Department of Information Engineering and Computer Science (DISI)

Autonomous Software Agents Course

Report

Taras Rashkevych
239948
taras.rashkevych@studenti.unitn.it

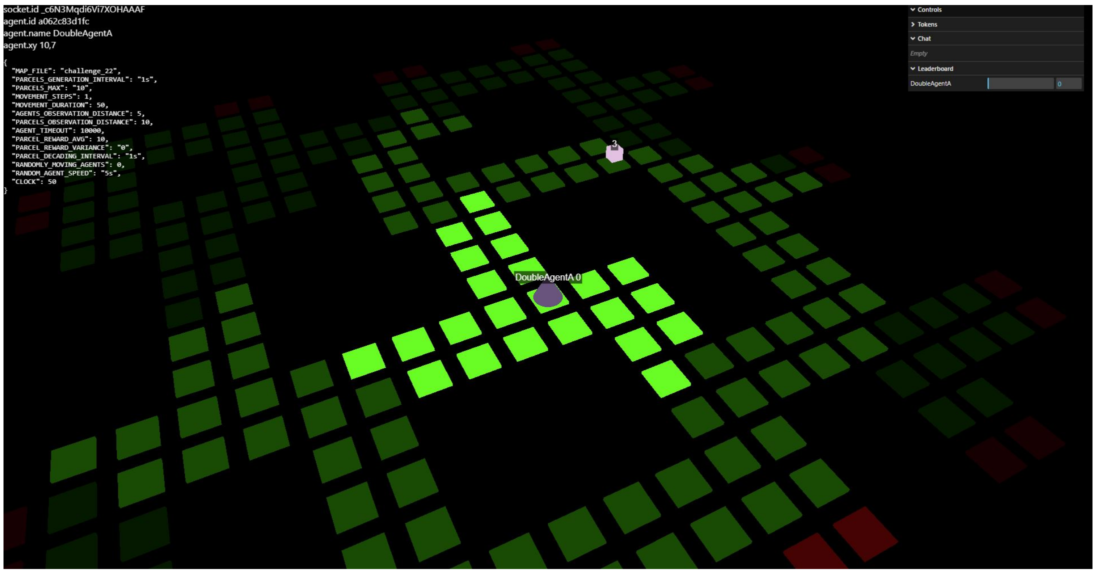
Alessandro Rizzetto
247542
alessandro.rizzetto@studenti.unitn.it

Academic year 2023/2024

Indice

1 Introduction

The aim of this work is to show how some theoretical concepts about autonomous software, such as belief–desire–intention (BDI) architecture and planning, can be practically applied in a concrete context of the so-called software agents that are involved in a 3D board game. The game is made of a board, whose layout changes based on the configuration, with three types of cells: the first type of cell on which agents can move, the second type of cell that plays the rule of a wall, therefore impossible to navigate, and the third type of cell that serves as a delivery destination. During the gameplay, some parcels appear on the first type of cell with some points assigned to them, which could decrease with time or remain stable depending on the board's configuration. The agents' final goal is to gain as many points as they can by delivering the parcels to the delivery cells. To prove the validity of the theoretical concepts and the quality of the chosen implementation, two main scenarios are taken into account: the first one regards a single autonomous software agent that tries to maximize its score and the second one regards a pair of autonomous software agents that try to achieve the same goal also by collaborating. In the next sections, some concrete implementation details are presented to show the chosen strategies for the two scenarios and the results that the agents can achieve in different board configurations.


```
socket.id _c6N3Mqdi6Vi7XOHAAAF
agent.id a062c83d1fc
agent.name DoubleAgentA
agent.xy 10,7

{
  "MAP_FILE": "challenge_22",
  "PARCELS_GENERATION_INTERVAL": "1s",
  "PARCELS_MAX": "10",
  "MOVEMENT_STEPS": 1,
  "MOVEMENT_DURATION": 50,
  "AGENTS_OBSERVATION_DISTANCE": 5,
  "PARCELS_OBSERVATION_DISTANCE": 10,
  "AGENT_TIMEOUT": 10000,
  "PARCEL_REWARD_AVG": 10,
  "PARCEL_REWARD_VARIANCE": "0",
  "PARCEL_DECADING_INTERVAL": "1s",
  "RANDOMLY_MOVING_AGENTS": 0,
  "RANDOM_AGENT_SPEED": "5s",
  "CLOCK": 50
}
```

Figure 1: *Example of a situation in the Deliveroo game within Challenge 21*

2 Single Autonomous Software Agent

This section describes the actual implementation details of a single autonomous agent exposed through the entry point method `play` that is called at the moment of the creation of the agent.

