Fly-in

Drones are interesting.

Summary: Design an efficient drone routing system that navigates multiple drones through connected zones while minimizing simulation turns and handling movement constraints.

Version: 1.1

Contents

Chapter I

Foreword

Drones have been used to herd sheep in New Zealand, replacing sheepdogs with buzzing aerial shepherds. In Japan, office buildings deploy drones that play loud music and flashlights to literally chase overworked employees home. One drone was trained to paint graffiti on walls mid-flight — a rebellious blend of tech and street art. In Sweden, scientists used drones to sniff out whale poop floating on the ocean to study endangered species. Some experimental drones are shaped like birds or insects to spy without being noticed, flapping wings and all. There's even a drone that flies by flapping soap bubbles, no propellers involved. In volcano research, a drone once flew straight into an eruption cloud, melted mid-air, but managed to send back data just seconds before disintegration. And in South Korea, synchronized drone shows have replaced fireworks — safer, silent, and somehow even more magical.

The phrase comes from the idea that the wheel is a brilliant invention that's been around forever and works really well. Since there's nothing wrong with it, trying to invent it again wouldn't really help and could be a waste of time—especially when that time could be spent solving new problems instead.

In programming, this happens when someone builds something from scratch that already exists — like writing your own sorting algorithm or framework when solid, open-source versions are already out there. But it's not all bad: doing it yourself can be a great way to learn how things work under the hood. The key is to find a balance — don't rebuild everything, but take time to explore how the tools you're using actually work. That way, you'll grow as a developer without getting stuck reinventing the same old wheels.

Chapter II

AI Instructions

● Context

During your learning journey, AI can assist with many different tasks. Take the time to explore the various capabilities of AI tools and how they can support your work. However, always approach them with caution and critically assess the results. Whether it's code, documentation, ideas, or technical explanations, you can never be completely sure that your question was well-formed or that the generated content is accurate. Your peers are a valuable resource to help you avoid mistakes and blind spots.

● Main message

☞ Use AI to reduce repetitive or tedious tasks.

☞ Develop prompting skills — both coding and non-coding — that will benefit your future career.

☞ Learn how AI systems work to better anticipate and avoid common risks, biases, and ethical issues.

☞ Continue building both technical and power skills by working with your peers.

☞ Only use AI-generated content that you fully understand and can take responsibility for.

● Learner rules:

• You should take the time to explore AI tools and understand how they work, so you can use them ethically and reduce potential biases.

• You should reflect on your problem before prompting — this helps you write clearer, more detailed, and more relevant prompts using accurate vocabulary.

• You should develop the habit of systematically checking, reviewing, questioning, and testing anything generated by AI.

• You should always seek peer review — don't rely solely on your own validation.

⬤ **Phase outcomes:**

- Develop both general-purpose and domain-specific prompting skills.

- Boost your productivity with effective use of AI tools.

- Continue strengthening computational thinking, problem-solving, adaptability, and collaboration.

⬤ **Comments and examples:**

- You'll regularly encounter situations — exams, evaluations, and more — where you must demonstrate real understanding. Be prepared, keep building both your technical and interpersonal skills.

- Explaining your reasoning and debating with peers often reveals gaps in your understanding. Make peer learning a priority.

- AI tools often lack your specific context and tend to provide generic responses. Your peers, who share your environment, can offer more relevant and accurate insights.

- Where AI tends to generate the most likely answer, your peers can provide alternative perspectives and valuable nuance. Rely on them as a quality checkpoint.

✓ Good practice:

I ask AI: "How do I test a sorting function?" It gives me a few ideas. I try them out and review the results with a peer. We refine the approach together.

✗ Bad practice:

I ask AI to write a whole function, copy-paste it into my project. During peer-evaluation, I can't explain what it does or why. I lose credibility — and I fail my project.

✓ Good practice:

I use AI to help design a parser. Then I walk through the logic with a peer. We catch two bugs and rewrite it together — better, cleaner, and fully understood.

