Drone Delivery Operations: A Reinforcement Learning Approach

Daniele Commodaro
daniele.commodaro@studio.unibo.it

September 2024

The project proposes to develop a simulation of drone behavior using reinforcement learning techniques. The main objective is to train a fleet of drones to perform deliveries in a dynamic environment, facing challenges such as obstacles, bad weather areas, battery management, and optimized routes.

In the simulation, the drones must pick up packages from a warehouse and deliver them to predefined destinations, while managing their own battery, they avoid obstacles and stuck situations. Reinforcement learning is used to progressively improve the drones' decision-making, based on past experiences, in order to optimize drone decision making.

The drones' behavior is modeled through the use of a q-table, which allows them to learn the best actions to take based on the state of the environment. The simulation environment features obstacles, charging stations, and areas subject to adverse weather conditions, requiring the drones to plan alternative routes to complete deliveries safely.

This project aims to explore how reinforcement learning techniques can be used to solve autonomous decision-making problems, which could potentially also be applied in real-world scenarios.

Contents

1 Objectives and expected outcomes

The project has several objectives, with the aim of developing a simulation system that realistically models the behavior of autonomous drones engaged in delivery operations. The main objectives are listed below:

- Environment development: create a simulated environment that includes all the entities necessary for realistic delivery management: drones, charging stations, warehouses, packages to be delivered, and adverse weather conditions. Each entity will have specific characteristics and will interact with the others dynamically. In particular, the system will have to manage the spatial arrangement of charging stations, warehouse and delivery points, together with the creation and management of weather zones that impact drone navigation.

- Training System: implement a reinforcement learning training system for drones that allows them to learn the best actions to take based on the current state of the environment. This will include movement towards targets, obstacle avoidance and route optimization to avoid congestion situations and to circumvent areas affected by adverse weather conditions that can hinder the drones' journey. The training process must allow drones to learn to generalize, taking advantage of the knowledge acquired in different contexts. This will allow drones to adapt effectively even in situations where the composition of the environment changes, making the system robust and versatile.

- Realism and efficient navigation: the goal is to achieve a realistic navigation system, in which drones can reach their destinations efficiently. This includes implementing a good obstacle avoidance mechanism that is simple but effective. Through the learned navigation system the drones will be able to detect static and dynamic obstacles, such as other drones or areas of bad weather, and calculate alternative routes to avoid congestion and stall situations, making sure to avoid stuck situations.

- Performance verification and results evaluation: the final objective is to create a simulation that allows to verify the performance of the system and the results of the drone training. It will be essential to analyze the efficiency of the system. The results observed through the simulation will be used to evaluate the effectiveness of the RL algorithm and to identify potential areas for improvement.

- Implement a user interface: develop a graphical interface for real-time viewing of the simulation. The interface will allow to monitor the location of the drones, the status of deliveries, the availability of charging stations, the battery level of the drones, the number of deliveries made, and the presence of bad weather areas. The GUI will be a fundamental tool for observing the progress of the simulation and evaluating the behavior of the drones during the learning process.

The expected results of the project are to obtain a working simulation, in which the drones behave as realistically as possible during their delivery operations. In addition to this, demonstrate that the choices made by the drones are correct and effective, completing all the expected deliveries without generating unwanted situations, such as drones running out of battery, collisions or other types of problems. Another key achievement is to develop an intuitive user interface, which allows to easily monitor the simulation and effectively manage the state of the environment and drones.

2 Project requirements

- Environment:
  - State space: the system must manage an environment represented by a grid, where drones move to perform deliveries. Each grid cell can contain an element which can be a drone, a delivery point, an obstacle, a charging station or an area with bad weather.
  - Stochastic behavior: the environment must introduce variability through random events, such as bad weather, which affect the drones' movement.
  - Battery management: drones must manage their battery levels, stopping at charging stations when necessary to avoid running out of power.
- Reinforcement learning algorithm:
  - Exploration-exploitation policy: the system must implement an exploration strategy using an epsilon-greedy approach to balance between exploring new actions and exploiting acquired knowledge.
  - Q-learning approach: the main algorithm must be based on q-learning, updating a q-table for each drone based on the actions, rewards, and subsequent states observed during the simulation.
  - Generalization capacity: training must enable the drones to generalize the acquired knowledge to different contexts, allowing them to adapt to new environment configurations.
- Rewards and penalties:
  - Reward system: during training, each action taken by the drones yields a reward or penalty based on its effectiveness. correct movements are rewarded, while errors or unnecessary movements should be penalized.
  - Reward adjustment: the system must be able to adjust the value of rewards to determine when to favor the choice of certain types of actions over others.
- Actions management:
  - Available actions: drones must be able to choose from various actions, such as moving in different directions, avoiding obstacles and circumnavigating areas where there are bad weather conditions or obstacles.
  - Action validation: the system must prevent invalid actions, such as moving outside the grid or colliding with obstacles, and penalize the choice of incorrect actions.
- Obstacle management:
  - Obstacle detection and avoidance: drones must be able to detect nearby obstacles and adopt strategies to avoid them, including calculating alternative routes when necessary.
  - Circumnavigation: drones must be able to efficiently navigate around obstacles and areas of bad weather, calculating optimal routes.
- Battery management:
  - Charging management: drones must be able to recharge to avoid running out of battery, accessing charging stations when necessary.
  - Charging time: charging time must be based on the drone's remaining battery level and the time needed to complete the recharge.

  - Battery consumption: drones consume a predetermined amount of battery with each movement. Navigating in areas affected by bad weather results in increased battery consumption.

The non-functional requirements of the project specify that the overall system must efficiently handle the planned operations, allowing adequate monitoring of the simulation progress and rewards obtained during the training process. The graphical interfaces must be simple and intuitive, enabling users to monitor the system easily. Additionally, the system must be robust and reliable, aiming to avoid and prevent any error situations and ensuring continuous package distribution without interruptions. The code must be well-structured to facilitate future maintenance and updates.

3 Simulation overview and interface

3.1 Simulation dynamics

The simulation begins with a certain number of objects in the warehouse. Each time a drone picks up an object, a delivery point is generated in the environment, which becomes the specific target of that drone, and it must reach it to complete the delivery. The delivery points created by the drones are considered obstacles by other drones, which avoid them during their movement.

Each drone has an assigned charging station, towards which it heads when its battery level drops below a certain threshold. Drones consider the charging stations of other drones as obstacles and, as long as their battery level is not low, they also avoid their own station. When a drone goes to recharge, it remains stationary at the station for a time proportional to its remaining battery level.

The simulation ends when all deliveries have been completed, and the number of objects in the warehouse reaches zero. At that point, the drones return to rest at their charging stations.

The environment also includes the creation of bad weather areas, generated randomly with varying sizes. These zones persist for a certain period of time before being removed and affect the movement of the drones within the simulation. Drones attempt to avoid the bad weather areas, but when they cannot, their battery consumption increases by one unit for each cell they traverse that is affected by bad weather conditions.

3.2 Explanation of graphical interfaces

3.2.1 Rewards chart

At the end of the training process, a graph is generated to visualize the reward trends. This graph shows the average cumulative reward for blocks of 100 simulation episodes. The x-axis represents the number of episodes (aggregated in blocks of 100), while the y-axis shows the average cumulative reward. This tool is used to assess the performance of the drones and monitor the effectiveness of the training over time.

3.2.2 Simulation GUI

The simulation GUI is designed to provide a clear and intuitive visualization of the current state of the environment and the drones. The main elements are divided into:

- Grid: the main visualization is a grid representing the simulation environment. The key elements are:
  - Drones: each drone has a label with a unique identifier composed of a letter "D" and an identification number. Drones are normally colored in light blue. While charging, they are colored based on their remaining battery level. The coloring ranges from red (very low battery) to light green (fully charged), with intermediate shades representing different charge levels.

