Test Case Generator for Metro Map Planning Assignment

1. Overview

This script, test_case_generator.py, is a tool designed to produce a wide variety of test cases for the "Metro Map Planning" assignment. It allows for the programmatic generation of .city files based on a set of specified parameters such as grid size, number of metro lines, and turn limits.
The primary purpose of this generator is to create a comprehensive and robust test suite for evaluating student submissions. It can produce test cases that are guaranteed to be solvable, deliberately unsolvable, or completely random.

2. Key Features

- **Multiple Generation Modes:** The script can generate test cases with different solvability guarantees:
    - constructive: Guaranteed to be **satisfiable** (solvable).
    - unsat: Deliberately constructed to be **unsatisfiable**.
    - random: Randomly generated, with no guarantee of solvability.
- **Arbitrary Turn Limit (J):** The constructive mode uses a Breadth-First Search (BFS) pathfinding algorithm, which can find valid, non-colliding paths for **any value of J**, not just a limited subset.
- **Batch Generation:** The script can generate multiple test cases with a single command (--count).
- **Reproducibility:** Test cases can be regenerated consistently by using a specific random seed (--seed), which is essential for fair grading and debugging.
- **File Organization:** Generated files can be automatically saved to a specific directory (--outdir) with a custom filename prefix (--prefix).
- **Full Scenario Support:** Supports both Scenario 1 (standard) and Scenario 2 (with popular cells).

3. Generation Modes Explained

The core strength of this generator lies in its different modes, which allow for the creation of a well-rounded test suite.

constructive (Default, Guaranteed SAT)

This mode ensures that every generated test case has at least one valid solution.
How it works:

1. It iteratively adds one metro line at a time.

2. For each line, it samples random, unoccupied start and end points on the grid.
3. It then uses a **BFS pathfinder** to find a valid path between these points that does not collide with existing paths and respects the turn limit J.
4. If a path is found, its cells are marked as occupied, and the process repeats for the next line.
5. This method is powerful because the BFS can find a path for any given J, allowing for the creation of complex but solvable problems.

If the script, when running in `constructive` mode, cannot find a valid non-colliding path for one of the metro lines, it won't get stuck in an infinite loop. It is designed to fail gracefully.

Here is the specific behavior:

1. **Retry Loop:** For each metro line it needs to create, the generator will try up to 200 times to pick a random start/end point pair and find a valid path between them.
2. **Failure:** If, after 200 attempts, it still cannot place the line (usually because the grid is too crowded or the constraints are too tight), it concludes that it's impossible to proceed.
3. **Action on Failure:** The script will then:
    - Print a clear error message to the console, like this:

```
Error: Failed to construct a satisfiable instance. Try a larger grid
or fewer lines or Just Try again ;).
```

    - Immediately terminate execution.

This ensures that the generator either produces a valid, solvable `.city` file or it stops with a helpful message explaining why it failed, preventing you from getting an incomplete or invalid test case.

unsat (Guaranteed UNSAT)

This mode is designed to test if a student's solver can correctly identify when a problem has no solution.
How it works:

- The primary strategy is to create a logically impossible constraint. It generates a random set of metro lines but forces at least one of them to be unsolvable.
- For example, when run with --J 0, it ensures that the start and end points for at least one line are **not on the same row or column**, making a 0-turn path impossible.

random

This is the most straightforward mode, providing a mix of easy, hard, and potentially

unsolvable problems.

How it works:

- It simply samples 2 * K unique coordinates from the grid and assigns them as start and end points for the K metro lines.
- There is no check for solvability, so the resulting test case could be trivial, complex, or impossible.

4. Usage Examples

The script is run from the command line.

A. Generate a simple, solvable test case:

python3 test_case_generator.py --N 10 --M 10 --K 3 --J 1 --mode constructive --output easy_sat.city

B. Generate 5 solvable test cases with a high turn limit (Scenario 2):

python3 test_case_generator.py --N 25 --M 25 --K 8 --J 4 --P 5 --mode constructive --count 5 --outdir sat_cases --prefix medium_j4

C. Generate 3 deliberately unsolvable test cases for J=0:

python3 test_case_generator.py --N 15 --M 15 --K 5 --J 0 --mode unsat --count 3 --outdir unsat_cases --prefix j0_impossible

D. Generate a reproducible set of random test cases:

python3 test_case_generator.py --N 20 --M 20 --K 10 --J 3 --mode random --count 10 --seed 42 --outdir random_seed42

5. Command-Line Arguments

Argument	Type	Default	Description
--N	int	(Required)	Grid width (number of columns).
--M	int	(Required)	Grid height (number of rows).
--K	int	(Required)	Number of metro lines.
--J	int	(Required)	Maximum number of turns allowed per line.
--P	int	0	Number of popular cells (triggers Scenario 2 if > 0).
--count	int	1	The number of test case files to generate.
--mode	str	random	The generation mode:

			constructive, random, or unsat.
--seed	int	None	A random seed for generating reproducible results.
--outdir	str	.	The directory where the generated files will be saved.
--prefix	str	case	The prefix for the output filenames (e.g., prefix_001.city).