# Deliveroo

Autonomous Software Agents - A.A. 2022-2023

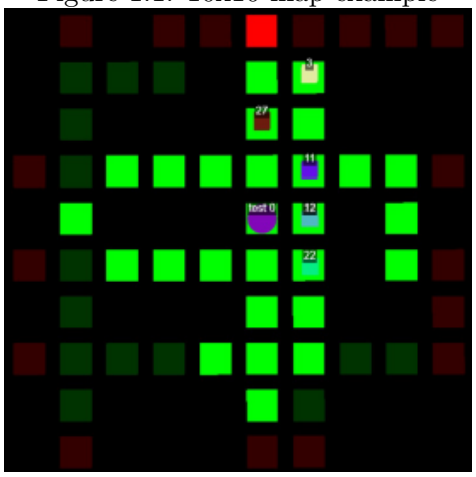
Sabin Andone
232098
sabin.andone@studenti.unitn.it

09/06/2023

## Introduction and Related Work

For the purpose of this project, it has been requested to develop autonomous software agents capable of moving inside the game environment of Deliveroo, accumulating points through package collection and delivery within the delivery area. The agents must implement a BDI architecture, which involves perceiving the environment, managing beliefs, deliberating intentions, selecting plans from a library by using an external planning component and executing the plan. Moreover, it is also necessary to implement a game strategy for two agents that cooperate with each other by exchanging information about their mental states and plan their intentions accordingly in order to maximize the overall score. The code source related to this project work can be found in this link. The structure of the repository is self-explanatory: the source code itself is found inside the src folder, in which it is possible to find the three folders for the Agent A and for the team of two agents (Agent B and Agent C) together with a config.js file used to configure the tokens and ids of the two agents (or for the first one, in case of the single agent). There are two folders for Agent B and Agent C because on of them contains the optimized version of the strategy, which is an extension of the original strategy adopted that adds complexity in order to solve a certain issue which will be discussed later on this report. Inside each sub-folder it is possible to find the .js files used for the implementation, the .pddl file that defines the domain and a test folder containing several logger files that have been used for testing purposes.

Figure 1.1: 10x10 map example



## Agent A: Design Choice and Implementation

The first agent that has been realized for this project is the agent A, which has to satisfy the following requirements:

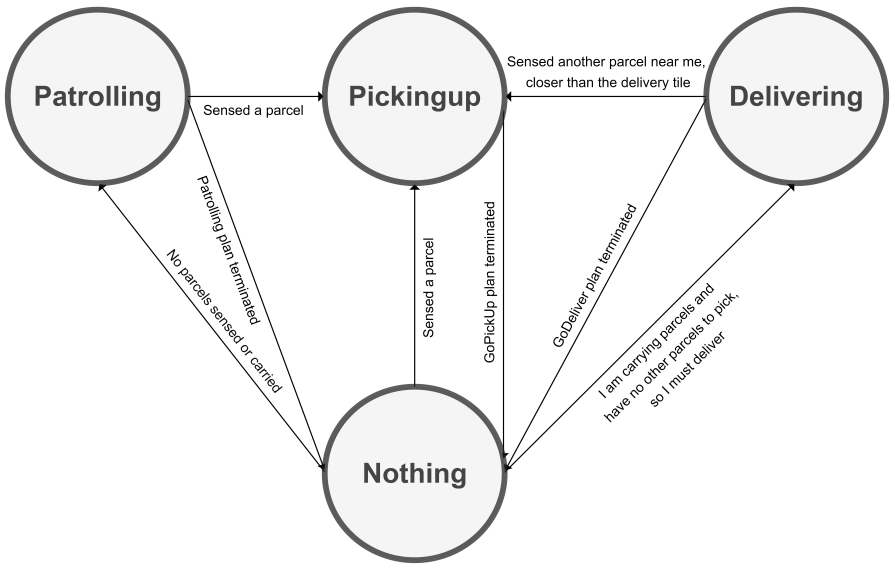
- Sense the environment
- Revise Beliefs
- Revise Intentions
- Use a PDDL-based planner to elaborate plans

The first step for this project has been taking the Benchmark agent from the DeliverooAgent.js repository provided during the course and such agent example has been used as the starting point for this project. This agent comes with an Intention class and a Plan class used for realizing the intention and planning plan. In order to realize an agent that can satisfy all its requirements, I came up with the idea of assigning the agent a state based on what it is doing in the game environment in that moment, thus four states have been realized for this project work, one for each plan plus another state used as an initial state and when a plan is completed. These states can be described as follows:

