Autonomous Software Agents

Course Project

Master's degree in Artificial Intelligence Systems

A Course Project Submitted By

Sr #	Student Name	Reg. No.
01	Adnan Irshad	241763
02	Hira Afzal	241351

To

Prof. Paolo Giorgini
&
Dr. Marco Robol

UNIVERSITY OF TRENTO

Department of Information Engineering and Computer Science

Table of Contents

1. **Introduction** 3
2. **Delivery** 3
3. **Problem Statement** 3
4. **Implementation Strategy** 3
5. **Scope and Future Improvements** 7
6. **Results/outcomes**: 8
7. **References** 10

Table of Figures

Figure 1: Project Model Diagram 7
Figure 2: Multiple Agents working on Leader board. 8
Figure 3: Agent's Chat 9
Figure 4: Receiving Friend's Agent Information 9
Figure 5: Changing Intention if Both Agent's Target is same 10

1. Introduction:

This report presents the submission for the Autonomous Software Agents course project, which aimed to develop a sophisticated software capable of playing on behalf of the user, with the objective of collecting parcels and delivering them to the designated delivery zone to earn points. The project utilized a Belief-Desire-Intention (BDI) architecture, enabling the software to sense the environment, manage beliefs, deliberate intentions, and select plans from a library. Additionally, an external planner component was integrated to assist in plan selection and execution of actions using A* search algorithm. The software was implemented using NodeJS as the platform. Furthermore, an essential aspect of the project involved establishing effective coordination and collaboration between multiple agent instances to facilitate information exchange about the environment, including package locations and agent positions. This report outlines the problem statement, project architecture and techniques employed to accomplish these objectives and presents the results and outcomes of the project.

2. Delivery

Inside the delivered zip file, along with this report, it can be found:

- Agent Folder:
  - domain.pddl, the PDDL domain file, problem.pdll, the PDDL problem and other necessary JS files.
- Utils folder:
  - Containing all the scripts and additional files used to to support the main files in agent folder.
- Deliveroo.js-master: Will be provided in case to run the project locally.

3. Problem Statement:

The project focuses on two main tasks: Task-1 and Task-2.

Task-1: Agent A - Basic Requirements

In Task-1, the objective was to develop Agent A (Champ), which is a single agent capable of collecting packages and delivering them to designated delivery zones. Agent A needs to exhibit basic functionalities, including sensing the environment, revising its beliefs and intentions based on new information, and generating plans using a planner based on the Planning Domain Definition Language (PDDL).

Task-2: Coordinated Agent Team - Agent B and C

Task-2 focuses on developing a team of two coordinated agents, namely Agent B and Agent C. The objective was to establish effective coordination and collaboration between these agents. In response to problem statements, we developed two agents named Agent B (Adnan) and Agent C (Hira) to exchange information about the environment, such as the location of packages and the positions of other agents.

This project report covers both individual agent development and team coordination aspects, providing a comprehensive understanding of agent-based systems and multi-agent coordination mechanisms.

4. Implementation Strategy:

Task 1: Implementation Strategy: In task 1, our strategy was to implement the behavior of the agent within the Deliveroo environment. We focused on several key aspects, including perception, belief formation, intention generation, planning, and execution of actions.

1. Perception and Belief Formation:

- We used the Deliveroo API to gather information about the environment, including the agent's position, parcels, and other agents.
- Also interacting with the map, such as retrieving map dimensions, finding paths, and identifying closest spots based on position and other parameters.
- The Beliefs class was responsible for updating and maintaining the agent's beliefs based on the perceived information.
- Challenges faced: Handling asynchronous events and ensuring that the beliefs were updated correctly and in a timely manner.
- **Code Explanation:**

  class Beliefs: This class represents the beliefs of an agent in the Deliveroo simulation.