  - Delivery Points: represented by cells with a specific yellow color on the grid and a label with the letters "DP" and a number identifying the drone assigned to that delivery point.
  - Warehouse: indicated with a brown color and the label "Warehouse" along with the number of objects to be delivered. This represents the pickup point for items to be delivered. When a drone reaches the warehouse to pick up a new item, the warehouse label changes to "Load" to indicate that a drone is picking up an item, and the warehouse cell is occupied.
  - Charging stations: highlighted distinctly with a dedicated green color and a label consisting of the letter "R" and the number of the drone assigned to that charging station. When a drone is recharging, the label changes to "Recharge" and the cell is colored to represent the current battery level of the drone.
  - Bad weather areas: represented by blue rectangles with transparency. These rectangles are randomly generated and can vary in size and position to simulate varying atmospheric effects during the simulation.

- Interface text: outside the grid of the environment, the GUI displays information such as the battery percentage of each drone, the number of deliveries made by each drone, and the total number of deliveries completed.

4 System architecture and design

In the project, the reinforcement learning problem is defined in terms of state, action, and reward. The environment represents a dynamic scenario in which the drones operate to complete deliveries, while the RL system trains the drones to navigate this environment effectively using an exploration and exploitation approach.

4.1 System class diagram

The following class diagram illustrates the structure of the classes that make up the project, with the aim of providing a quick and visual understanding of the external methods that each class invokes. To make the diagram more readable and reduce visual complexity, it was decided not to represent the internal attributes of the classes. Including these details would have significantly increased the size of the diagram, making it difficult to view clearly within this report.

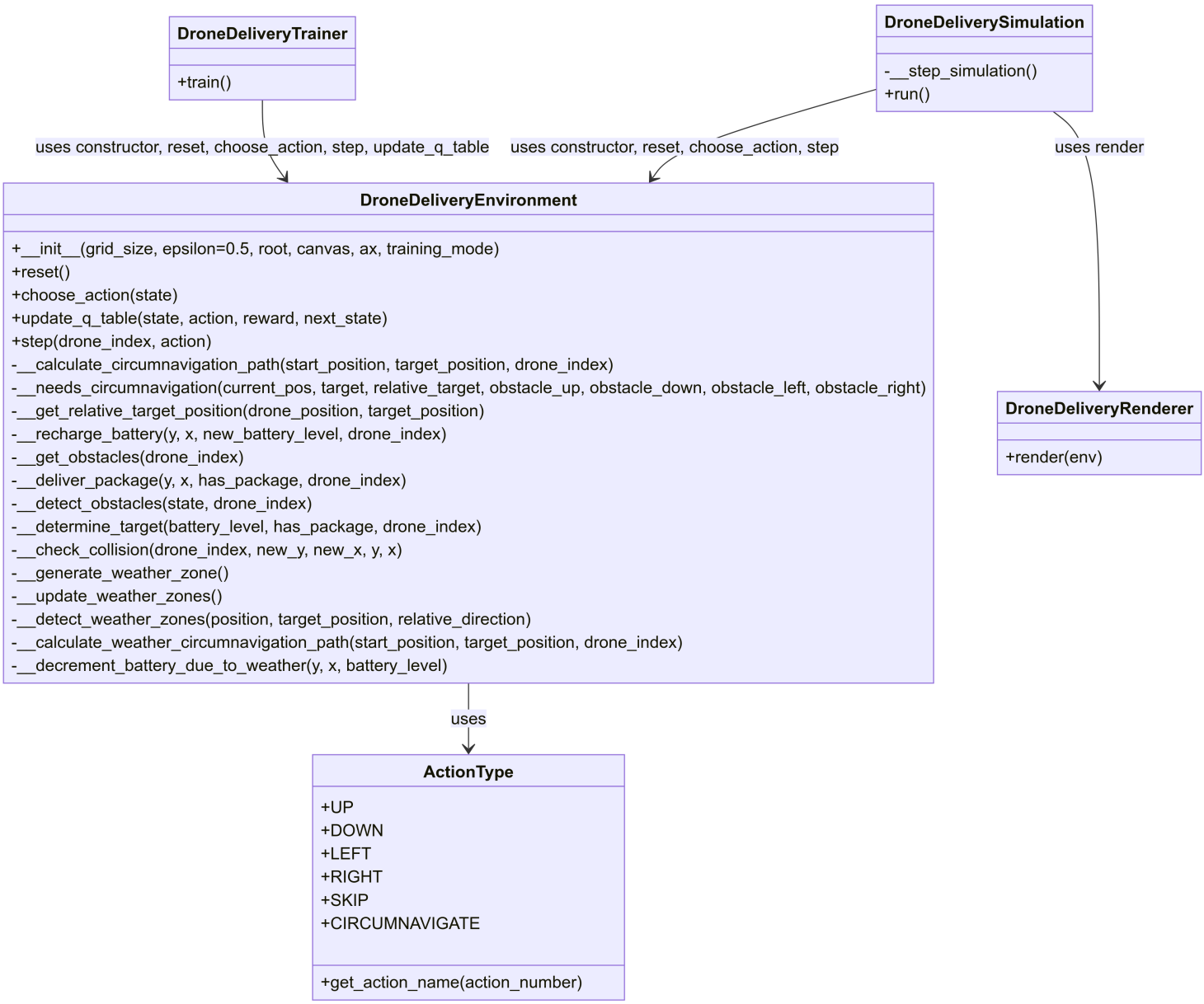

Figure 1: System classes

The main class, DroneDeliveryEnvironment, contains various methods to manage the drones' behavior. The classes DroneDeliverySimulation and DroneDeliveryTrainer utilize this class to

run the simulation and training process. The simulation also involves the DroneDeliveryRenderer class to visualize the environment, while the ActionType class is an enumeration that defines the possible actions the drones can take.

4.2 Simulation architecture

The diagram provides an overview of the simulation system architecture. The DroneDeliverySimulation class manages the entire simulation, creating the environment and handling the drones, and starting the simulation process.

The simulation creates the environment and initializes it. The simulation also handles loading the q-table, which is crucial for the drones' action selection. The q-table is read from a file generated during the training process, and the information is loaded into the environment through a file named "q_table.npy". This step allows the drones to make decisions based on the knowledge acquired during training.

The drones operate within the environment, each with its own state. The drones use the environment's choose_action method to select actions based on the q-table values. After executing actions through the environment's step method, the drones update the environment with the new states.

An important aspect of the simulation is that the epsilon parameter is set to zero during execution. This disables exploration, ensuring that the drones follow only the knowledge learned from the q-table without introducing new random explorations. Finally, the render method is used to visually update the environment, showing the progress of the simulation following the execution of the step method by each drone.

The simulation ends when the drones successfully complete all the deliveries, with the drones returning to their charging positions to rest.

4.3 State

The system uses two types of states:

1. State returned by the environment: at each step, the environment returns a specific state for each drone. This state contains detailed information about the drone's current situation, such as its position, battery level, presence of obstacles in adjacent cells, the package being delivered, recharge time, and circumnavigation path. This state is rich in detail and is continuously updated to reflect the drone's real-time situation within the environment.

2. State stored inside the q-table: the state saved in the q-table is a simplified version of the drone's state. It contains only the essential information for learning, reducing the number of values to consider facilitating and making the training process more effective at the same time, and also promoting faster convergence. The state stored in the q-table includes:

    - Presence or absence of obstacles in adjacent positions.
    - Relative position of the target compared to the drone.
    - Need to circumnavigate an obstacle or a bad weather zone.
    - Available actions for the drone.

The decision to store only a subset of the information from the environment helps simplify the training process, reducing computational complexity and improving learning efficiency.

4.4 Actions

The actions that drones can learn include movement in the four directions (up, down, left, right) and the circumnavigation of obstacles and bad weather zones. Movement is discrete and takes place on a two-dimensional grid, where each cell can be either empty or occupied by an obstacle.