- The 'nothing' state is the initial state, and is assumed by default by the agent when we execute the .js file. The 'nothing' state indicates that the agent is doing nothing and it must get some operations to do. In this case, when the agent is in state 'nothing' it checks if there are any parcels near it and if that is the case, then the agent enters the 'pickingup' state, otherwise it may enter in either 'patrolling' or 'delivering' state, based if it is carrying any parcels or not. It is possible for the agent to go back to this state when he successfully finishes any plan or when the plan fails because it cannot be executed.
- The 'patrolling' state is the state where the agent does not sense any parcels, so it goes to a random tile where parcels can spawn, hoping that it is possible to find a parcel in the vicinity in that moment. If it reaches the tile without finding any parcel, then it continues to patroll by choosing another tile where to go. If, instead, the agent senses any parcels while it is patrolling, then it computes if it is worth or not to go pick up that parcel, based on its distance and reward, and if it is worth to pick it, then the agent stops the patrolling plan and enters the 'pickingup' state.
- The 'pickingup' state is the state where the agent has sensed one or more parcels and now it goes to pick them up. During this state the agent might find other parcels in its way. When this happens, the agent computes again if it is worth picking the sensed parcel, and always based on its distance, the agent either postpone its current plan in order to take the new parcel or puts it in a "waiting queue" and picks it up later. Once the agent picks up a parcel it goes to pick up other parcels that sensed meanwhile, or if there are no other parcels to pick up, it goes to deliver them, entering in this way in the 'delivering' state.
- The 'delivering' state is the state in which the agent finally goes to deliver all the parcels that it picked up. Thanks to the external pddl planner, the agent always goes to the nearest delivery tile from its position and once it reaches that tile the agent puts down the parcels and in this way it gains points. If there are any parcels that spawn while the agent is delivering and these are closer than the delivery tile, the agent might decide to pick up them, always based on its reward and distance. If the agent decides to postpone

the delivery, then its state transits from state 'delivering' to state 'pickingup', so it can start to go pick up the new parcel.

Figure 2.1: A diagram that explains graphically the four states assumes by the agent




Talking regards the beliefset revision and the environment sensing, the agent interacts with the game environment with the help of DeliverooApi which provides different functions used for sensing the environemnt. The Api establishes a connection with the environment and receives updates about the various entities such as the agent itself, the other agents around, the map and the parcels. The agent's beliefset subscribes to events like onYou, onMap, onTile, onAgentsSensing and onParcelSensing, all of which provide information about the agent's state, the map layout, the perceived agents and parcels. By capturing these events its possible to revise beliefs. The agent maintains its belief set represented by the "me" object. It includes properties such as agent's state, carrying status, current position, id, name and score. The position and the score of the agent are updated by the onYou event. The state of the agent is updated based on the events inside the game environment. The onTile event provided information about the map of the game environment. The onParcelsSensing and onAgentsSensing events updates information about respectively parcels and other agents.

Regarding the intention revision requirement, such requirement is fulfilled by the IntentionRevision class that handles the revision of the agent's intentions. In this case an instance of the IntentionRevision class is being created and initiates its loop function. This function continuously monitors the agent's beliefs and available options for actions, and allows the agent to adapt its intentions based on the changes in the environment. In particular, the (go_pickup) options for intention revision are generated when a new parcel is sensed and then it is decided through a filtering component if a parcel is picked or not and if yes then decide if the agent should pick up it immediately or not. This filtering option component can be splitted in different parts:

- isWorthPickup function, which computes if picking up a certain parcel could increase or not the agents score. The decision is made based on the parcel's reward and its distance and the following formula is used:

$$\text{Resulting Score} = \text{R} - \frac{(1 + \#\text{parcels} + \text{X}) \times (\text{MD} + \text{L})}{\text{PDI}} \times (\text{Distance agent-parcel} + \text{Distance parcel-delivery tile})$$

where R is the reward provided by the parcel, #parcels is the number of parcels the agent is carrying, X can be either 0 (if the state is not 'pickingup') or the size of the parcelsToPick queue plus one (if the state is 'pickingup'), MD is the movement duration, L is the latency, which for simplicity has been set to 500,

PDI is the parcel decading interval. It is necessary to differentiate the formula based on its state because if the agent is not in 'pickingup' state it is assumed that the parcelsToPick queue is empty so it can be excluded from the formula.

