Autonomous Delivery Agent - CSA2001 (AI & ML Project)

## Project Overview

This project involves designing and implementing an autonomous delivery agent that navigates a 2D grid-based city to deliver packages. The agent must plan paths considering static/dynamic obstacles, varying terrain costs, and efficiency constraints like time and fuel.

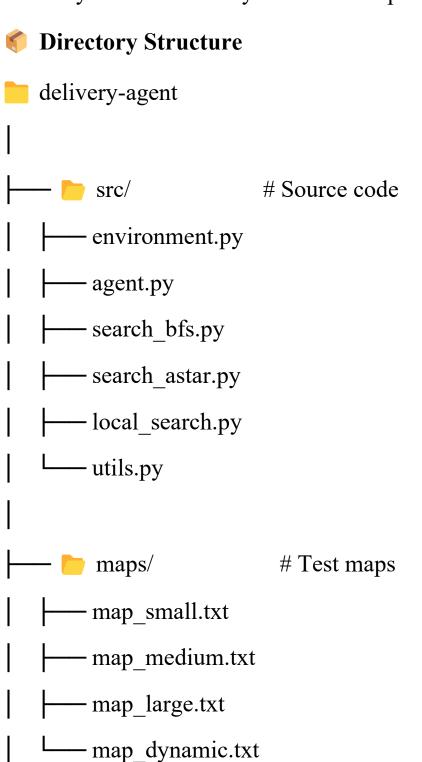
## **Objectives**

- Model environment with:
  - Static obstacles
  - Varying terrain movement costs
  - Dynamic moving obstacles (vehicles)
- Rational agent behaviour, choose actions that maximize delivery efficiency.
- Implement multiple planning strategies:

Uninformed Search: BFS / Uniform-cost

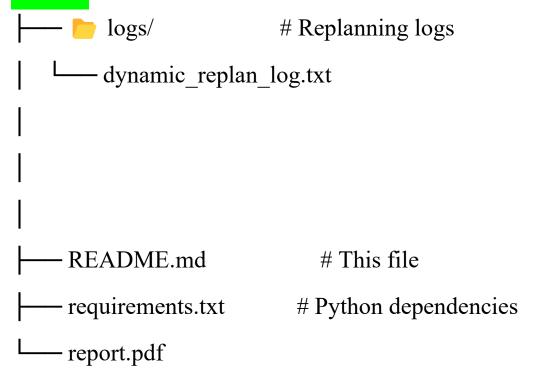
- Informed Search: A\* with admissible heuristics
- Local Search: Hill-climbing (with random restarts) or Simulated Annealing
- Compare performance of algorithms (cost, expanded nodes, execution time)

Analyse when and why each method performs better



# README AND REQUIREMENT

# FILE



## 🚺 Experimental Results

- Compare algorithms on:
  - Path cost
  - 。 Nodes expanded
  - 。 Planning time

All results, plots, and tables are included in the report.pdf.

#### Features and Deliverables

- Well-documented **Python** source code
- **4** Test maps:
  - o Small, Medium, Large, Dynamic Obstacles
- V Dynamic replanning proof-of-concept
- CLI interface for each planner

- Report ( $\leq$  6 pages) covering:
  - Environment model
  - Agent design
  - Heuristics used
  - Results and analysis
- Demo ( $\leq$  5 min) or screenshots for dynamic planning

## **\*** Constraints & Assumptions

- Grid cell movement  $cost \ge 1$
- Moving obstacles:
  - Follow deterministic schedules or
  - Move unpredictably (for local search)
- Agent movement:
  - 4-connected (Up, Down, Left, Right)
  - Diagonal optional (state in report)

## Project Requirements



CSA2001 - Fundamentals of AI and ML Project-Based Learning — Autonomous Delivery Agent

# Project Title:

Design and Implementation of an Autonomous Delivery Agent in a 2D Grid City

# **©** Objective:

To design an intelligent autonomous agent capable of navigating a **2D grid-based city** to deliver packages efficiently. The agent must plan optimal routes considering **dynamic obstacles**, **terrain costs**, and **time/fuel constraints** using various AI search algorithms.

# **✓** Functional Requirements:

#### 1. Environment Model:

- 。 Represent static obstacles (walls, buildings, etc.).
- Incorporate varying terrain movement costs (e.g., road, grass, water).
- o Include dynamic obstacles like moving vehicles.

#### 2. Rational Agent:

 The agent must make rational decisions to minimize delivery time and fuel usage.

#### 3. Search Algorithms Implementation:

- Uninformed Search: BFS or Uniform-cost Search
- Informed Search: A\* (with admissible heuristic)
- Local Search: Hill-climbing with random restarts or Simulated Annealing

#### 4. Dynamic Replanning:

Replan when new obstacles appear or when paths are blocked dynamically.

#### 5. Algorithm Comparison:

- Evaluate and compare each algorithm's performance on different maps.
- Metrics: path cost, number of nodes expanded, time taken.

#### 6. Analysis Report:

 Provide detailed analysis describing when and why certain algorithms perform better.

## **Required Deliverables:**

#### 1. Source Code (Well-documented):

- o Preferably in Python.
- o Include CLI to run each planner.
- Must include:
  - Logging of dynamic replanning
  - Modular structure with comments and doestrings

#### 2. Test Maps (Minimum 4):

- <sub>o</sub> Small
- 。 Medium
- Large
- One with dynamic moving obstacles
- Maps must be in .txt or grid file format.

#### 3. Short Report (Max 6 pages):

- Environment model
- Agent and algorithm design
- Heuristics used
- Experimental results with tables and plots
- Analysis and conclusion

## 🏶 Technical Constraints & Assumptions:

- Grid cells have integer movement costs  $\geq 1$
- Moving obstacles:
  - Move deterministically based on a known schedule (for A\*, BFS)
  - Or move unpredictably (for local search testing)
- Agent can move in **4 directions**: up, down, left, right (diagonal optional mention in report)
- Code must be testable and reproducible (include dependencies and instructions)

#### **Yes** Tools & Libraries:

- Programming Language: **Python 3.8**+
- Libraries:
  - numpy, matplotlib, pandas, pygame (for visualization), tqdm
- CLI Interface: argparse
- Version Control: Git (with README and setup instructions)