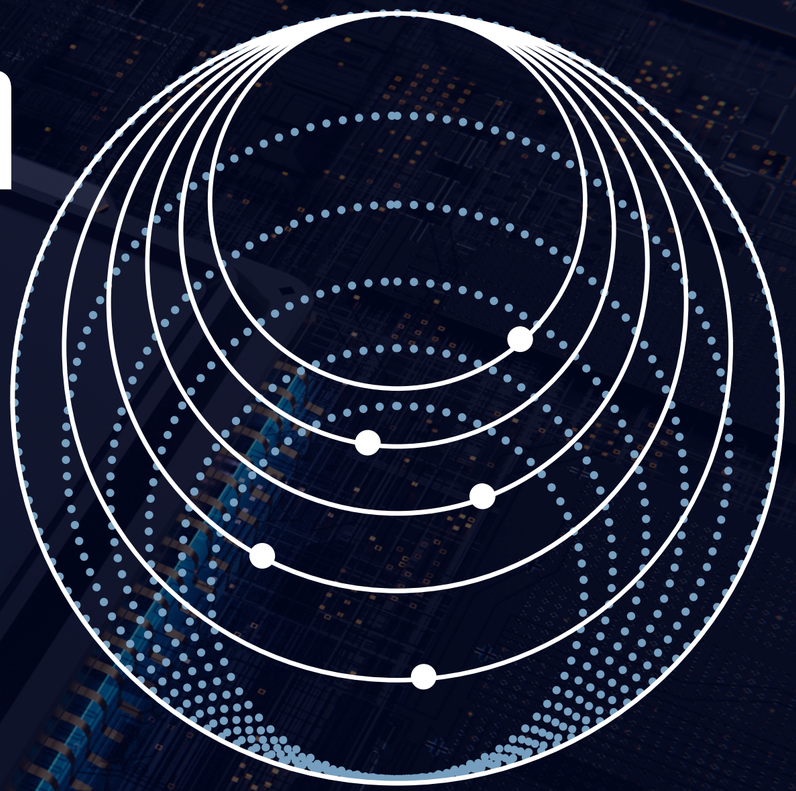
TAXI DISPATCHER SIMULATION

Simulating urban ride dispatch

Fabio Pecora & Abayomi Shosilva

Introduction



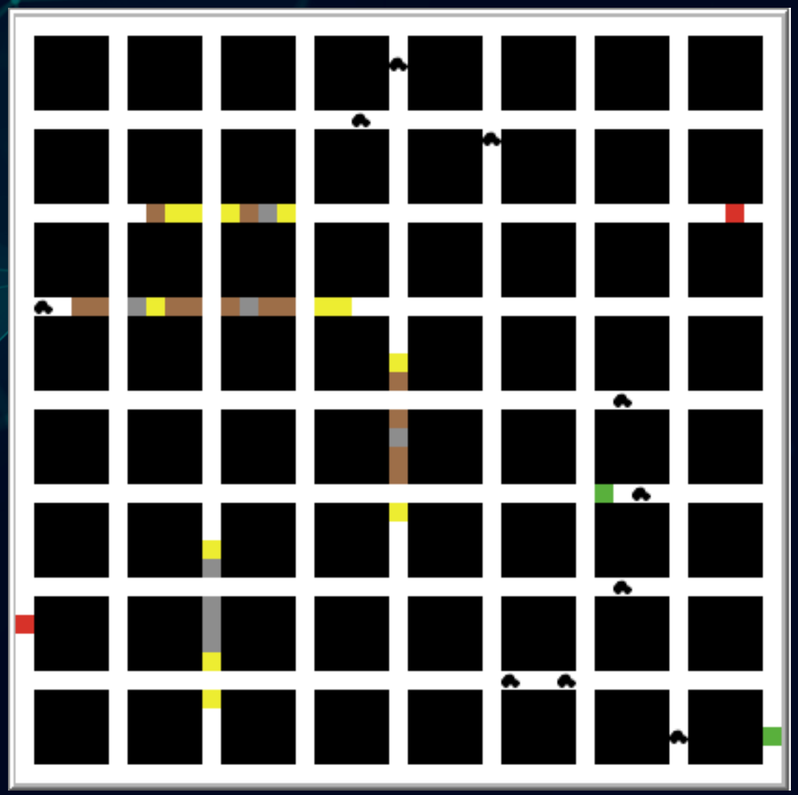
Goal: Simulate a taxi dispatch system in an urban grid