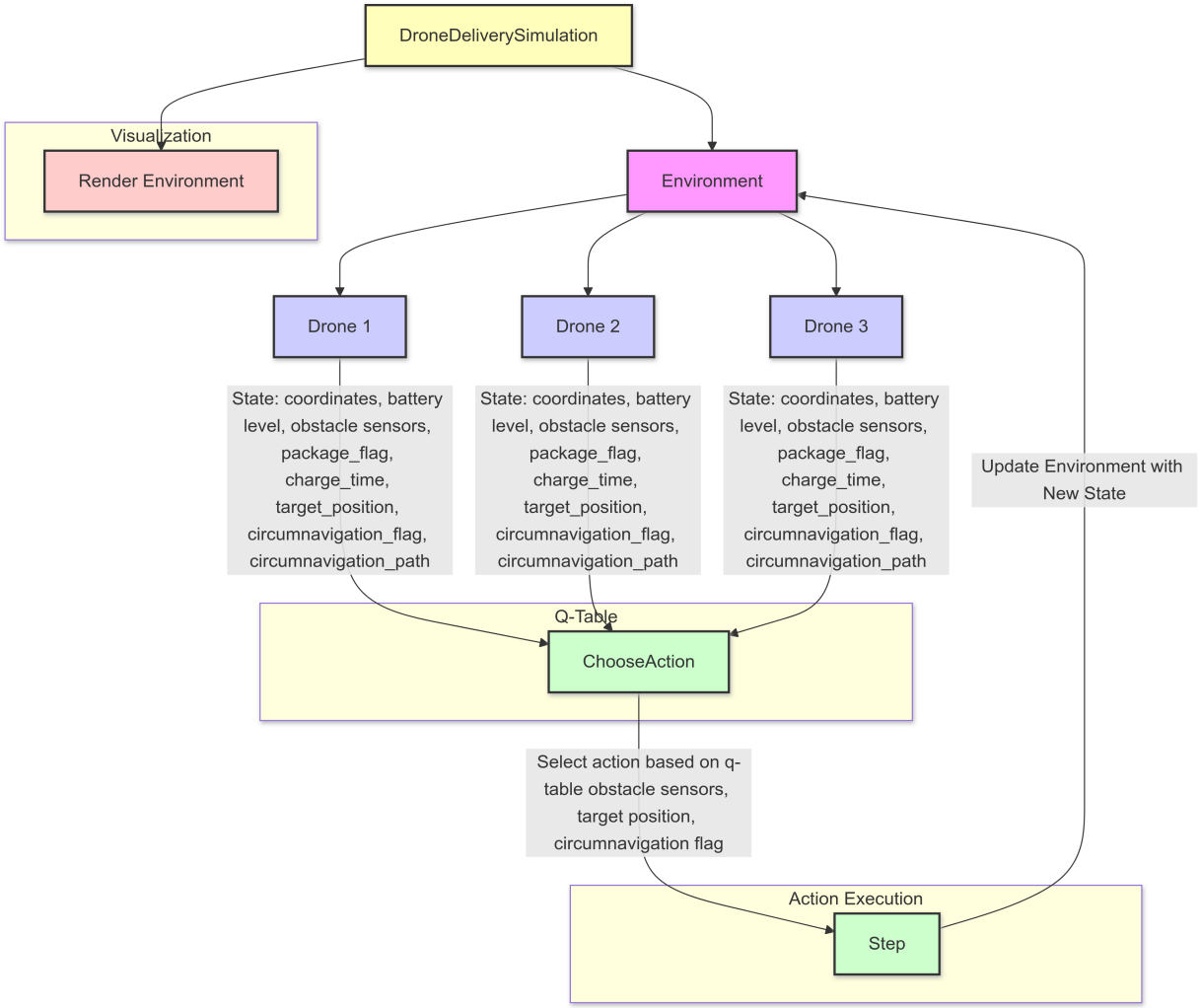

Figure 2: System architecture

At each step, the drone evaluates the possible directions to move, considering the surrounding obstacles and the grid's edges, and decides which action to execute based on the implemented learning policy.

The drone perceives the presence of obstacles in adjacent cells but does not directly perceive the edges of the grid. However, movement outside the boundaries is not allowed, so the environment automatically manages this constraint.

4.5 Rewards and penalties

The reward system plays a crucial role in training. Each action taken by the drone incurs a reward or penalty based on its effectiveness in reaching the goal. The value of the rewards is proportional to the importance of performing one action over another, with higher rewards assigned to actions that bring the drone closer to its target. For example, moving in the direction of the target yields a positive reward, while attempting to move into a cell occupied by an obstacle incurs a penalty.

Penalties are balanced to discourage undesirable behaviors, such as trying to pass through obstacles or moving away from the goal. Penalties are proportional to the severity of the error made, contributing to a balanced and effective training of the model.

4.6 Q-learning algorithm

The q-learning algorithm is a reinforcement learning method used to enable the drones in the simulation to learn the best actions to take in the environment based on their interactions. In this context, each drone learns to choose the best action based on its current state, progressively updating the q-table, which is a table of values mapping states and actions to expected rewards.

The algorithm works as follows:

1. Action selection: based on the current state, the drone selects an action using an epsilon-greedy policy. If the epsilon value is high, the agent explores random actions; otherwise, it chooses the action with the highest q value. During the simulation, epsilon is set to 0, so the drone only exploits what it has learned.

2. Q-table update: after performing an action, the environment returns a reward and a new state. The agent uses this information to update the q-table, gradually improving its future decisions. When updating the q-table, consideration is given not only to the immediate value of the action just taken but also to how that action leads to a state where a better future reward can be obtained. In other words, the aim is to optimize not only the immediate reward but also the potential future gain. The algorithm is explained in more detail in the detailed design section.

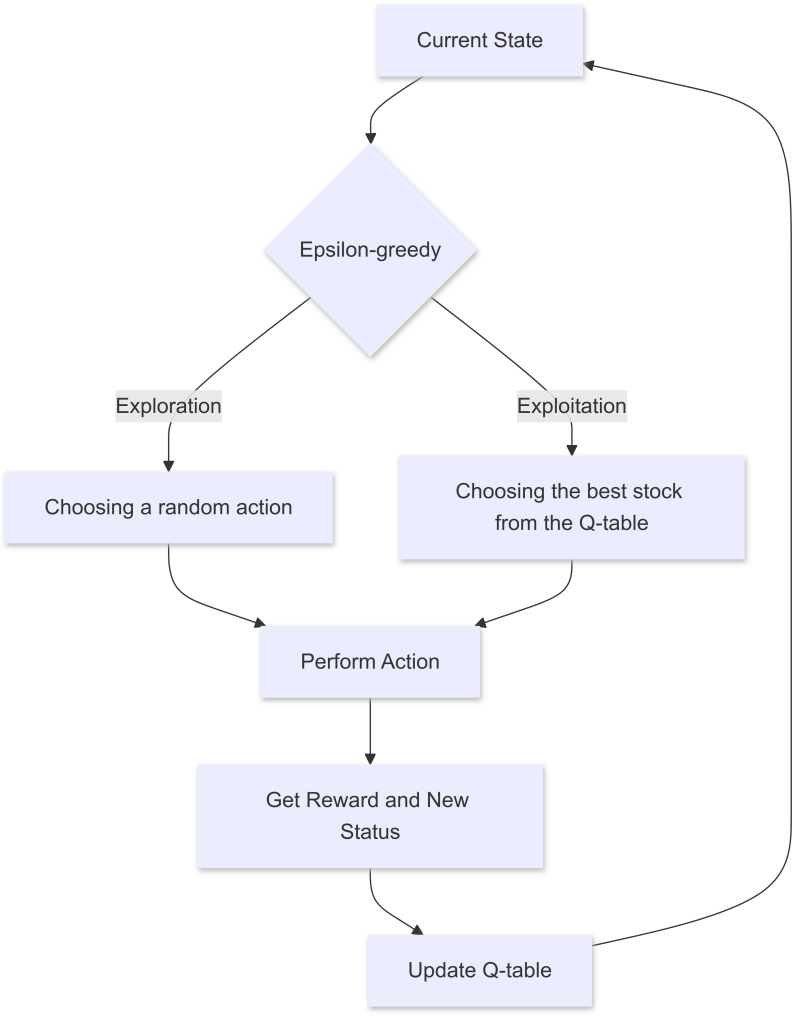

Figure 3: Q-learning algorithm

For this project, a single q-table was chosen to keep the model simple, given the limited number of actions to learn. However, if the problem were more complex, with more actions

or sub-actions, it would have been beneficial to split the training into multiple q-tables. This would allow focusing learning on specific subsets of actions while retaining knowledge gained from previous training without needing to retrain the entire model from scratch each time.