2.1 Architecture

To have full and fine-grained control over the changes in the state of the agent and its actions it has been decided to use an event-driven architecture based on the official Node.js event emitter object called EventEmitter. The fundamental idea is that given some concrete situation for example when a plan can be generated or the agent has no plan to execute, the corresponding event is raised to communicate this information so the agent can take some particular action in response. The following are the events that can be emitted:

- `explore`: used to initiate an exploration technique of the map;
- `generatePlan`: used to initiate a plan generation;
- `restart`: used to restart the usual agent logic flow;
- `parcelsOnTheWay`: used to initiate the collection of nearby parcels.

In this manner, it is possible to instruct the agent to perform actions without being tied to some state variables that decide what the agent should do. This also simplifies a lot the development process and allows adding new behavior to the agent by simply defining new types of events and how to react to them.

2.2 Environment Sensing

At the moment of the creation of the single autonomous agent, the event listeners for the game are set to perceive the surrounding environment and to get some general data about the map. The following are the listeners:

- `onConnect`: gets the socket id upon connection;
- `onDisconect`: gets the socket id upon disconnection;
- `onConfig`: gets the data about the agent speed and the decading time of the parcels;
- `onYou`: gets the data about the single agent like its position and score;
- `onAgentsSensing`: gets the data about the surrounding agents;
- `onParcelSensing`: gets the data about the surrounding parcels;
- `onMap`: gets the data about the map cells and their types.

Other listeners as `onTile` and `onNotTile` have not been used because all the listed ones are sufficient for achieving the final goal of the agent.

2.3 Beliefs and Intentions Revision

The revision part of the autonomous software agent is split into two parts: the revision of its beliefs, that is all the information that represents its state and the surrounding environment, and the revision of its intentions, that is the goals that the agent tries to achieve with the information contained in its beliefs.

- The revision of the beliefs is continuously done thanks to the listeners defined above and the *no memory* approach is used in this case to be able to have the most fresh information and avoid making false assumptions about past data, such as the position of other agents or parcels. However, if the listeners don't provide any new information because the visible surrounding environment is not changing, then a *exploration* strategy is applied to explore the environment. The strategy is implemented by the method `explore` and its two fundamental pieces, which are the methods `generateExplorationPlan` and `exploreRandomly`:

  – `generateExplorationPlan`: this method works by considering the division of the whole map into four distinct sectors called quadrants and one of these is chosen to be the target quadrant the agent should reach by generating a plan for the actual target cell of the quadrant that would be the centroid of that quadrant. The default choice for the quadrant is the one corresponding to the current quadrant of the agent, but the choice will dynamically and randomly change based on the evolution of the game. This approach is especially useful when the map is quite big, so the agent has a good chance to explore it better by avoiding to be located inside the same piece of the map;

  – `exploreRandomly`: this method works by letting the agent randomly move in one of the four available directions so that if the move is possible then the agent will follow that path, otherwise it will choose one of the other three directions to try a different path. This random exploration will end if there are new parcels around or if the maximum number of random moves has been reached, which corresponds to half the width of the map. The default direction is the one that brings the agent up with respect to the coordinate system of the map. This approach is especially useful for local exploration so that if some parcels are not too far away from the agent then it will find them;

  Each of the two methods has a 50% chance of being used to make a bit more general the overall exploration strategy.

- The revision of the intentions on the contrary depends on the following idea: *the agent tries to get the most rewarding parcel*, which means that it will give maximum priority to the parcel that will give him the highest score by considering the points lost to reach and deliver the parcel. Based on this idea the agent will revise its intentions in three different situations:

  – `No plan`: if for some reason there is no plan to reach the parcel then it will try to get the best one after the beliefs revision by emitting the handling the `restart` event;

  – `Failed move`: if for some reason the agent cannot finish the intended move then it will try to explore the surrounding environment through the `explore` method and then will emit and the `restart` event to get the best option available for the visible parcels;

  - `Near parcels`: if the agent is able to pick up a parcel and at that moment there are other parcels at a certain distance specified by the `MAX_NEAR_PARCEL_DISTANCE` environment variable and with a certain number of points specified by the `MIN_NEAR_PARCEL_REWARD` environment variable. Then it will emit the `parcelsOnTheWay` event and will handle it by trying to pick the nearest parcel and will repeat this strategy recursively until there are no parcels in the range to ultimately deliver all the collected parcels to the nearest delivery cell.