  Constructor (client, grid, me): Initializes the beliefs class with the Deliveroo client, grid map, and agent's data.

  load_exploration_spots(): Loads predefined exploration spots on the grid map for the agent to visit.

  getAgentBeliefs(): Returns a reference to the agent's beliefs about other agents.

  getParcelBeliefs(): Returns a reference to the agent's beliefs about parcels.

2. Intention Generation:

- The Desires class was responsible for generating intentions based on the agent's beliefs and goals.
- Desires were filtered based on certain criteria to prioritize the most relevant and achievable intentions.
- Challenges faced: Determining the appropriate desires and filtering criteria to generate meaningful intentions.
- **Code Explanation:**

  class Intentions: This class manages the intentions of an agent in the Deliveroo simulation. It has an array of **intention** to store the intentions and a reference to the **beliefs** object.

  constructor(beliefs): Initializes an Intentions object with the given beliefs.

  filter(desires): Filters desires to determine the best intention(s) to achieve.

  reconsider(): Checks if a different intention with better utility should replace the current intention.

  utility(option, bag_score, bag_size): Calculates the utility score for a given intention option.

3. Planning:

- The Plan class is responsible for generating and managing a plan of actions to achieve a specific goal (intention). It uses the PDDL (Planning Domain Definition Language) to formulate the problem and generate a plan.
- We created list of actions using DeliverooAPI.PddlClient by defining action name, parameters, preconditions, effects of each action by an agent from a position to another.

- Same created a PDDL problem using DeliverooAPI by defining map variables (x, y), parcel & agents predicates from Intention and Beliefs.
- Then we generate PDDL files for domain and problem, which further used in generating the plan using A* search algorithm.
- **Code Explanation:**

  plan class: This class used the domain file (PDDL problem file) and online solver to generate a plan for achieving the current intention.

  constructor(intentions, beliefs): function initializes the `Plan` object with the given intentions and beliefs. It prepares the PDDL problem by creating the domain, variables, predicates, and objective based on the current beliefs and intention.

  getPlan(domain): method used to call the PDDL solver to generate a plan. It uses the `onlineSolver` function from the `@unitn-asa/pddl-client` library to interact with the solver. If the goal is to "wait," it simply adds the "wait" action to the plan and resolves immediately. Otherwise, it calls the PDDL solver and receives the plan steps. It then updates the `action_list` and `shortest_path` arrays with the extracted actions and coordinates respectively.

  is_sound(intentions, beliefs): method verifies if the current plan is sound, considering the agent's beliefs and intentions. It checks for obstacles in the path, the presence of parcels (if the intention is to pick up), and the validity of the plan based on the score of the remaining parcels.

  pop_front(): method retrieves and removes the first action from the `action_list`. If the list is empty, it returns an "error" indicator.

4. Action Execution:

- The execute() method in the Agent class was responsible for executing individual actions based on the plan.
- We utilized the Deliveroo API to perform actions such as movement, picking up parcels, and putting them down.
- Challenges faced: Coordinating the execution of actions with the agent's beliefs and intentions and handling any errors or failures during execution.

Task 2: Implementation Strategy

In task 2, our strategy was to enable communication between two agents and facilitate the exchange of relevant information for decision-making.

1. Communication Setup:

- We utilized a DeliverooAPI socket.io events to exchange messages between the agents.
- The Agent class utilized the client.onYou & client.onMsg functionality to send and receive messages.
- Challenges faced: Ensuring the authentication with server.

2. Shared Data:

- Agents shared information about their current state, including their ID, name, location, and score.
- Agents also shared their intentions, particularly the IDs of parcels they intended to pick up next.

- Sharing data about the environment one agent sensed including parcel's data and opponents' data in its range in the environment.

3. Actions Based on Received Information:

- When an agent received a message from another agent, it processed the information to update its beliefs and intentions accordingly.
- Agents compared their own intentions with the received intentions and took appropriate actions based on the shared information.
- Challenges faced: Synchronizing the agents' intentions and handling conflicts when multiple agents have the same intention.