4.7 Training

The training process uses the q-learning algorithm to optimize drones' ability to choose actions during their delivery operations. The design of the training process is as follows:

- Initialization: the training process is initiated by constructing a dedicated object that requires an environment and several configuration parameters, which are: the number of episodes, the learning rate (alpha), the discount factor (gamma), and the initial value of epsilon. The epsilon parameter controls the balance between exploration and exploitation. At the beginning, epsilon is relatively high, encouraging the drone to explore different actions randomly. As training progresses, epsilon gradually decreases, favoring the adoption of actions that have proven to be more effective. The learning rate alpha specifies how much the value estimates of an action should influence the update of existing values. The discount factor, on the other hand, adjusts the importance of future rewards compared to immediate rewards. A value close to 1 means that the agent places great importance on future rewards, while a value close to zero means that the agent gives more weight to immediate rewards.

- Training:
  - Episodes: the training takes place over a predefined number of episodes. Each episode begins with the reset of the environment and the return of the drone's initial state. An episode can end when a drone completes all scheduled deliveries or if it runs out of battery.
  - Action selection: at each step of the episode, the drone selects an action. The action is chosen based on the epsilon-greedy strategy, where epsilon determines the probability of selecting a random action (exploration) versus the action with the highest q-value (exploitation).
  - Q-table update: after performing the action and observing the new state and reward, the q-table is updated. The update uses the q-learning formula to improve the estimates of q-values and guide future decisions.
  - Epsilon decay: at the end of each episode, epsilon is slightly reduced. This progressive decay decreases the amount of exploration over time, encouraging the agent to exploit the knowledge it has gained.

- Training conclusion: at the end of the training process, the q-table, which contains the estimated q-values learned during the training process, is saved in a file named q_table.npy. This file allows the learning to be preserved and used in future simulations or for further improvements.

- Results Visualization: at the end of the training, a graph is generated that displays the average cumulative reward over blocks of 100 episodes. This graph helps monitor progress and assess the effectiveness of the training process.

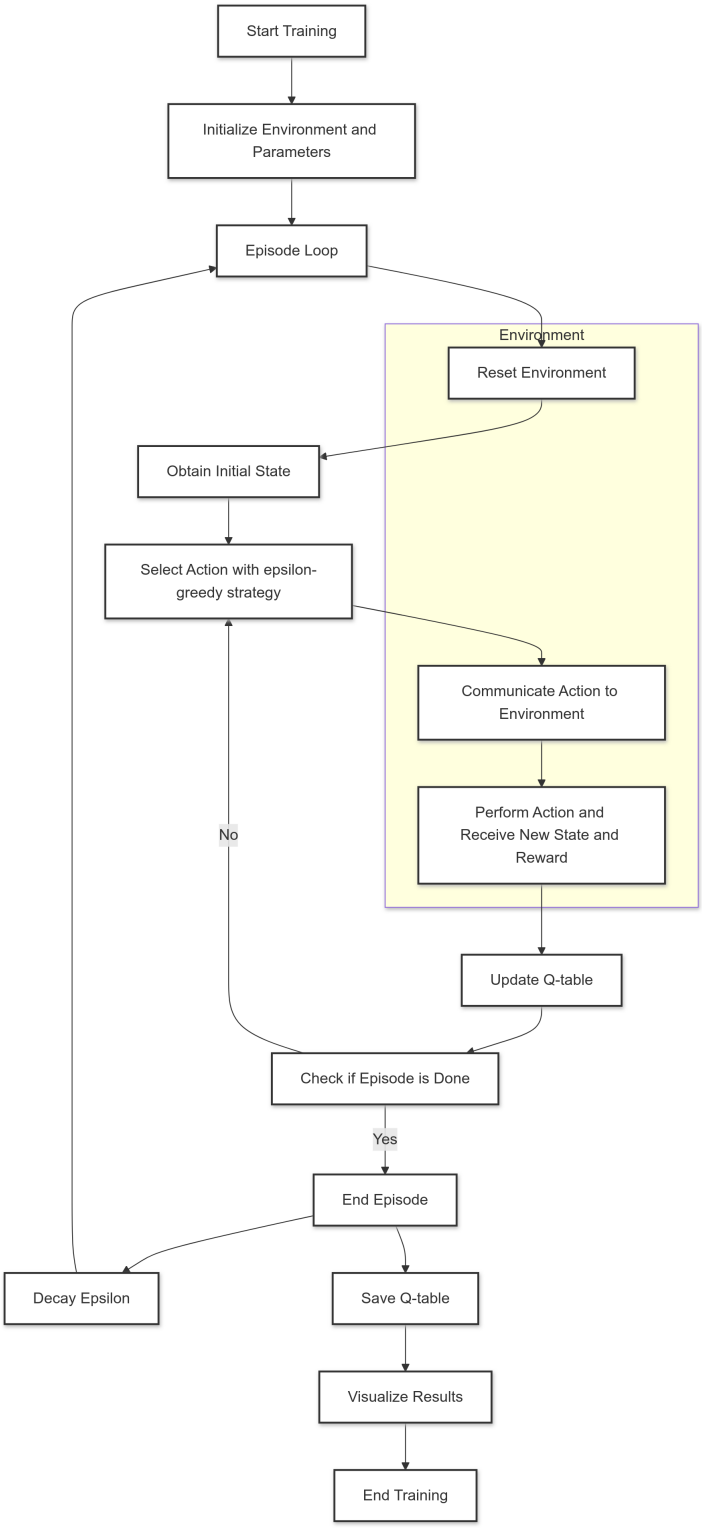

Figure 4: Training process diagram

The training relies on a continuous cycle of exploration and exploitation, with a gradual adjustment of the balance between these two aspects through epsilon decay. The q-table is iteratively updated to reflect the best choices and is eventually saved for future reuse.

5 Detail design

5.1 Q-learning implementation

The implementation of q-learning algorithm consists of two main parts: action selection and q-table updating. Below is a detailed description of these two processes:

1. Action selection: the choose_action method determines which action the drone should perform based on its current state. The selection occurs in two ways:
   - Exploration: with a probability defined by epsilon, the drone will perform a random action to explore new strategies. This happens when a randomly drawn number is lower than epsilon or when all q-values for the possible actions are zero, this happend where there is no information about which action is preferable.
   - Exploitation: if the drone is not exploring, it selects the action with the highest q-value in its current state, meaning the action with the best estimated reward. During the training process, the epsilon value is gradually decreased after each episode. However, during the simulation, the epsilon value is set to zero so that the drone can fully stop exploring and focus solely on making use of the learned knowledge.

```
1       if random.uniform(0, 1) < self.epsilon or np.sum(q_values) == 0:
2           return random.choice(range(self.num_actions))
3       else:
4           return np.argmax(q_values)
```

2. Updating the q-table: the table is a multidimensional matrix that represents the drone's knowledge of which action to take in each state. The update process is as follows:
   - Determining the current state and the new state: the data related to the current state and the next state are extracted and broken down into relevant components, such as the presence of obstacles, the relative position to the target, and the need to navigate around obstacles or unfavorable weather zones.
   - Best action in the new state: the method determines which action would be the best to take in that new state, meaning the action that leads to the maximum q-value.

```
1           best_next_action = np.argmax(self.Q_table[next_obstacle_state[0],
                ...])
```

   - Calculation of the reward target: the future reward target is calculated by adding the immediate reward received for the action just taken and the estimated maximum value of the future reward, multiplied by a discount factor gamma. This represents the prediction of the overall value of the action performed.

```
1           td_target = reward + self.gamma *
                self.Q_table[next_obstacle_state[0], ..., best_next_action]
```

   - Error computation: the temporal difference error is the difference between the calculated target and the current q-value for the current state and action. This error represents how much the initial reward estimate deviates from the actual observed reward.

```
1           td_error = td_target - self.Q_table[obstacle_state[0], ..., action]
```

   - Updating the q-value: finally, the q-value for the action just taken is updated using the calculated error, multiplied by the learning rate alpha. This allows the drone

to improve its strategy over time, refining its decisions based on the results of past actions.

```
1         self.Q_table[obstacle_state[0], ..., action] += self.alpha *
              td_error
```

With this sequence of steps, the drone learns which actions maximize the reward, continuously updating its action policy based on the experience it accumulates.