✗ Bad practice:

I let Copilot generate my code for a key part of my project. It compiles, but I can't explain how it handles pipes. During the evaluation, I fail to justify and I fail my project.

Chapter III

Common Instructions

III.1 General Rules

- Your project must be written in **Python 3.10 or later**.

- Your project must adhere to the **flake8** coding standard.

- Your functions should handle exceptions gracefully to avoid crashes. Use `try-except` blocks to manage potential errors. Prefer context managers for resources like files or connections to ensure automatic cleanup. If your program crashes due to unhandled exceptions during the review, it will be considered non-functional.

- All resources (e.g., file handles, network connections) must be properly managed to prevent leaks. Use context managers where possible for automatic handling.

- Your code must include type hints for function parameters, return types, and variables where applicable (using the `typing` module). Use `mypy` for static type checking. All functions must pass mypy without errors.

- Include docstrings in functions and classes following PEP 257 (e.g., Google or NumPy style) to document purpose, parameters, and returns.

III.2 Makefile

Include a `Makefile` in your project to automate common tasks. It must contain the following rules (mandatory lint implies the specified flags; it is strongly recommended to try `-strict` for enhanced checking):

- **install**: Install project dependencies using `pip`, `uv`, `pipx`, or any other package manager of your choice.

- **run**: Execute the main script of your project (e.g., via Python interpreter).

- **debug**: Run the main script in debug mode using Python's built-in debugger (e.g., pdb).

- **clean**: Remove temporary files or caches (e.g., ___pycache___, .mypy_cache) to keep the project environment clean.

- **lint**: Execute the commands `flake8 .` and `mypy . --warn-return-any --warn-unused-ignores --ignore-missing-imports --disallow-untyped-defs --check-untyped-defs`

- **lint-strict** (optional): Execute the commands `flake8 .` and `mypy . --strict`

III.3   Additional Guidelines

- Create test programs to verify project functionality (not submitted or graded). Use frameworks like `pytest` or `unittest` for unit tests, covering edge cases.

- Include a `.gitignore` file to exclude Python artifacts.

- It is recommended to use virtual environments (e.g., venv or conda) for dependency isolation during development.

If any additional project-specific requirements apply, they will be stated immediately below this section.

Chapter IV

Introduction

Autonomous drones are the future of transportation. They are already used in many industries, such as agriculture, construction, and logistics. They are also used in military operations, such as surveillance and reconnaissance.

Your task is to design a system that **efficiently routes a fleet of drones** from a **central base (start)** to a target **location (end)**, while navigating this dynamic network under a set of strict constraints and optimization goals.

You'll be given a **graph** representing the network of zones, and a **set of constraints** that you must respect.

The graph is represented as a network of connected zones, where connections define possible movement paths between zones.

Chapter V

Constraints

- Any library that helps for graph logic is forbidden (such as networkx, graphlib, etc.).

- The project must be completely typesafe. Using **flake8** and **mypy** is mandatory.

- The project must be completely **object-oriented**.

This will have to be demonstrated during the peer review.

Chapter VI

Let the drone fly

Attached to this subject, you'll find multiple files that represent the network of drones as the following format:

Example:

```
nb_drones: 5

start_hub: hub 0 0 [color=green]
end_hub: goal 10 10 [color=yellow]
hub: roof1 3 4 [zone=restricted color=red]
hub: roof2 6 2 [zone=normal color=blue]
hub: corridorA 4 3 [zone=priority color=green max_drones=2]
hub: tunnelB 7 4 [zone=normal color=red]
hub: obstacleX 5 5 [zone=blocked color=gray]
connection: hub-roof1
connection: hub-corridorA
connection: roof1-roof2
connection: roof2-goal
connection: corridorA-tunnelB [max_link_capacity=2]
connection: tunnelB-goal
```

Interesting, right? To be more precise:

- The first line defines the number of drones using `nb_drones: <number>`.

- Zone definition on each line using type prefixes:

  - `start_hub: <name> <x> <y> [metadata]` marks the starting zone.