2.4 Plan Generation and Execution

The plan generation of the single agent is handled through the use of `generatePlanToParcel`, `generatePlanFromParcel` and `generateExplorationPlan` methods that use the PDDL formal language. The domain file in this case is called `board-domain.pddl` and is defined only once because the general problem remains the same, meanwhile the problem file is called `problem.pddl` and is redefined every single time the agent wants to reach a parcel or a specified cell during the exploration phase. The following is the content of the domain file:

- `Predicates`: (tile ?t), (delivery ?t), (wall ?t), (agent ?a), (parcel ?p), (me ?a), (right ?t1 ?t2), (left ?t1 ?t2), (up ?t1 ?t2), (down ?t1 ?t2), (at ?agentOrParcel ?tile), (carriedBy ?parcel ?agent)
- `Actions`:
  - (:action right :parameters (?me ?from ?to) :precondition (and (me ?me) (not (wall ?from)) (not (wall ?to)) (tile ?from) (tile ?to) (at ?me ?from) (right ?from ?to)) :effect (and (at ?me ?to) (not (at ?me ?from))))
  - (:action left :parameters (?me ?from ?to) :precondition (and (me ?me) (not (wall ?from)) (not (wall ?to)) (tile ?from) (tile ?to) (at ?me ?from) (left ?from ?to)) :effect (and (at ?me ?to) (not (at ?me ?from))))
  - (:action up :parameters (?me ?from ?to) :precondition (and (me ?me) (not (wall ?from)) (not (wall ?to)) (tile ?from) (tile ?to) (at ?me ?from) (up ?from ?to)) :effect (and (at ?me ?to) (not (at ?me ?from))))
  - (:action down :parameters (?me ?from ?to) :precondition (and (me ?me) (not (wall ?from)) (not (wall ?to)) (tile ?from) (tile ?to) (at ?me ?from) (down ?from ?to)) :effect (and (at ?me ?to) (not (at ?me ?from))))
  - (:action pickup :parameters (?me ?parcel ?tile) :precondition (and (me ?me) (parcel ?parcel) (tile ?tile) (at ?me ?tile) (at ?parcel ?tile) (not (carriedBy ?parcel ?me))) :effect (and (not (at ?parcel ?tile)) (carriedBy ?parcel ?me)))
  - (:action putdown :parameters (?me ?parcel ?tile) :precondition (and (me ?me) (parcel ?parcel) (tile ?tile) (delivery ?tile) (at ?me ?tile) (carriedBy ?parcel ?me)) :effect (and (at ?parcel ?tile) (not (carriedBy ?parcel ?me))))

The above definitions are sufficient to generate a plan for the agent to reach a particular parcel and deliver it to a delivery cell through the `executePlan` method and reach a centroid through the `executeRandomPlan` method during the exploration phase. The particular aspect of the above definition is the fact that other agents are not taken into account as preconditions for the plan to avoid making false assumptions about their position.

3 Multiple Autonomous Software Agents

The second step undertaken in this project involves defining a multi-agent environment where two agents, namely doubleAgentA and doubleAgentB, collaborate to achieve better performance within the game.
Initially, we implemented all the necessary functions for communication between the two agents and the conditions under which they occur. Subsequently, we defined coordination strategies between the two agents focusing on robustness, generalization, and performance in the proposed challenges.

3.1 Approach

Regarding the multi-agent strategies, two strategies have been implemented which agents autonomously choose based on an analysis of the map structure. During development, the presence of delivery cells reachable only through long corridors with a single entrance proved significant and impactful, leading to instances where agents tended to obstruct each other.
By utilizing the *corridorFounder()* function, the agents analyze the map structure. If the majority of delivery cells are reachable through easily obstructed corridors, they adopt the *"Strategy that fits the map"*; otherwise, they resort to the more basic *"Extended knowledge strategy"*.

3.2 Extended Knowledge Strategy

This strategy represents the most basic approach where the two agents, unable to leverage strategies based on map shape, behave similarly to single agents with extended knowledge. Specifically, the two agents share all available knowledge about parcels, other agents, and themselves with their teammate.
Upon receiving and decoding messages, the agents appropriately update their environment knowledge and can make better decisions regarding parcels to collect and routes to take. This approach allows the two agents to enhance their individual performances by having a broader and more in-depth map analysis capability.
During development, some tests were conducted by assigning specific areas of expertise to the agents within the map to entirely avoid the possibility of them obstructing each other or both ending up in areas of the map with few parcels. However, this solution yielded comparable results, highlighting difficulties in maps with few parcels or delivery cells.