5.2 Management of drone steps and actions

This section focuses on how the drone performs actions, how these actions affect the drone's state, and how the system handles state updates and rewards. The step method is the core of the interaction between the drone and the environment. In each iteration, a drone performs an action that modifies the environment and its current state, considering various factors such as obstacles, battery level, and whether it is carrying a package.

The flowchart describes the logic for managing a drone's actions during a single step of the simulation. The process begins by checking if the drone is currently charging. If so, the charging timer, which consists of a number of simulation steps, is decremented, and the drone's state is immediately returned with this updated value, skipping all other operations.

If the drone is not charging, the process continues by selecting the action that was passed as an argument to the step method. The next check is whether the drone has detected the need to circumnavigate an obstacle. If no circumnavigation is necessary but the drone has chosen a circumnavigation action, a penalty of -30 is applied.

If circumnavigation is necessary, the flow checks if the drone has chosen the circumnavigation action. If so, the circumnavigation path is updated, the drone moves along that path, and a large reward of +50 is assigned. If the drone did not choose the circumnavigation action when needed, a penalty of -50 is applied.

After evaluating the need to circumnavigate an obstacle, the flow proceeds by checking if there are any obstacles in the direction the drone is attempting to move. If an obstacle is present in the chosen direction, a penalty of -20 is applied. If there is no obstacle, the system checks whether the movement is aligned with the relative position of the target. Correct movement toward the target is rewarded with +10, while incorrect movements are penalized with -5.

Finally, the drone's state is updated, considering factors such as the new calculated position, battery level, delivery or pickup of the package, target position, obstacle detection, and the need for further circumnavigation. Once all updates are made, the method concludes, and the new updated state is returned along with the value of the reward obtained that is used during training.

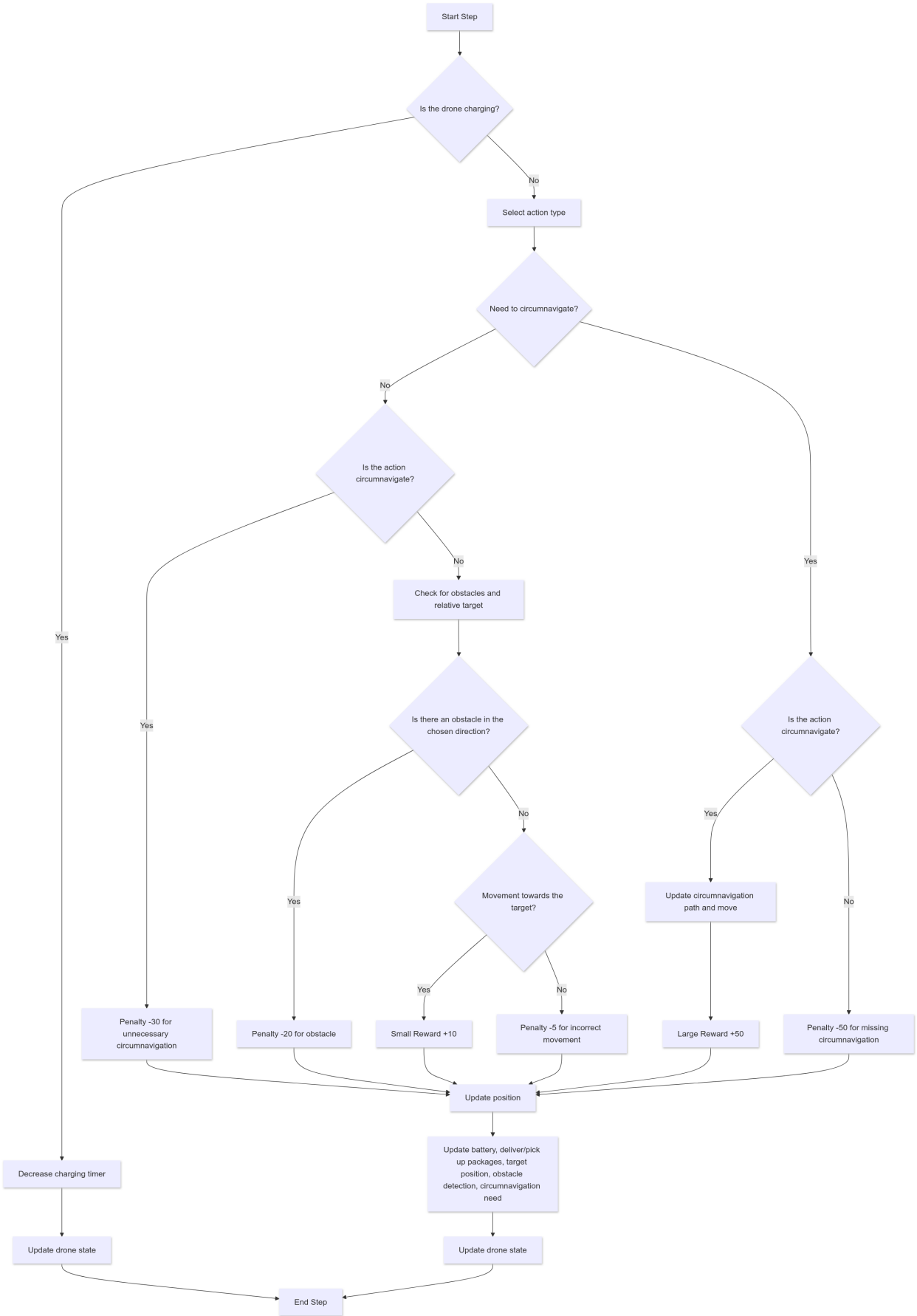

Figure 5: Drone action decision flow

5.3 Computation of the target's relative position

The relative position of the target is calculated to understand the direction in which the drone must move to approach its target. This mechanism, based on relative positioning, enables the ability to generalize, avoiding reliance on fixed x, y coordinates. Relying on absolute coordinates would cause the drone to merely memorize how to reach a fixed destination, which, if changed, would no longer be achievable through the learned behavior of the drone.

The computation is performed by comparing the current coordinates of the drone with those of the target. The horizontal and vertical differences between the drone's position and the target's are computed, and priority is given to movement in the direction where the distance is greatest. If the vertical distance is greater, the drone moves up or down; if the horizontal distance is greater or equal, it moves left or right. In cases where the drone has reached the target, the relative position of the target takes on a value indicating that no further movement is necessary.

The relative position of the target is encoded with integers ranging from zero to 4, representing the four possible directions and the skip action. These values are stored in the state of the drone, which is returned by the step method, and will be used to define a dimension of the q-table that will be exploited to train the drone to move towards its targets.

```
y_drone, x_drone = drone_position
y_target, x_target = target_position

delta_x = x_target - x_drone
delta_y = y_target - y_drone

if abs(delta_y) > abs(delta_x):
   if delta_y < 0:
      return ActionType.UP.value
   elif delta_y > 0:
      return ActionType.DOWN.value

if abs(delta_x) >= abs(delta_y):
   if delta_x < 0:
      return ActionType.LEFT.value
   elif delta_x > 0:
      return ActionType.RIGHT.value

return ActionType.SKIP.value
```

The method for calculating the relative position is depends on an operation executed at each invocation of the step method, determining the drone's target based on its current state. If the drone's battery is low or there are no more packages to deliver, the target becomes a charging station. If the drone is carrying a package, the target is the delivery point; otherwise, the target is the warehouse. This logic dynamically determines where the drone should go depending on the situation, providing a straightforward solution that allows the drone to move decisively toward its objective with minimal uncertainty.

5.4 Obstacle detection

To simulate a sensor-based mechanism that enables drones to detect obstacles in the adjacent cells around their current position and avoid collisions, the following operations are applied. At each step, the drone checks for obstacles above, below, to the left, and to the right of its current position. This is done by comparing the drone's position with an updated list of all possible coordinates containing obstacles, such as other drones, charging stations, and delivery points. Bad weather areas, which are considered as different types of obstacles, are handled differently and are not included in this list. Bad weather areas are managed in a different way because if they were to be generated above the drone's target, it would not be possible to circumvent them

and the drone is therefore forced to cross them. When this happens the drone will be penalized with increased battery consumption.