Code Explanation:

this.client.onYou((me) => {...}):

- This function is an event handler that is triggered when the agent receives information about itself.
  - It retrieves the next intention target from the agent's intentions and assigns it to the variable next_intention_target.
  - If there are no intentions in the list, it sets `next_intention_target` to null.
  - It retrieves the free parcels in the environment from the agent's beliefs using the getFreeParcels() method and assigns them to the variable parcels_in_env.
  - If no parcels are found, it sets parcels_in_env to null.
  - It retrieves the opponent agents' information from the agent's beliefs and constructs an array of sensed opponent agents in the sensed_opponents_agents variable.
  - It creates an env_data object that includes the parcels in the environment and the opponent agents' information.
  - It creates a my_data_str object that includes the agent's own data, next intention target, and environment data.
  - The my_data_str object is converted to a JSON string using JSON.stringify().
  - It sends the agent's information to the friend agent using the this.client.say() method.

this.client.onMsg((id, name, msg) => {...}):

  - It is an event handler that is triggered when the agent receives a message from the friend agent.
  - It parses the received message from JSON format using JSON.parse().
  - The agent's friend agent's data, next intention target, and environment data are assigned to the respective variables (friend_data, friend_next_intention_target, & friend_env_data).
  - It prints the friend agent's identifier, name, location, and score.
  - It prints the friend agent's next intention target.

this.client.say(friend, message):

  - This function sends a message to the specified friend agent.
  - The friend parameter represents the identifier or name of the friend agent to whom the message is being sent.
  - The message parameter represents the content of the message being sent.
  - The function facilitates communication between the current agent and the friend agent by exchanging information in the form of messages.
  - The message can be in any format, but it is common to use JSON for structured data exchange.

