture (Lab, Part A) help you implement this project?
8. (1.0) If you were to explain your process of architecting and developing this project (Parts A and B) to a recruiter, what would you say?
9. (1.0) What was the most interesting thing you learned from this lab?
10. (1.0) What did you add from this lab into your Dev Journal and why?

Grading Scheme – Each question will be graded according to the rubric below:

For team submissions, only a single submission per team. (Make sure to add all team members!!!!)

100%	**70%**	**60%**	**40%**	**20%**
1. All parts of the question are correctly answered. 2. Answer is comprehensive and shows great level of reflection.	1. All parts of the question are correctly answered. 2. Answer is not comprehensive.	1. Most parts of the question are correctly answered.	1. Some parts of the question are correctly answered.	1. Very few parts of the question are correctly answered.