Features:

- Taxis, passengers, traffic, and ride requests
- Different dispatch strategies

Implementation: NetLogo

City:



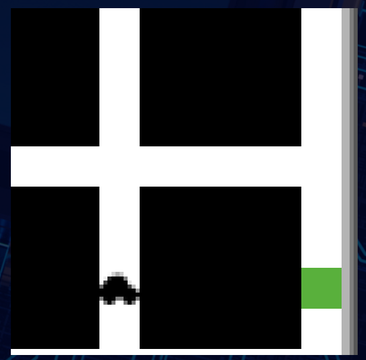
Grid Based Layout

- The city is modeled as a grid world.
- Streets are defined every 5 patches (both horizontally and vertically).
- Only these street patches are navigable by taxis.

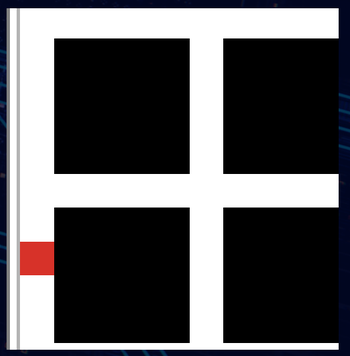
Ride Requests

- Ride requests are randomly generated throughout the simulation.
- Each ride includes:
  - A pickup location (shown in green)
  - A drop-off location (shown in red)

Pickup

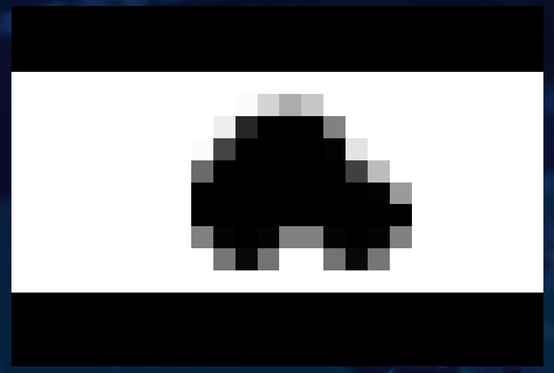


Drop Off



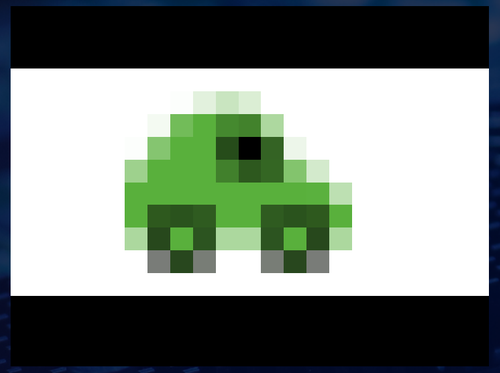
Taxi

- Taxis start on random street patches.
- They navigate only through street segments.
- Movement logic guides them to pickup/drop-off points.