  - `end_hub: <name> <x> <y> [metadata]` marks the end zone.

  - `hub: <name> <x> <y> [metadata]` defines a regular zone.

  - The connection syntax forbids dashes in zone names (see below).

- All metadata is optional and enclosed in brackets `[...]` with default values:

  - `zone=<type>` (default: `normal`)

  - `color=<value>` (default: none)

  - `max_drones=<number>` (default: 1) - Maximum drones that can occupy this zone simultaneously

  - Tags inside brackets can appear in any order.

- Zone types:

- ○ `normal` – Standard zone with 1 turn movement cost (default)

- ○ `blocked` – Inaccessible zone. Drones must not enter or pass through this zone. Any path using it is invalid.

- ○ `restricted` – A sensitive or dangerous zone. Movement to this zone costs 2 turns.

- ○ `priority` – A preferred zone. Movement to this zone costs 1 turn but should be prioritized in pathfinding.

- Colors:

  - ○ Colors are optional and can be used for visual representation (terminal output or graphical display).

  - ○ Accepted values for `color` are any valid single-word strings (e.g., `red`, `blue`, `gray`). There is no fixed list of allowed colors.

  - ○ When colors are specified, the implementation should provide visual feedback through colored terminal output or graphical representation.

- Connections are defined using `connection:  <name1>-<name2> [metadata]`:

  - ○ Define a bidirectional connection (edge) between two zones.

  - ○ The connection syntax forbids dashes in zone names.

  - ○ Optional metadata can be specified in brackets `[...]`:
    - ∗ `max_link_capacity=<number>` (default: 1) - Maximum drones that can traverse this connection simultaneously

- Comments start with '#' and are ignored.

> The zones coordinates will always be positive integers, and there will always be a unique start and a unique end zone.

Chapter VII

Mandatory Part

As you may have guessed, the main objective is to move all drones from the start zone to the end zone in the fewest possible simulation turns.

VII.1   Pathfinding and Algorithm Requirements

- Drones may move simultaneously. The algorithm must schedule paths to maximize throughput and avoid unnecessary delays.

- Your implementation must handle:

    - Distribution of drones across multiple paths.

    - Strategic waiting when movement is not possible.

    - Avoidance of path conflicts and deadlocks.

- The algorithm must take into account:

    - Path lengths, including movement costs associated with zone types (e.g., restricted or priority).

    - Turn scheduling, to prevent drones from colliding or blocking each other.

    - Graph structure, to determine available disjoint or overlapping paths.

    - Zone capacity constraints (max_drones) and connection capacity (max_link_capacity).

- Your algorithm should be adaptable: different maps may require different routing strategies, depending on the topology and zone types.

- **Visual Representation**: Your implementation must provide visual feedback of the simulation, either through:

    - Colored terminal output showing drone movements and zone states

    - A graphical interface displaying the network and drone positions

    - Both options for enhanced user experience

- How efficient is your algorithm?

- Can it work with a large number of drones?

- What are the complexity (e.g., O(n), O(log n), etc.)?

- Are you recalculating or caching paths?

- How does it impact memory usage?

- How does your visual representation enhance understanding of the simulation?

VII.2   Zone Occupancy Rules

- By default, a zone may contain at most one drone at any given simulation turn.

- Zones with `max_drones=N` metadata can contain up to N drones simultaneously.

- The only special exceptions to occupancy rules are:

  - The `start` zone: all drones begin here and may share the space initially.

  - The `end` zone: multiple drones can arrive here and are considered delivered.

- Two drones may not enter the same zone on the same turn unless the zone's capacity allows it.

- A drone may not move into a zone that would exceed its maximum capacity.

- Connection capacity (`max_link_capacity`) defined on connections limits how many drones can traverse the same connection simultaneously.

- Drones may move simultaneously, as long as all capacity constraints are respected.

VII.3   Movement and Turn Mechanics

The simulation proceeds in discrete turns. At each turn, every drone may:

- Move to an adjacent connected zone (if capacity allows).