- parcelsToPick, which is a variable created for the IntentionRevision instance that acts as a queue for options that have to wait to be executed before the current intention is completed, thus ensuring that the agent can keep track and pick parcels even while executing a plan.
- parcelsToPick sorting function: when the agent needs to pick the next parcel, if there are many parcels inside the parcelsToPick queue, it may be clever to go to the closest parcel. In this case, before picking the next option to execute, parcelsToPick is sorted based on the distance of the parcels indicated inside the option.

## PDDL planner: Domain and Problem file

The domain file "domain-deliveroo.pddl" defines the context and the basic rules for the deliveroo problem. In this file are specified the available predicates and actions inside the domain, allowing the planning to determine a sequence of actions to achieve the desired goals. The requirement of the domain file is ":strips" which means that the domain makes use of a planning based on simple effect operations. Next, we have the predicates, which represent a property or a condition that can be true or not inside the domain. In this case the predicates are:

- (me ?me): it represents the agent inside the domain.
- (tile ?t): it represents a tile in the map.
- (at ?me ?t): it indicates that the agent (indicated with "?me" parameter) is located at "?t" tile.
- (right ?t1 ?t2), (left ?t1 ?t2), (up ?t1 ?t2) and (down ?t1 ?t2): it indicates if it possible to go in one direction from tile t1 to tile t2. In other words, it indicates if the tile t2 is adjacent to t1 and is reachable by going into a certain direction from t1 to t2.
- (blocked ?t): the tile ?t is blocked because there is another agent occupying that tile.
- (parcelSpawner ?t): the tile ?t is a tile in which is known parcels spawn.
- (arrived): it indicates that the agent has reached its destination. This predicate is used for the patrolling action.
- (parcelTile ?t): the tile ?t has the parcel that has to be picked by the agent.
- (carryingParcel): it indicates the state that the agent is carrying a parcel, used for pickup action.
- (delivery ?t): the tile ?t is a delivery tile, thus it is possible to deliver parcels into that tile.
- (deliveryMade): it indicates the state that we have done successfully the deliver of the parcel. It is used for the putdown action.

After the predicate definition, we also have the actions, defined as following:

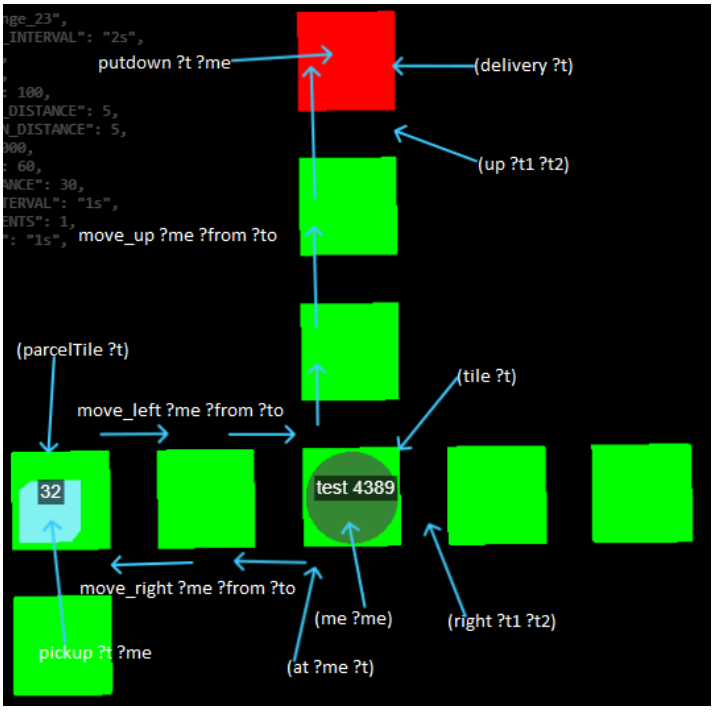
- move_right: Represents the action of moving the agent "?me" to the right from tile "?from" to tile "?to". The action requires that the element is initially in the starting tile "?from" and the destination tile "?to" is adjacent to "?from" on the right. Moreover, the destination tile must not be blocked. The effect of the action is that the element "?me" moves to the destination tile "?to" and is no longer in the starting tile "?from". The same can be said about the other three directions.