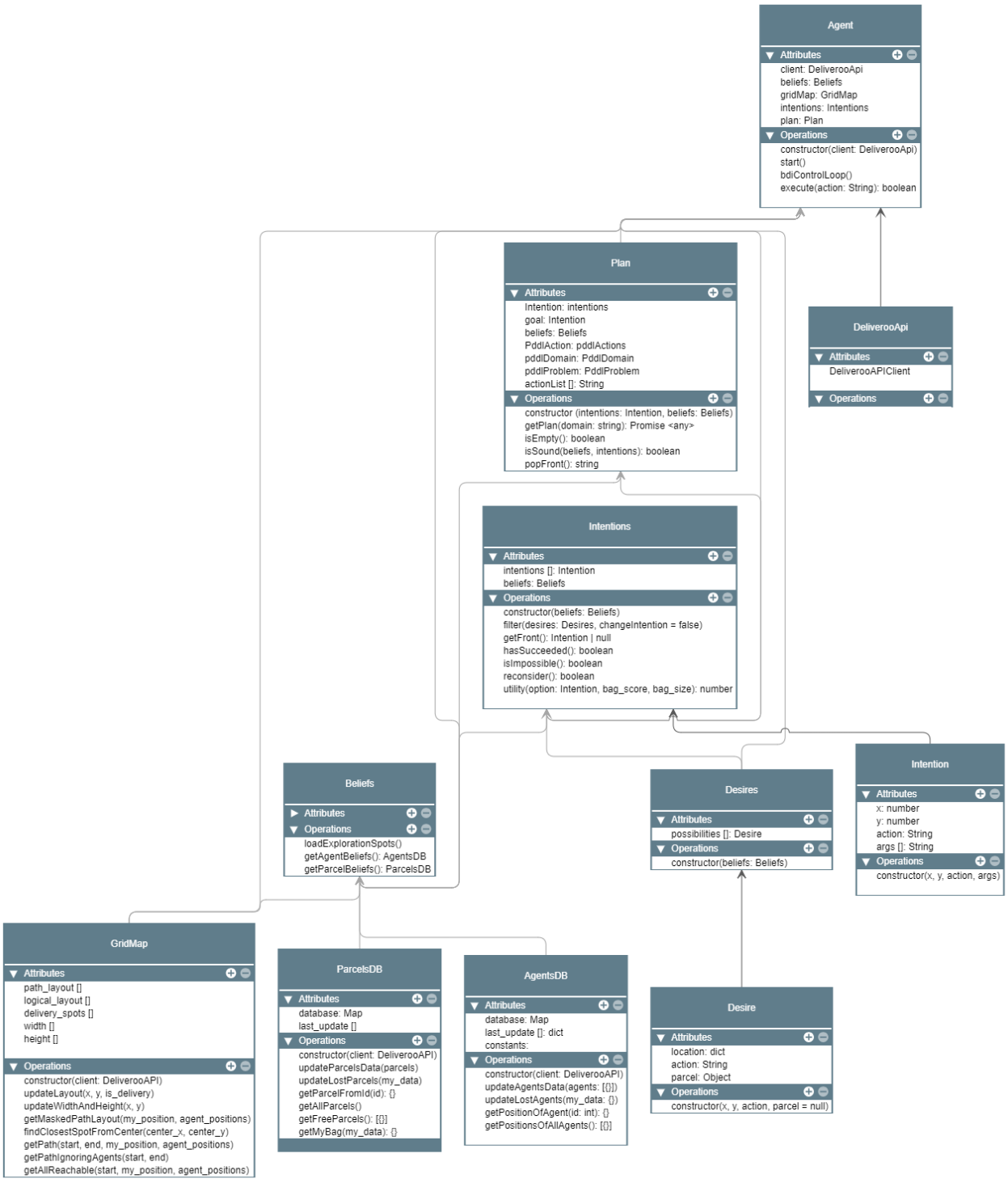

Figure 1: Project Model Diagram

5. Scope and Future Improvements:

The scope and potential future improvements of the Deliveroo Agent project include:

- ***Advanced Planning and Decision-Making:*** The project can explore more advanced planning algorithms and decision-making techniques to enhance the agent's ability to generate optimal plans and make intelligent decisions in dynamic and uncertain environments. This could involve integrating techniques such as reinforcement learning or hierarchical planning.

- ***Learning and Knowledge Sharing:*** The project can incorporate machine learning techniques to enable agents to learn from their experiences and improve their performance over time. Agents can share knowledge and learn from each other, allowing for collective intelligence and improved coordination.
- ***Enhanced Communication and Collaboration***: The communication and coordination mechanisms between agents can be further improved to enable more sophisticated collaboration and allowing for better coordination and joint decision-making.
- ***Making decisions on received Information from Friend's Agent:*** After getting the information from friend agent currently, we are changing the intention. It can be further improved to regenerate strategies for parcel picking. Also, agent needs strategy to beat the opponents after getting information from friend agent.
- ***Optimization and Efficiency:*** The project can explore optimization techniques to improve the efficiency of the agent's actions and delivery routes. This can involve techniques such as route optimization algorithms, load balancing, and resource allocation strategies to minimize delivery time, energy consumption, or other relevant metrics.