3.3 Strategy That Fits the Map

This coordination strategy involves defining two new potential roles that agents can embody: winner and supporter.
At the beginning of the game, both agents start playing as single agents, behaving exactly as described in the preceding sections. Meanwhile, at regular intervals, a message is sent to communicate the presence to the teammate. Once the other agent also begins playing, the setup phase begins. At this point, armed with information about their teammate, the two agents are assigned the roles of winner and supporter.
The objective of this strategy is to enable the winner agent to earn more points than they would collect on their own by receiving assistance from the supporter, who transports parcels to strategic positions. Alongside sharing environmental information, the

two agents also exchange information about their `mental state`, aiming to aid without hindering each other as much as possible.
When the winner agent devises a plan to collect a parcel, it is communicated to the teammate, who adds the parcel ID to a blacklist of options to be disregarded during objective selection. This information minimizes interference between agents and allows them to choose different parcels if they happen to be close to each other.
This strategy operates based on the definition of a checkpoint cell. Through analysis of the map structure, the supporter agent delivers all collected parcels to this checkpoint cell, strategically positioned. The winner agent can then collect parcels left by the teammate each time they make a delivery.

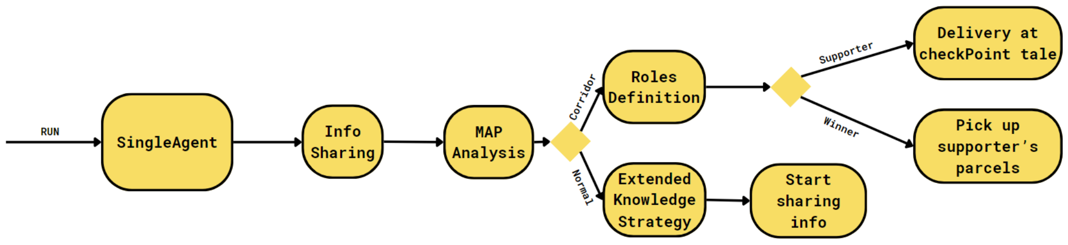


Figure 2: *Scheme of how the Strategy That Fits the Map works*

3.4 Communication Between the Two Agents

To streamline management, communication between the two agents solely employs two functions: *OnMsg* for message reception and *say()* for message transmission.
Upon each call to the *OnParcelsSensing* and *OnAgentsSensing* functions, the agents invoke the *say()* function, passing a message containing the relevant information they deem necessary to send. They employ the same methodology to communicate information about their position, score, mental state, and role assumed in the strategy.
These messages, to be sent as strings and correctly understood by the recipient, undergo encoding and decoding based on the message class (*'parcels'*, *'agents'*, *'mentalState'*, *'strategyInformations'*, *'mateInfo'*).
Upon receiving the information contained in the message, the agents appropriately update their local variables.

4 Testing and Data Analysis

To assess the functionality and performance of the agents, we conducted single-agent tests in challenges 21, 22, 23, and 24. As per the instructions, the two multi-agent agents were tested in challenges 21, 32, and 33.
Each challenge was tackled for five minutes and repeated three times to mitigate variance related to environmental conditions.
Challenges 31 and 33 involved special cases where the two agents could find themselves isolated: in challenge 31, within the same corridor where neither agent could collect nor deliver, and in challenge 33, isolated in two separate, unconnected map sections. This was done to scrutinize and analyze the behavior of the agents and their respective strategies under the proposed extreme scenarios.

4.1 Single Agent testing

Challenge 21 is handled quite well by the single agent, in fact, its ability to resume its movement after not being able to move due to the presence of other agents lets it effectively collect the visible parcels. There are even situations during the run of the first test in which the parcels are very localized, so by following its intentions revision strategy the agent is able to pick them all.
Challenge 22 is more difficult to tackle since the available points on each parcel are quite small so the agent has to be smart about picking and delivering the parcels. However, the obtained results show a very good consistency over the three tests, which confirms its stability even in different runs.
Challenge 23 is handled quite well, the overall score is very high compared to the other challenges and is also very consistent through the different runs as in the previous challenge.
Challenge 24 is more difficult to handle because the parcels are quite sparse and with very varying scoring points, so sometimes the agent is lucky to find the parcels with high scores as during the first test and other times it is less lucky as during the second test where mainly parcels with few points are picked.