- patrollingDestination: Represents the action of patrolling by the agent "?me" towards tile "?t" as a parcel spawner point. This action is used more specifically to check if the agent has arrived at the choosen destination when in the patrolling state. The action requires that the agent "?me" is initially in tile "?t" and the tile is a parcel spawner point. The effect of the action is that the element "?me" reaches the destination tile "?t" and sets the (arrived) predicate.

- pickup: Represents the action of picking up a parcel from tile "?t" by the element "?me". This action is used when the agent reaches the parcel and requires that the element "?me" is initially in tile "?t" that contains the parcel. The effect of the action is that the agent starts carrying the parcel and sets the (carryingParcel) predicate to indicate that now it is carrying a parcel.

- putdown: Represents the action of putting down a parcel in tile "?t" by the agent "?me". This action is used when the agent reaches the tile where it will putdown the parcel. the action requires that the agent "?me" is initially in tile "?t" and that the tile is a delivery point. The effect of the action is that the element "?me" completes the delivery of the parcel and sets the (deliveryMade) predicate to terminate the deliver plan.

Figure 3.1: A map example that shows how predicates and actions are used and interpreted inside the game environment. In this example, the agent plans to go to pick the parcel and then decides to deliver it to the delivery tile.




Regarding the problem of the domain file, it is generated "on fly" during runtime, this because based on the agent's state and on the plan that we have to do we generate a different problem each time. All of these problems have some elements in common, these being the problem domain (which is always the "deliveroo" domain) and the objects used for all these problems, which are the agent and all the tiles inside the map. For the initial state of every problem, we first consider the agent as "me" and we define it inside the problem as (me me), so we can consider it as the agent that moves through the tiles. Then we define its position with the

(at me t-x-y) predicate, where x and y are the position of the tile where the agent is in that moment. Then we define every tile with the (tile t-x-y) predicate, indicating if they are blocked or not (so we can ignore them for the plan realization) and we define also if that tile is adjacent to another one in a certain direction. Moreover, for the delivery tiles we define them as delivery tiles with the (delivery t-x-y) predicate, useful especially in the GoDeliverPlan, since the PDDL planner computes the shortest path to the nearest delivery path with the objects and initial states of the problem. Now, in order to differentiate the three plans from each other, we have to define the goal and initial state of the agent for that plan as follow:

- Patrolling: in this case we define that initially the agent has not arrived (we use the (not (arrived)) predicate to do that) and also we indicate the tile where parcels can spawn by using the (parcelSpawner t-x-y) predicate. The goal is to reach the state (arrived) and it is possible to do that when we execute the patrolling action from the domain, which will be done only when the agent satisfies the (at me t-x-y) where t-x-y is the tile that is indicated as the parcel spawner tile.
- GoPickUp: in this case we define that the agent is not carrying the parcel we want to pick up by using the (not (carryingParcel)) predicate and we also indicate in which tile the parcel is located, by using the (parcelTile t-x-y) predicate. The goal in this case is to reach the state (carryingParcel) which is possible when we execute the pickup action which can be done only when the agent is at the tile where the parcel is also located.
- GoDeliver: for this plan we only add the (not (deliveryMade)) predicate since we need to deliver the parcels and any delivery tile can be used to do the putdown action. The goal is to enter in the deliveryMade state by reaching a delivery tile. The external PDDL planner calculates the path to the nearest reachable delivery tile.

## Agent A: Experiments and evaluation

In this chapter we will comment some experiments that have been done on Agent A. The experiments done on this agent will regard about the performance of the agent inside the game enviornment using the 4 levels for Challenge 1 and for each one of these levels the experiment has been repeated using different configuration setups in order to prove how the agent behaves based on the configuration values. These values are:

- the maximum number of parcels that can be present in the map (initially set to "infinite" as default)
- the movement duration (initially set to 500)
- the parcel decading interval (initally set to "infinite" which means that parcels won't expire)
- the number of randomly moving agents present inside the map (it is 2 by default)

All of these experiments have been done by starting the agent loop and letting it run for 3 minutes and the number represented for each test is the score of the agent after the test. The other number (the one in the parenthesis) represents the score that the agent would have if it could deliver in time the parcels it was carrying.