6. Results/outcomes:

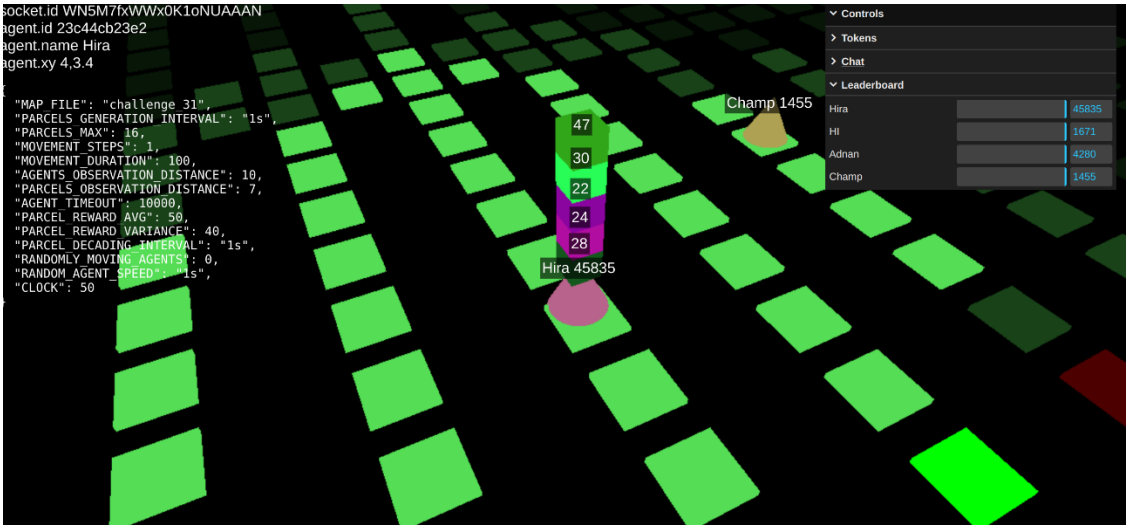

Figure 2: Multiple Agents working on Leader board.

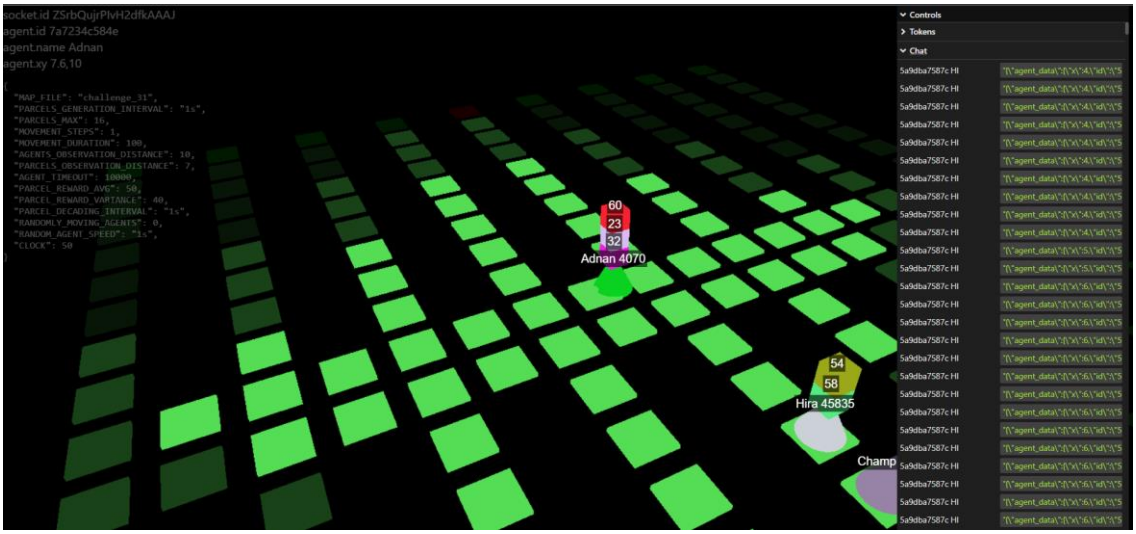

Figure 3: Agent's Chat

```
//////Friend Agent Data////////
Friend ID:  23c44cb23e2
Friend Name:  Hira
Location:  5 10
Score:  46265

//////Friend Environment Data////////
Parcels Data:  [
  { id: 'p374', x: 0, y: 0, carriedBy: null, reward: 1 },
  { id: 'p385', x: 14, y: 0, carriedBy: null, reward: 12 },
  { id: 'p399', x: 2, y: 0, carriedBy: null, reward: 27 },
  { id: 'p416', x: 14, y: 19, carriedBy: null, reward: 1 },
  { id: 'p426', x: 0, y: 19, carriedBy: null, reward: 19 },
  { id: 'p428', x: 2, y: 19, carriedBy: null, reward: 8 },
  { id: 'p429', x: 4, y: 19, carriedBy: null, reward: 29 },
  { id: 'p434', x: 16, y: 19, carriedBy: null, reward: 13 },
  { id: 'p439', x: 12, y: 19, carriedBy: null, reward: 50 },
  { id: 'p440', x: 10, y: 19, carriedBy: null, reward: 74 }
]

Opponents Agents Data:  [
  { id: '064d4d000e2', name: 'Champ', position: [ 8, 5 ], score: 1455 }
]

//////Friend Next Intention of Parcel Picking////////
Friend's Next Parcel Intention:  p429
```