Challenge 21	**Test 1**	**Test 2**	**Test 3**	**Mean**
Single Agent	480	290	220	330
Challenge 22	**Test 1**	**Test 2**	**Test 3**	**Mean**
Single Agent	306	372	276	318
Challenge 23	**Test 1**	**Test 2**	**Test 3**	**Mean**
Single Agent	1898	2041	1831	1923
Challenge 24	**Test 1**	**Test 2**	**Test 3**	**Mean**
Single Agent	786	484	646	639

Table 1: Results on single agent's challenges

4.2 Multi-Agent Testing

Challenge 31 is successfully tackled by the two agents, wherein after an initial setup phase, justifying the points collected by the supporter in the second test, the winner significantly boosts the score in single-agent mode.
Challenge 32 is overcome by the agents, albeit with some difficulty. The exploration technique adopted is not ideal for this challenge, and the randomness of movements is reflected in the results, which indeed exhibit the most pronounced variance. Our proposed code provides for handling the case in which the two agents are in the same corridor, however, this only works if there is proper coordination between the two agents of a random nature that cannot be reproduced in all starting conditions.
Challenge 33 is successfully navigated, both in Case A where the two agents are on separate map portions and in Case B where they share the same portion. In Case B, the scores are lower due to the map's small size and few parcels, offering limited alternatives to the agents.
The results highlight the consistency of the adopted strategies, enabling the attainment of high and fairly similar scores. Throughout each test, it was observed how the agents, over time, expedited their movements using previously saved plans in memory.

Challenge 31	**Test 1**	**Test 2**	**Test 3**	**Mean**
Winner	1041	1185	1325	1183
Supporter	0	67	0	22
Challenge 32	**Test 1**	**Test 2**	**Test 3**	**Mean**
Winner	934	764	751	816
Supporter	715	797	1042	844
Challenge 33A	**Test 1**	**Test 2**	**Test 3**	**Mean**
Winner	1305	1296	1213	1271
Supporter	1248	1332	1317	1299
Challenge 33B	**Test 1**	**Test 2**	**Test 3**	**Mean**
Winner	591	523	384	499
Supporter	0	0	0	0

Table 2: Results on multi-agent's challenges

5 How to Run

To execute the code, it's necessary to firstly run the game repository shown in the references. It's also necessary to start the server on a map or challenge of the user's choice. Then the user can go to the localhost address and enter the tokens in the .env file to monitor the execution of the game.
Due to the unavailability of the online solver, a Docker environment has been set up to replicate its functionalities. The reference Dockerfile is available in the project's main folder.
After generating the image, build the container and it's necessary to change the URL inside the PddlOnlineSolver.js file of the @unitn-asa/pddl-client library on the line 21 to point to the local instance of the planner.
To run the code at this point, simply navigate to the project folder and execute one of the following commands:

- `npm i`: installs the the node modules;
- `npm run singleAgent`: launches an agent in singleAgent mode;
- `npm run doubleAgentA`: launches an agent with a predisposition to collaborate;
- `npm run doubleAgentB`: launches the second agent with a predisposition to collaborate.

This setup allows for the continuation of the project's development without dependency on the online solver.

6 Conclusions

In this report, a solution of autonomous software agents capable of playing the game Deliveroo both individually and coordinating their actions is presented.
Utilizing the Belief-Desire-Intention (BDI) architecture and planning, the agents can autonomously and successfully engage in the game. The proposed solutions demonstrate how the single agent can perceive the environment, revise beliefs, revise intentions, and formulate plans using a PDDL-based planner.
Regarding the multi-agent approach, two strategies have been proposed: one simpler and more generalizable, and one more elaborate but specific to certain types of maps. In the multi-agent approach, the two agents are capable of exchanging information about the environment, such as the position of parcels, agents, or themselves. They can also communicate their mental states to better coordinate the parcels to collect.
The agents have been tested in dedicated challenges, and the results reported in Table 1 and Table 2 emphasize their potentials and critical aspects.

6.1 Future Developments

Potential future developments will focus particularly on the creation of multi-agent plans based on the information contained in the agents' mental states. This would further enhance the performance and coordination of the two agents, as well as pave the way for new, more complex and effective strategies.
Another area for potential improvement is the handling of certain edge cases, such as that of challenge 32, which were not addressed in this project.

References

[1] GitHub repository of the project. *Available online:* `https://github.com/TarasRashkevych99/autonomous-software-agent`

[2] GitHub repository of the game. *Available online:* `https://github.com/unitn-ASA/Deliveroo.js`

[3] GitHub repository of lesson laboratories. *Available online:* `https://github.com/unitn-ASA/DeliverooAgent.js`

[4] PDDL online solver website. *Available online:* `http://planning.domains/`

[5] PDDL documentation website. *Available online:* `https://planning.wiki/ref/pddl/domain`