The method for the obstacle detection returns four values, one for each of the four directions, where 1 indicates the presence of an obstacle and 0 indicates its absence. These values are treated as boolean inputs in the q-table and are used to train the drone to avoid moving toward obstacles.

While this mechanism helps the drone avoid collisions, it does not guarantee that the drone will never get stuck. The drone can learn not to collide with obstacles, but if a static obstacle is located between the drone and its target, it might end up in a loop of repetitive movements without ever reaching its goal. To handle these situations, a learning mechanism is introduced that allows the drone to navigate around obstacles even in more challenging scenarios, enhancing its ability to operate in complex environments. This is the circumnavigation action and is explained in the next sections.

5.5 Detection of the need to circumnavigate

The circumnavigation action is learned by the drone during training and must be able to be implemented to appropriately avoid obstacles or areas where bad weather is present. The need to carry out a circumnavigation action is signaled to the drone through a true or false value which is stored in its state and which will be used among the values of the q-table.

To determine when the drone needs to circumnavigate an obstacle, the following logic is applied. If the drone has already started a circumnavigation path in a previous step, the circumnavigation flag is immediately set to true because the drone should continue following the already calculated path.

The need to circumnavigate an obstacle or bad weather area must be handled differently, because bad weather areas are not treated as real obstacles since in some cases it may be necessary to cross them, and their detection must also take place in a different way, allowing evaluation beyond just a single nearby cell.

5.5.1 Circumnavigation in case of obstacles

If there is no preexisting path, the drone assesses whether circumnavigation is necessary by analyzing the situation relative to the target. Specifically:

Obstacle between the drone and the target: if there is an obstacle directly between the drone's current position and the target's position, circumnavigation is required. This is the first case where the flag is set to true.

If the target is above or below the drone: the drone checks if there are obstacles to the left or right of its position. This check is important because even if the target is directly above or below the drone, lateral obstacles may cause block situations.

If the target is to the left or right of the drone: the drone checks for obstacles in the vertical directions. Similarly, this check helps manage situations where nearby obstacles, even if not directly between the drone and the target, could block the drone's path.

These conditions are set to ensure that the drone not only avoids moving into obstacles but also recognizes when it might become stuck. This prompts the drone to initiate a circumnavigation path to avoid the obstacle. This system enables the drone to handle complex scenarios where simply avoiding obstacles may not be enough to reach the target.

5.5.2 Circumnavigation in case of bad weather

To detect adverse weather-affected zones, the drone considers all known adverse weather-affected areas and checks if its current path will cross through them. When the drone identifies that it is about to enter in these areas, it tries to determine if it will have to pass through more than one cell subject to these conditions. If this is the case, the flag for the need to circumnavigate is set to

"true," indicating that the drone must activate the mechanism to perform the circumnavigation action.

The detection depends on the drone's direction of movement. If it is moving upward, it checks if there are bad weather areas above it; if moving downward, it checks below, and similarly for left and right movements. If a adverse weather-affected zone is detected, the drone continues checking along its path, cell by cell, to see if it will need to cross more than one weather-affected cell. If the drone detects that he will have to cross at least two adjacent cells affected by bad weather, then the true value is returned which will allow him to understand that he needs to choose circumnavigation as his next action.

5.6 Route calculation for circumnavigation

The circumnavigation path is calculated by the drone when it chooses the action to circumnavigate obstacles or adverse weather-affected areas. Upon first selecting this action, the drone uses a breadth-first search (BFS) algorithm to find a path that allows it to avoid physical obstacles and bad weather areas while still aiming to reach the target location.

In subsequent steps, once the path has been calculated, the drone will follow the individual positions in this predefined path. However, if during execution the drone encounters new obstacles that weren't present at the time of the initial calculation, the BFS algorithm will be run again to recalculate the path and circumvent the new obstacles as well. During the training process the drone learns to continue choosing the circumnavigation action once the BFS route has been calculated and has not been completely flown.

The computation of the circumnavigation path follows these steps:

- Retrieving physical obstacles: the positions of all obstacles in the environment are retrieved.
- Retrieving adverse weather-affected areas: all cells corresponding to active adverse weather-affected areas are retrieved.
- Merging obstacles: physical obstacles and bad weather areas are combined into a single list, forming a set of cells the drone must avoid.
- During the BFS execution, the algorithm:
    1. Starts from the drone's current position, exploring all possible directions (up, down, left, right).
    2. At each step, it checks whether the new position is valid (for example, not out of bounds and not occupied by obstacles).
    3. For each new position that turns out to be valid, the position is marked as already visited within a set, and a new tuple is created consisting of a new position and the current path updated with the new position, the tuple is inserted into the queue of paths to try to follow to reach the destination.
    4. When a path is found that reaches the target while avoiding all obstacles, the path is returned.
    5. If the algorithm fails to find a safe path, a new BFS search is initiated, this time ignoring the weather areas and considering only physical obstacles. In this case, the drone will traverse the adverse weather-affected areas in order to reach the destination.

```
combined_obstacles = obstacles.union(weather_zones)
queue = deque([(start_position, [])])

visited = set()
visited.add(start_position)
```

```
6
7  directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
8
9  while queue:
10    (current_position, path) = queue.popleft()
11    y, x = current_position
12
13    if current_position == target_position:
14       return path
15
16    for dy, dx in directions:
17       new_y, new_x = y + dy, x + dx
18       new_position = (new_y, new_x)
19
20       if (0 <= new_y < self.grid_size[0] and 0 <= new_x < self.grid_size[1]
21          and new_position not in combined_obstacles and new_position not in visited):
22
23             visited.add(new_position)
24             queue.append((new_position, path + [new_position]))
25
26 return self.__calculate_circumnavigation_path(start_position, target_position,
       drone_index)
```

In the last return statement, the same BFS algorithm is invoked again, but in this version, only physical obstacles are considered, excluding adverse weather-affected areas. This design choice was made because there are situations where it's impossible to find a path that avoids bad weather areas entirely. For example, a bad weather area might be generated directly over the target location the drone needs to reach. In such cases, the drone would have no alternative and would be forced to cross these areas to complete its objective.

5.7 GUI operation

During the simulation, each time all drones have executed the step method, the visualization is updated, and a series of graphical operations are performed. It starts by creating a grid representing the drone's environment. Delivery points, charging stations, and the warehouse are added to the grid, with specific values to differentiate them visually. Adverse weather-affected areas are drawn as semi-transparent rectangles, which are removed and redrawn with every update to reflect any changes.

The drones are positioned on the grid, and if they are at a charging station, their color changes based on the time spent charging. The GUI also removes and redraws the labels for various elements (drones, delivery points, warehouse, charging stations) at every step. This strategy of removing and redrawing was chosen because handling both static and mobile elements (like drones) dynamically was simpler this way.

Finally, the canvas is updated to reflect visual changes, showing the current state of the environment, such as the drones' battery levels and the number of deliveries completed.

The main goal of these rendering operations is to make the progress of the simulation as clear as possible. To achieve this, each element of the environment such as drones, charging stations, delivery points, the warehouse, and adverse weather-affected areas is identified with appropriate colors that change depending on the circumstances.

Outside the grid space, above the main viewing area, useful information is displayed to understand the progress of the simulation. These details include the remaining battery levels of the various drones and the number of deliveries completed. This informational section is crucial for monitoring the efficiency and progress of the simulation, providing a clear and immediate overview of the drones' performance and the state of their activities.

6 Testing process

During the development of the project, numerous tests were conducted to ensure the correct functioning of the implemented features and the overall system. Each feature was tested individually, observing its behavior within the simulation in isolation when possible. Later, once integrated into the system, additional tests were carried out to assess the overall behavior of the simulation and to ensure that no regressions occurred. Some issues encountered during testing were difficult to identify as they occurred sporadically, making it challenging to immediately pinpoint the cause. In such cases, in-depth observations and multiple simulation runs were required to understand and resolve the issues, thereby improving the system.

To effectively monitor the training process, the average rewards received after a certain number of episodes were calculated. These values were represented on a graph with the number of episodes on the x-axis and the average cumulative reward on the y-axis. This approach made it possible to quickly check, at the end of each training session, whether the drones were correctly learning the new actions introduced, offering a clear view of the learning progress.