Figure 4: Receiving Friend's Agent Information

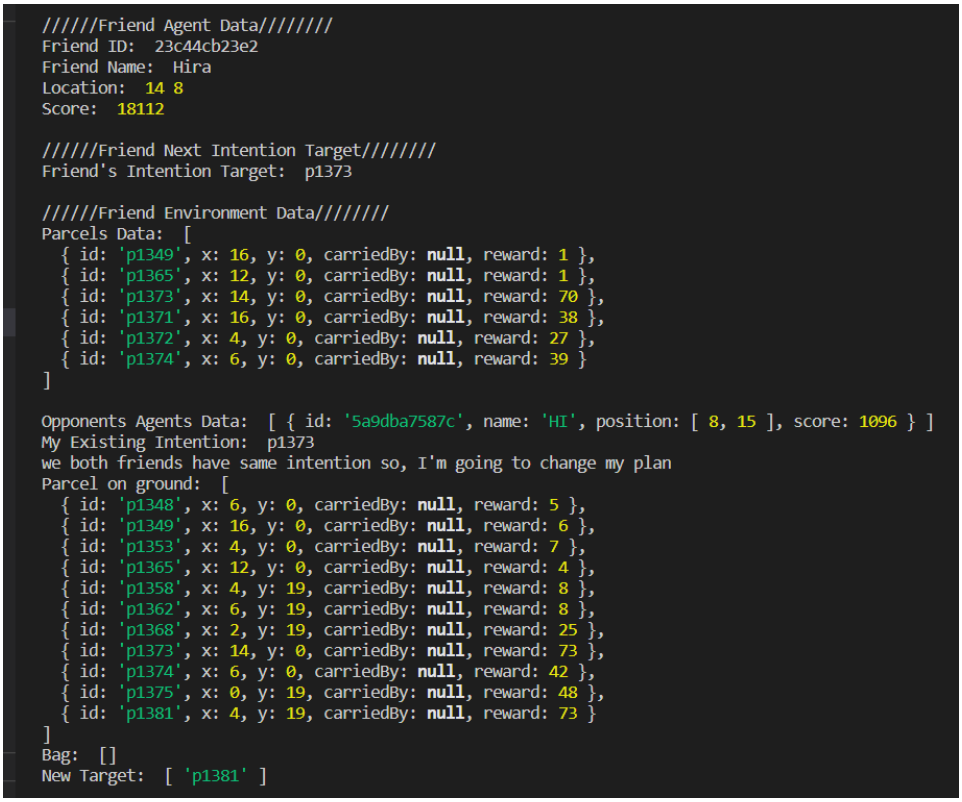

Figure 5: Changing Intention if Both Agent's Target is same

Overall, the project aims were developing intelligent agents capable of autonomously performing delivery tasks in a simulated environment. The project utilizes perception, belief formation, intention generation, planning, and action execution to enable agents to operate effectively. The implementation strategy involves integrating the Deliveroo API and PDDL for environment interaction and developing classes like Beliefs, Intentions, and Plan to manage agent behaviour.

7. References

1. unitn-ASA/Deliveroo.js (github.com)
2. A* Search Algorithm - GeeksforGeeks
3. bgrins/javascript-astar: A* Search / Pathfinding Algorithm in Javascript (github.com)