| Levels Challenge 1 | Default values | Movement duration = 100 | Maximum number of parcels = 20 | Parcels decading interval = 1s | Number of agents = 10 |
|---|---|---|---|---|---|
| Level 1 | 26 (904) | 784 (1427) | 725 | 190 | 454 (792) |
| Level 2 | 328 (873) | 817 (1269) | 623 | 192 | 377 |
| Level 3 | 479 | 1239 (1550) | 717 | 345 | 507 (678) |
| Level 4 | 0 (900) | 811 | 501 | 46 | 34 (210) |

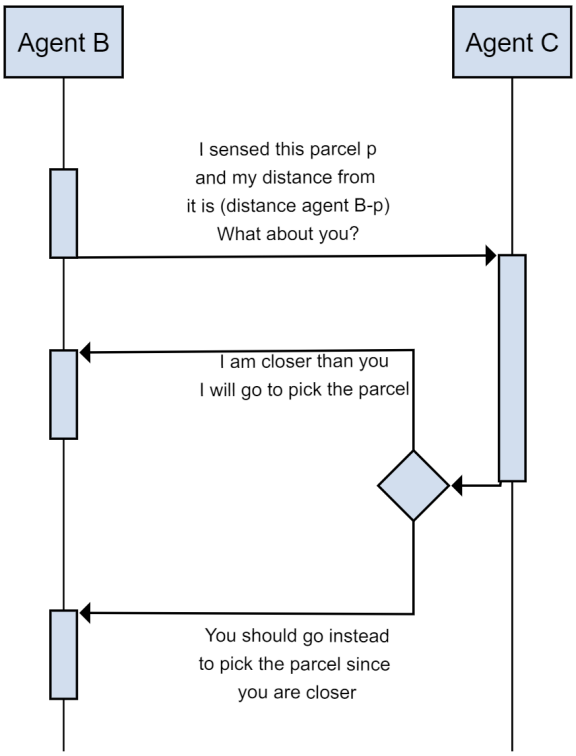
After the first experiments using the default values, it has been immediately observed an issued with our agent A: it is too slow and since it will continue to pick up parcels as long as they will continue to spawn near it (due to its "greedy" approach), the agent will go to pick them all, which means that potentially the delivery would never happen since the parcels spawn too fast in respect to the agent's speed movement. This also happens because the parcels by default don't decay so they would never expire. So it has been decided to repeat the experiments using all default values expect for the movement duration, which has been set equal to 100, to see if an increase of the agent's speed could solve this issue. Obviously, a speed increase means more parcels delivered and thus a higher score, but after doing such experiment, it has been observed that the problem is actually the fact that parcels can spawn infinitely with the "infinite" value for the maximum number of parcels that can be present in the map. So it has been decided for the rest of all the experiments to set this number to 20 parcels, so the agent can eventually go to deliver. The movement speed will go back to 500 for the rest of this part of validation. Now the problem should be solved, since the agent A is now able to deliver the parcels when this limit is set. We also repeat the same experiment but making the parcels expiring, in order to see how the agent behaves in this case. Obviously, the score will be lower since the parcels will decay overtime. In this case the agent behaves as it was programed, i.e. when it is carrying a lot of parcels it stops to pick up other parcels and starts to go to deliver the ones it is already carrying in order to get the points and go to pick the next parcels. One last experiment that has been done on Agent A is to see how it would behave when there are a lot of agents playing in the same map. By default there are 2 agents that randomly move inside the game, for this last part of the experiment we set this number to 10. The parcels in this part of the experiment do not expire. It is expected that the agent A will be blocked more often and even being stuck at some point, thus resulting in less points gained. Instead, for the first two levels the agent behaves as usual since the map is so big that there is enough space to move in the map even with a lot of agents moving around. Surprisingly, even in the third level the agent managed to deliver the parcels despite being blocked a lot by the other agents. In the last level instead, the agent struggled to deliver the parcels, since there are less tiles where parcels spawn.

## Agent B and Agent C: Game Strategy

In this chapter we will discuss about the second part of the work done for this project. For the agent B and agent C, in fact, it has been requested to realize a strategy that let these two agents cooperate with each other in order to maximize the overall score of the team. These two agents are very similar to the agent A discussed previously, but in addition, the two agents communicate with each other in order to coordinate their actions. There is no difference between these two agents apart of their token and id. Since the implementation and the pddl part of these two agents are very similar to the agent A's one, this section will have more focus on the strategy adopted for the two agents.