To verify the system's ability to generalize, once the training was completed, simulations were conducted in which the scenarios were varied by changing the grid size and the positions of the elements. This allowed testing the system's flexibility in environments different from those used during training. Additionally, to test complex features such as obstacle circumnavigation, specific tests were carried out with environmental configurations where obstacles or walls were arranged in a way that forced the drones to use the implemented circumnavigation actions.

Finally, the graphical interface proved indispensable for monitoring the progress of the simulation, facilitating the identification of any issues or errors during system operation.

During the training process, issues with slowdowns or even system freezes were encountered in scenarios where multiple drones were being trained simultaneously. To solve this problem, it was decided to train only one drone at a time. This solution did not negatively affect the effectiveness of the training, as all obstacles present in the environment were treated equally. In other words, the presence of other drones was not necessary for learning how to avoid obstacles, as the existing obstacles were sufficient to ensure adequate training.

However, it was necessary to identify the causes of the slowdowns or freezes that occurred with multiple drones. Two main causes were identified. The first involved the hardware used: a computer that was not particularly powerful, with performance that quickly degraded under heavy load. The second, and more significant cause, was related to the circumnavigation algorithm based on Breadth-First Search. Although BFS is effective in finding the shortest path, it is known for its inefficiency in terms of computational complexity, as it requires exploring the entire map with each recalculation.

In the current implementation, optimizing this feature was not a priority, but a more efficient algorithm, such as A*, could have significantly reduced these issues. Unlike BFS, A* uses a different approach, avoiding the need to reconstruct the entire path with each recalculation. The problems encountered seem to stem from the use of BFS, which in some cases was repeated multiple times within short time intervals for each drone. This caused a computational overload that, in the case of multiple drones, the hardware used was unable to handle.

7 Evaluation and analysis of the results

7.1 Assessment of the simulation in relation to the project objectives

An evaluation follows regarding the achievement of the project objectives:

- Creation of the simulation environment: the environment was designed to include all the features and entities initially defined: drones, charging stations, warehouses, packages to be delivered, and adverse weather conditions. Each component of the environment is equipped with specific properties that reflect real behavior and interactions, and these

entities interact with each other in a consistent and functional manner. The interactions between components are well-implemented: drones correctly manage their navigation and interaction with obstacles and other elements in the environment, such as charging stations or packages, while adverse weather conditions appropriately influence the drones' paths, leading to the need for circumnavigation. This demonstrates that the simulation environment is complete and accurate in relation to the initial project objectives.

- Training System: the reinforcement learning training process was successfully implemented. Based on the tests conducted, the drones have effectively learned to select the best actions depending on the current state of the environment. The actions initially intended to be taught to the drones through the training process have been learned, enabling them to optimally navigate, avoid obstacles, handle adverse weather conditions, and complete deliveries. Furthermore, the drones demonstrated the ability to generalize, applying the knowledge they acquired to different environmental contexts. This indicates that the training system has fully achieved its purpose, making the drones capable of adapting and responding effectively to changes in the simulation environment.

- Realism and efficient navigation: the developed navigation system is considered sufficiently realistic, taking into account the intrinsic limitations of a 2D scenario. The obstacle avoidance mechanisms and the overall navigation system proved to be effective, although the potential for optimizations, as discussed in previous sections, was identified. From the tests conducted, the drones demonstrated smooth navigation without encountering stalls or abnormal behaviors, responding correctly to obstacles and completing their routes without errors. This outcome confirms the robustness of the system and its ability to simulate appropriate navigation behavior for the project's objectives.

- Implement a user interface: a graphical interface has been developed to allow for effective real-time monitoring of the simulation. The visual representation of various environment elements, such as drones, charging stations, and packages, has been implemented clearly, with dynamic changes in labels and colors that adapt to different situations. The interface facilitates an immediate understanding of what is happening during the simulation, fully achieving the goal of providing an intuitive and clear user experience, allowing for detailed and accessible observation of the simulation's progress.

- Performance verification and results evaluation: the developed simulation has effectively allowed for the evaluation of the overall system performance and the results achieved by the drones through the training process. All expected functionalities have been successfully tested, and the system has demonstrated operation in line with expectations. Additionally, the simulation has identified potential areas for improvement, which are addressed in more detail in the next section. Considering these aspects, this objective can also be considered fully achieved.

7.2 Analysis of the obtained results

For the analysis of the obtained results, the list of the various functionalities tested and verified during training and simulation follows. This list includes:

1. Effectiveness of the training process: the effectiveness of the drone training process is evaluated through the graph of average cumulative rewards obtained during training episodes. This graph shows the trend of average rewards every 100 episodes, with episodes on the x-axis and average cumulative rewards on the y-axis. The graph clearly demonstrates that the training process has led to a significant improvement in drone performance. The continuous increase in average rewards indicates that the drones are effectively learning to make optimal decisions. Fluctuations in reward values may reflect variations in environmental difficulty or drone action management, but overall, the positive trend confirms

the effectiveness of the training process. This approach has allowed for monitoring and assessing the drones' progress and identifying potential issues and improvements where necessary.

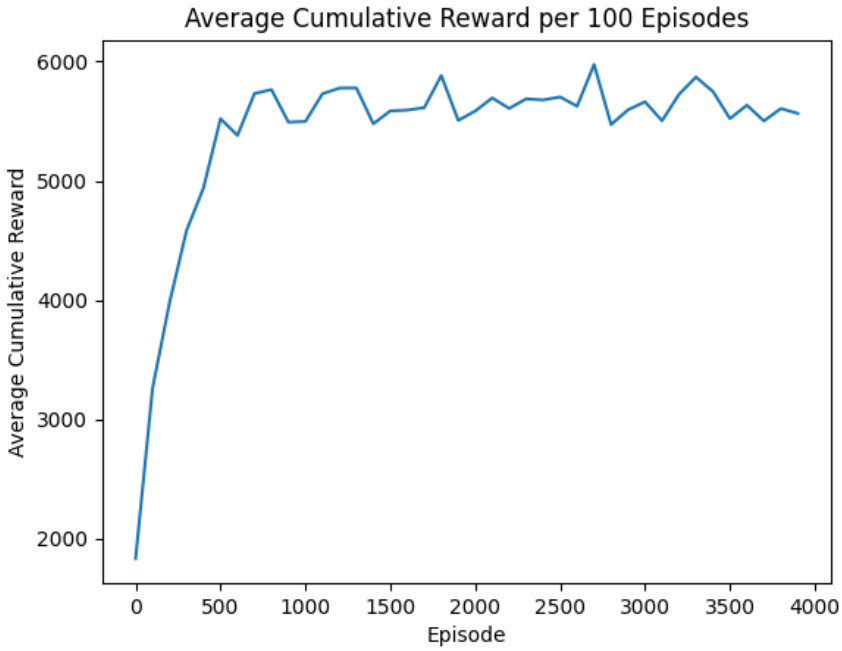

Figure 6: Average rewards graph

2. Drone movement towards the warehouse:

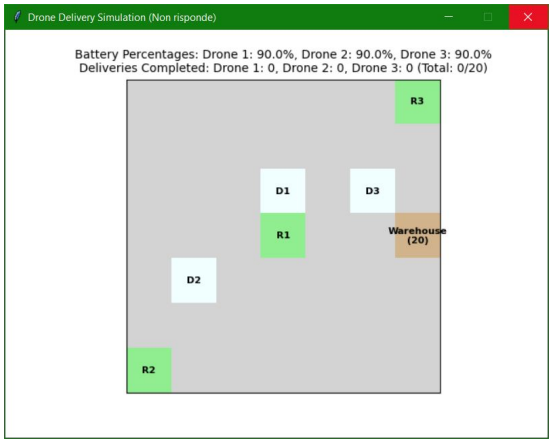

Figure 7: Drones D1, D2, and D3 heading towards the 'Warehouse'

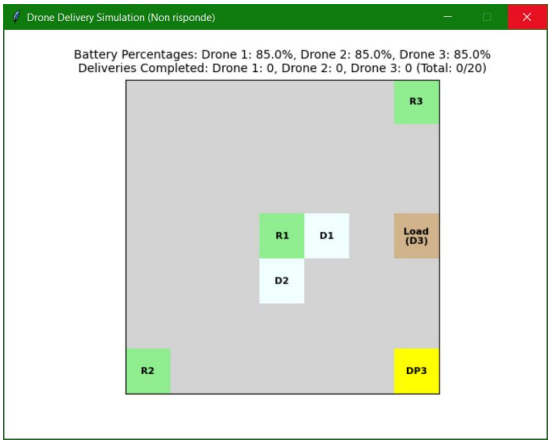

Figure 8: Drone D3 has reached the warehouse and is loading an item

In the first image shown, the simulation's start is visible, with the 3 drones, labeled D1, D2, and D3, correctly heading towards the warehouse labeled "Warehouse." The green squares represent the charging stations for each drone.