- Move to a connection towards a restricted zone (that requires 2 turns to be reached). In this case, the drone MUST reach its destination during the next turn. It can't wait extra turns on the connection.

- Stay in place (e.g., to wait, or if movement is blocked).

The simulation must prevent conflicts and ensure valid movement scheduling based on turn-by-turn state evaluation:

- Drones moving out of a zone free up capacity for that same turn.

- A zone must have available capacity for a drone to move into it (after all drones moving out have free up space).

- For multi-turn movements (restricted zones), the drone occupies the connection during transit and MUST arrive at the destination after the specified number of turns. It cannot wait on the connexion for an empty space in the destination zone.

Each movement between zones has a cost in turns, based on the **zone=type** of the destination:

- `normal`: 1 turn (default)

- `restricted`: 2 turns

- `priority`: 1 turn (but should be preferred in pathfinding algorithms)

- `blocked`: Inaccessible — cannot be entered

VII.4   Parser Constraints

The input file must respect the expected structure and syntax:

- The first line must define the number of drones using `nb_drones: <positive_integer>`.

- The program must be able to handle any number of drones.

- There must be exactly one `start_hub:` zone and one `end_hub:` zone.

- Each zone must have a unique name and valid integer coordinates.

- Zone names can use any valid characters but dashes and spaces.

- Connections must link only previously defined zones using `connection: <zone1>-<zone2> [metadata]`.

- The same connection must not appear more than once (e.g., `a-b` and `b-a` are considered duplicates).

- Any metadata block (e.g., `[zone=... color=...]` for zones, `[max_link_capacity=...]` for connections) must be syntactically valid.

- Zone types must be one of: `normal`, `blocked`, `restricted`, `priority`. Any invalid type must raise a parsing error.

- Capacity values (`max_drones` for zones, `max_link_capacity` for connections) must be positive integers.

- Any other parsing error must stop the program and return a clear error message indicating the line and cause.

> It's highly recommended to make your own map files on top of the ones provided in the subject for handling edge cases and error handling.

VII.5   Simulation Output Format

- The simulation must output the step-by-step movement of drones from the start to the end zone.

- Each simulation turn is represented by a line.

- A line must list all the drone movements that occur during that turn, space-separated. Each movement must follow the format: `D<ID>-<zone>`, or `D<ID>-<connection>` in case of drones still in flight toward restricted zones.

  - `D<ID>` refers to the unique drone identifier (e.g., `D1`, `D2`).
  - `<zone>` is the name of the destination zone.
  - `<connection>` is the name of the connection toward a restricted zone.

- Drones that do not move in a given turn are omitted from that line.

- Drones that reach the end zone are considered delivered and are no longer tracked.

- The simulation ends when all drones have reached the end zone.

- Example:

```
D1-roof1 D2-corridorA
D1-roof2 D2-tunnelB
D1-goal D2-goal
```

VII.6   Scoring System

- The performance of a solution is evaluated based on the **total number of simulation turns** required to route all drones from the start zone to the end zone.

- The fewer the number of turns, the better the score.

- A valid simulation must:

  - Comply with all movement and occupancy rules.
  - Correctly handle movement costs associated with zone types.
  - Respect all capacity constraints (zone and connection limits).
  - Avoid all conflicts (e.g., exceeding zone or connection capacity).

Secondary (optional) evaluation metrics may include:

- The number of drones moved per turn (efficiency of path allocation).

- The average number of turns per drone.

- The total path cost (sum of weighted movement costs across all drones).

- Quality and usefulness of visual representation.

In case of identical turn counts, solutions may be compared based on secondary metrics or code quality.

> These secondary metrics are not mandatory to compute automatically, but learners are encouraged to display them in their simulation output or documentation to help peers evaluate performance.

VII.7   Performance Benchmarks

The following performance targets define the expected optimization level your implementation must achieve.

- **Expected performance:**

  ○ Easy maps should be solved in less than 10 turns

  ○ Medium maps should be solved in 10–30 turns

  ○ Hard maps should be solved in less than 60 turns

  ○ Challenger map (optional) should aim to beat the reference record of 41 turns
    This level is purely optional and does not affect your grade.