The strategy adopted for the two agents can be explained with the following example: when one (we call it agent B) of the two agents senses a new parcel, it computes its distance from that parcel and then asks to the other agent (agent C) if it is closer or not. To do so, it passes all the details about that parcel, like its coordinates, id and reward, and also the distance between the parcel and the agent B. Agent C receives the message and it also computes its distance from the parcel. Then it does a comparison between the two distance, in order to determine who should go to pick the parcel among the two agents. If the agent C is closer and can find a path to reach the parcel, then it will be the agent that will pick up the parcel, otherwise the task will be done by agent B and agent C will just send an acknowledgement answer to notify agent B that it should go instead to pick the parcel. Obviously, the roles can be swapped if agent C is the one that senses a parcel.

Figure 5.1: A diagram that shows graphically how the communication takes place between the two agents.




This strategy is simple and effective in almost all cases, except in one case: when the two agents block each other. The experiments done on this part of the project will make evident this issue and in order to solve this problem an "optimized" version of this part of the project has been created. This optimized version of the strategy is similar to the previous one, except for the fact that when the agent that is delivering the parcels is being stuck by its teammate, they communicate in order to unlock this situation by exchanging the parcel they have. More precisely, the agent that has to deliver but is blocked by its teammate puts down its parcel and leaves the tile empty so the teammate will go to pick these parcels. This plan can happen in two ways, based on where the agent is blocked: if it cannot move nor apply a plan because stuck by the teammate, then the agent will ask its teammate to move away so it can have then enough space to move and leave the parcel (if it does not do this, then the agent cannot leave its parcels because they will be unreachable); if it is blocked by the teammate while the agent was moving, then the agent will putdown its parcels and move away so the teammate can take care of the parcels. In this way the strategy can still be applied and also solve this issue, which is especially notable in one of the levels used for testing the team.

## Agent B and Agent C: Experiments and evaluation

The experiments that have been done on the team of two agents are basically the same type of experiment done on Agent A but in this case the tests have been done on the three levels of Challenge 2 and the score reported on the table will be the overall score of the team. For the experiments done for the team of agent B and agent C we consider the following values:

- the maximum number of parcels that can be present in the map set to 8
- the movement duration set to 100
- the parcel decading interval (initally set to "infinite" which means that parcels won't expire and then repeating the experiment by putting a decading interval of 1 second)
- the number of randomly moving agents present inside the map set to 2 (only for the first level, 0 for the other 2 levels)

| Levels Challenge 2 | Without decading interval | With parcels decading interval = 1s | Without decading interval (optimized version) | With parcels decading interval = 1s (optimized version) |
|---|---|---|---|---|
| Level 1 | 448 | 169 | 668 | 200 |
| Level 2 | 0 | 0 | 663 | 214 |
| Level 3 | 224 | 127 | 336 | 67 |

The experiments done on the not optimized version of the team immediately indicates one issue: the agents do not cooperate enough to maximise their score. This because they just limit to exchange data about their beliefset and distances in order to manage their intention, but they don't have a way to share the parcels they are carrying. Due to this reason plus for the fact that the agents can get stucked by each other (as discussed in the previous section), the game strategy cannot be applied to the level 2 since it requires that one agent brings the parcel to the other one so the latter can deliver the parcel to the delivery tile. This problem also occurs when an agent goes to take a parcel but for some reason the other agent was patrolling also in that direction, thus blocking the way to the agent intended to pick the parcel.

To solve this issue, the optimized version of the strategy has been created. The experiments done on this version immediately show the fact that now the agents are able to deliver parcels when playing in the second level of the challenge, thus solving the issue and reaching their goal of taking the parcels. Regarding the other levels instead they behave as in the non optimized version, except for the fact that now when the agents get stuck they exchange their parcels and then continue their execution.

## Final considerations

The inital goal of developing autonomous agents capable of moving inside the game environment in respect to the requirements defined previously has been achieved. The agents are able to align their behaviour with the BDI architecture and they are also able to make use of the external PDDL planner to realize their plans. Also, talking more specifically about the team of two agents, the strategies applied to them, even tough they are not the most optimal ones, are still effective enough for the team to gain points and achieving their goals in respect with the requirements.

The agents analysed in this report may be subject to further improvements by adding complexity and also other (optional) features that may make the agents more interesting but also more complex, but due to the deadline for this project, it has to be submitted as it is described in the report. Moreover, it has been decided that it is better to leave the agents a bit simplier yet effective enough to achieve their goals without adding too many (non requested) features that may not bring much improvement to the project.