IART - Assignment 1

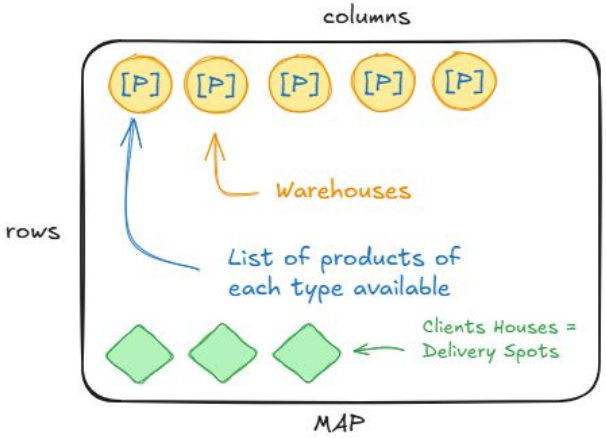
Drone Delivery

Leonardo Garcia
Marcel Medeiros
Manoela Américo

General Description

Goal - Given a fleet of drones, a list of customer orders and availability of the individual products in warehouses, schedule the drone operations so that the orders are completed as soon as possible

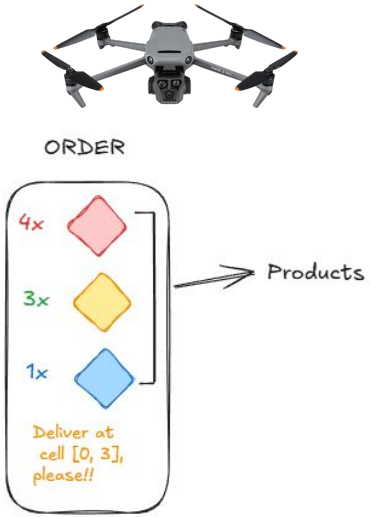
- Each **warehouse/client house** is in a position [r, c] on the grid
- Each warehouse has a list of products with different types
- Drones can take a product from warehouse A to warehouse B
- The grid is not cyclic and a drone cannot fly outside of the grid. The drones can fly over all cells within the grid.
- Warehouses does not necessarily need to have every product type available.



$[r, c]\ (0 \le r < row\ count,\ 0 \le c < column\ count)$

Orders and Drones

- The total number of product items in all orders is not greater than the total availability of product items of this product type in all warehouses
- Each product type has a fixed product weight, identical for all product items, **(product_weight ≤ payload_drone_carry)**
- The drones always use the shortest path to fly from one cell in the grid to another: **distance**$[r_a, c_a]$,$[r_b, c_b]$) = $\sqrt{|r_a - r_b|^2 + |c_a - c_b|^2}$
- Drones can: **Load, Deliver, Unload, and Wait**



Inspiration/References of Related Work

- https://github.com/msagi/hash-code-2016-qualification
- https://www.kaggle.com/code/spacelx/2020-hc-dd-2nd-place-solution-w-or-tools/notebook
- https://www.youtube.com/watch?v=Y-CBGk1yCdY

Formulation of the Problem

- **Solution:** represented as a sequence of commands for each drone, a tuple (drone_id, **C**, warehouse_id, product_type, num_items), where **C** is one of the commands **Load, Deliver, Unload or Wait.**
- **Neighborhood/Mutation Functions:** swap two deliveries between drones, change the order of deliveries for a single drone or insert a "wait" command at a random point.
- **Crossover Function:** combine two solutions (parents) and select a random point in the command sequences to swap after the crossover point to create two offsprings. **Ensure the offsprings are valid**.
- **Hard Constraints:**
- 1. A drone cannot carry more than its maximum payload.
- 2. A drone cannot load more items than are available in a warehouse.
- 3. A drone cannot deliver more items than the ones required by an order
- 4. All commands must be completed within the simulation’s deadline.
- Evaluation Function:

$$\text{Score} = \left\lceil \frac{T - t}{T} \times 100 \right\rceil$$

where **T** is the number of turns, and **t** is the turn in which the order is completed.

Work carried out so far

- Programming language: Python

```python
class Grid:
  def __init__(self, rows: int, cols: int):
    self.rows = rows # Number of rows in the grid
    self.cols = cols # Number of columns in the grid
```

```python
class Product:
    def __init__(self, product_id: int, weight: int):
        self.product_id = product_id  # Unique product identifier
        self.weight = weight  # Weight of a single unit of this product
```

```python
class Warehouse:
    def __init__(self, warehouse_id: int, location: tuple, stock: dict):
        self.warehouse_id = warehouse_id  # Warehouse identifier
        self.location = location  # Tuple (row, col) on the grid
        self.stock = stock  # Dictionary {product_id: quantity}
```

```python
class Order:
    def __init__(self, order_id: int, location: tuple, items: dict):
        self.order_id = order_id  # Unique order identifier
        self.location = location  # Tuple (row, col) on the grid
        self.items = items  # Dictionary {product_id: quantity}
        self.completed = False  # Whether the order has been fulfilled
```

Work carried out so far

```
class Drone:
    def __init__(self, drone_id: int, max_load: int, location: tuple):
        self.drone_id = drone_id  # Unique drone identifier
        self.max_load = max_load  # Maximum weight capacity
        self.location = location  # Tuple (row, col) on the grid
        self.inventory = {}  # Dictionary {product_id: quantity}
        self.commands = []  # List of issued commands
```

```
class Simulation:
    def __init__(self, grid: Grid, drones: list, warehouses: list, orders: list, deadline: int):
        self.grid = grid
        self.drones = drones  # List of Drone objects
        self.warehouses = warehouses  # List of Warehouse objects
        self.orders = orders  # List of Order objects
        self.deadline = deadline  # Maximum simulation time
        self.current_time = 0  # Tracks turns
```