To help you evaluate your algorithm's efficiency, here are reference performance targets based on the provided test maps:

- **Easy Maps**:

  ○ Linear path with 2 drones: Target ≤ 6 turns

  ○ Simple fork with 3 drones: Target ≤ 6 turns

  ○ Basic capacity with 4 drones: Target ≤ 8 turns

- **Medium Maps**:

  ○ Dead end trap with 5 drones: Target ≤ 15 turns

  ○ Circular loop with 6 drones: Target ≤ 20 turns

  ○ Priority puzzle with 4 drones: Target ≤ 12 turns

- **Hard Maps**:

  ○ Maze nightmare with 8 drones: Target ≤ 45 turns

  ○ Capacity hell with 12 drones: Target ≤ 60 turns

  ○ Ultimate challenge with 15 drones: Target ≤ 35 turns

- **Challenger Map** (optional — for exceptional implementations):

  ○ The Impossible Dream with 25 drones: **Reference record: 41 turns**

  ○ This quasi-unsolvable challenge is designed for algorithmic research and optimization

  ○ Solving this map demonstrates exceptional pathfinding and optimization skills

  ○ **Note**: This level is purely optional and does not affect your grade

> These benchmarks are provided as optimization targets to help you
> evaluate your algorithm's performance. Meeting these targets
> demonstrates a well-optimized implementation and will be assessed
> during peer evaluation.

- Can your algorithm meet these performance benchmarks?

- How does your solution compare to the reference targets?

- What optimizations did you implement to achieve better performance?

- Can you solve the Challenger map and beat the 41-turn record?

Chapter VIII

Readme Requirements

A `README.md` file must be provided at the root of your Git repository. Its purpose is to allow anyone unfamiliar with the project (peers, staff, recruiters, etc.) to quickly understand what the project is about, how to run it, and where to find more information on the topic.

The `README.md` must include at least:

- The very first line must be italicized and read: *This project has been created as part of the 42 curriculum by <login1>[, <login2>[, <login3>[...]]].*

- A "**Description**" section that clearly presents the project, including its goal and a brief overview.

- An "**Instructions**" section containing any relevant information about compilation, installation, and/or execution.

- A "**Resources**" section listing classic references related to the topic (documentation, articles, tutorials, etc.), as well as a description of how AI was used — specifying for which tasks and which parts of the project.

➠ **Additional sections may be required depending on the project** (e.g., usage examples, feature list, technical choices, etc.).

Any required additions will be explicitly listed below.

- A detailed description of your algorithm choices and implementation strategy must also be included.

- Documentation of the visual representation features and how they enhance the user experience.

> **i**   Your README must be written in English.

Chapter IX

Bonus Part

This Bonus part will be reviewed only if all the mandatory requirements are met.

Here are features you can implement to enhance your project:

- Exceptional performances:

  - You 'perfectly' meet the performance reference targets for all provided maps.

  - 'perfectly' means you have exactly or fewer turns than the target.

- Challenger map:

  - The Impossible Dream map is solved and beats the reference record of 41 turns.

Chapter X

Submission and peer-review

Submit your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the peer-evaluation. Don't hesitate to double-check the names of your files to ensure they are correct.

> ⚠️ Place all your files at the root of your repository.

A fully working simulation written in Python, including:

- A parser for the input file format.

- A simulation engine respecting movement and zone rules.

- A pathfinding algorithm (or multiple) capable of minimizing total turns.

- A visual representation system (terminal colors and/or graphical interface).

- A terminal or log output that follows the specified format.

> ℹ️ Note that we may ask you to explain your code or possibly even to write some code. Make sure to be prepared for this.

> ℹ️ Evaluation maps may be different from the ones provided in the subject.

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behaviour change, a few lines of code to write or

rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific time frame is defined as part of the evaluation.
You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may vary from one evaluation to another for the same project.