In the second image, drone D3 is depicted as having reached the warehouse. The warehouse's label has changed to indicate that an item is being loaded for delivery. The yellow label "DP3" represents the delivery point for the D3 drone, generated as a result of picking up an item for delivery.

3. Movement towards the delivery point:

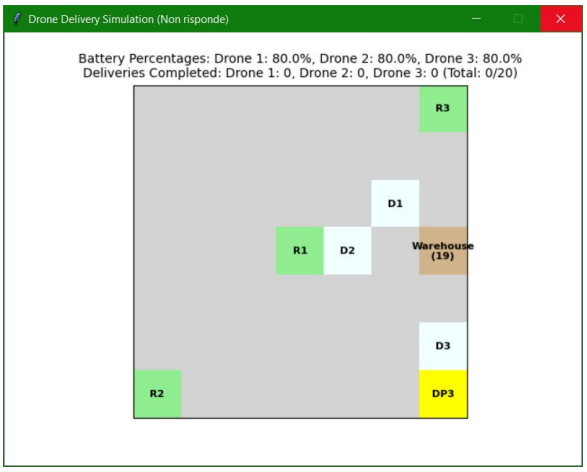

Figure 9: D3 drone is heading towards the generated delivery point

In this image, D3 drone is following the planned route towards the delivery point, while the other drones are still heading towards the warehouse.

4. Battery charging:

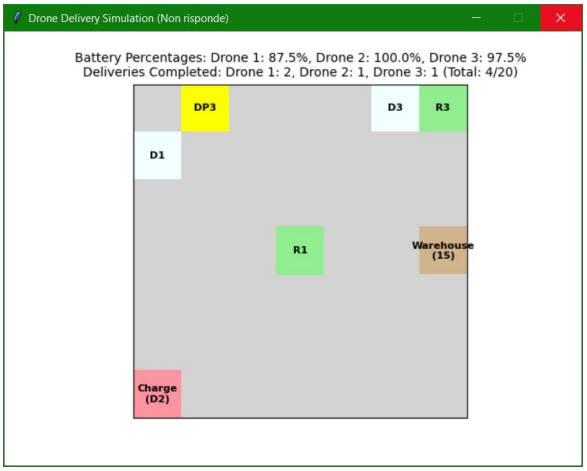

Figure 10: D2 drone is charging its battery at the charging station

In the image D2 drone has reached its charging station and is currently recharging its battery. Meanwhile, D1 drone has just delivered a package to a delivery point located near the top-left corner of the screen, and D3 drone has just finished recharging and is now heading towards its delivery point to make a delivery

5. Circumnavigation of an area affected by bad weather:

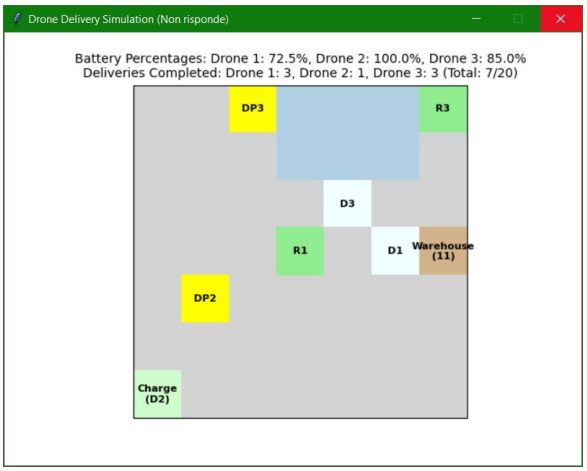

Figure 11: D3 drone detects bad weather and calculates a route to avoid it

In the displayed image, D3 drone is facing a bad weather area while attempting to reach its delivery point. The drone detects the presence of this adverse weather condition and, as a result, calculates an alternative route that will allow it to avoid the area without crossing through it.

6. End of simulation:

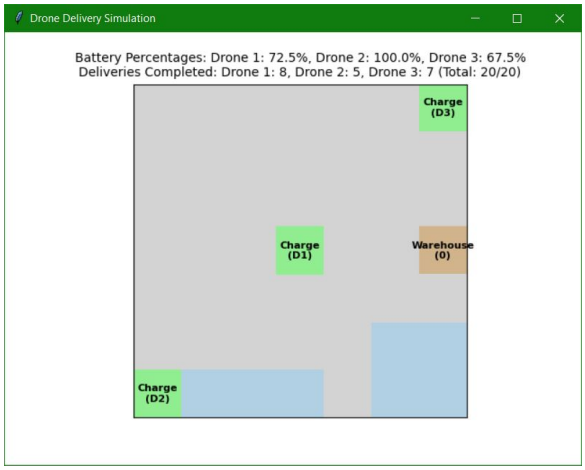

Figure 12: End of simulation: all deliveries have been completed

The final image shows the end of the simulation, in which all deliveries have been completed. The drones have returned to their resting positions, marking the conclusion of the simulation.

When examining the simulation results, several significant observations emerge.

First, the simulation system has proven effective in achieving the set objectives. It successfully verifies that key functionalities, including goods delivery and retrieval, battery charging, and the learning and application of actions for movement, obstacle avoidance, and circumvention, have been implemented effectively. The extensive application tests confirmed the correct functioning of these features, with no significant bugs or critical errors detected.

However, there are areas that could have been improved. Specifically, the choice of pathfinding algorithm was not the most performant option available. The current algorithm, while simpler to implement, led to issues during the training process due to its performance limitations. Despite

this, the algorithm was selected for its ease of implementation, as optimizing performance was not the primary goal of this project. Given the project's objectives, this choice was deemed acceptable and suitable.

Additionally, the simulation could have been made more realistic by incorporating more advanced actions that would enhance the drones' intelligence. For instance, allowing drones to decide whether to continue towards a nearby target despite having a battery level below the critical threshold could prevent abrupt route changes when they are close to their destination. This adjustment would enable them to complete deliveries before returning for recharging. Furthermore, additional actions could have been implemented, such as diversifying the types of packages delivered, adding synchronization mechanisms with the warehouse, and introducing a greater variety of obstacles, which are currently treated uniformly.

Further developments and improvements could enhance the system's efficiency and realism. One avenue is the adoption of advanced algorithms and navigation strategies to improve performance. Additionally, expanding the range of actions that drones can learn could add complexity and realism. For instance, implementing a hierarchical learning structure could allow drones to learn actions at different levels of abstraction, improving their decision-making capabilities. This might involve breaking down training into subprocesses and creating multiple q-tables to manage different aspects of the task. Such enhancements could provide a more nuanced and effective approach to training and navigation, leading to a more sophisticated simulation.

8 Conclusions

This project offered the opportunity to explore new technologies, such as a different programming language and a library for creating graphical interfaces with it. Through this work, it was possible to improve the ability to create sufficiently realistic simulations, while also acquiring new skills and some experience in the field of reinforcement learning. This last was the main reason for choosing to develop this project, as artificial intelligence is a topic of great interest today and is particularly valued by companies.

This project provided the opportunity to implement and experiment with key reinforcement learning techniques, while also deepening the understanding of fundamental AI concepts, such as designing a system capable of learning and generalizing effectively.

Undertaking a project in a new field, using technologies different from those usually employed, required time and dedication. However, there is satisfaction in being able to add this experience to one's skill set, as it has allowed for a deeper understanding of artificial intelligence concepts and its concrete applications.