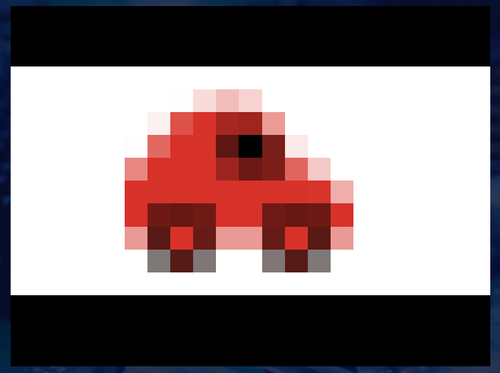
Black

Idle, waiting for ride



Green

Moving Toward a Pick Up Location



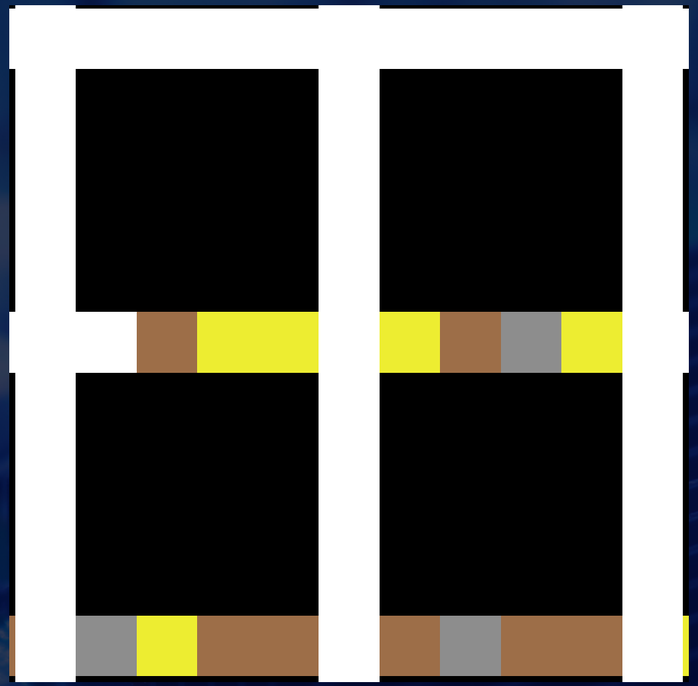
Red

Moving Toward a Drop off Location

Traffic Levels:

Traffic Levels:

- Level 1 – No Traffic
  - ⚪ Color: White
  - 🚗 Taxis move at full speed
- Level 2 – Moderate Traffic
  - 🟨 Color: Yellow
  - 🚗 Minor delays in movement
- Level 3 – Heavy Traffic
  - 🟫 Color: Brown
  - 🚗 Noticeable slowdowns
- Level 4 – Severe Traffic
  - ⚫ Color: Gray
  - 🚗 Movement is significantly delayed

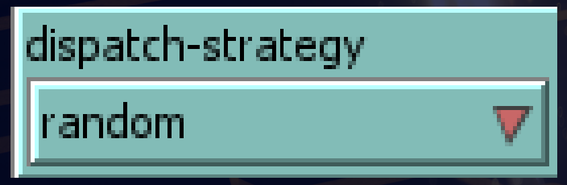


The higher the traffic level on a street, the more often taxis have to wait ticks before moving.

Dispatch Strategies Overview

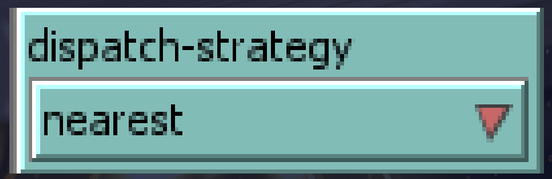
Random

Chaos



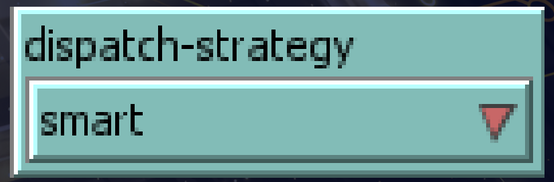
Nearest

Basic



Smart

Traffic Aware



Random Dispatch Strategy

1. Select all available taxis.
2. For each taxi:
   a. Randomly assign requests.
   b. Mark the taxi as dispatched.
   c. Remove the request from the unassigned list.

Strengths ✅

- Very easy to implement
- No computation required

Flaws ❌

- Inefficient and inconsistent
- Long travel distances
- Worst wait times

Nearest Dispatch Strategy

For each unassigned request:

a. Get the pickup location.

b. Find the closest available taxi (Euclidean distance).

c. Assign that taxi to the request.

d. Mark it as dispatched.

Strengths ✅

- Simple & effective
- Minimizes distance to pickup

Flaws ❌

- Ignores traffic congestion
- Can lead to clustered taxi usage (taxies sitting down, and taxies overused)

Smart Dispatch Strategy

1. Select all available taxis.
2. For each taxi:
   a. Sort unassigned ride requests by distance to the taxi.
   b. Consider the top 3 closest requests.
   c. Among those, pick the one with the lowest traffic at pickup.
   d. Assign that request and mark the taxi as dispatched.

Strengths ✅

- Balances distance and congestion
- Adaptive to traffic patterns

Flaws❌

More computationally expensive

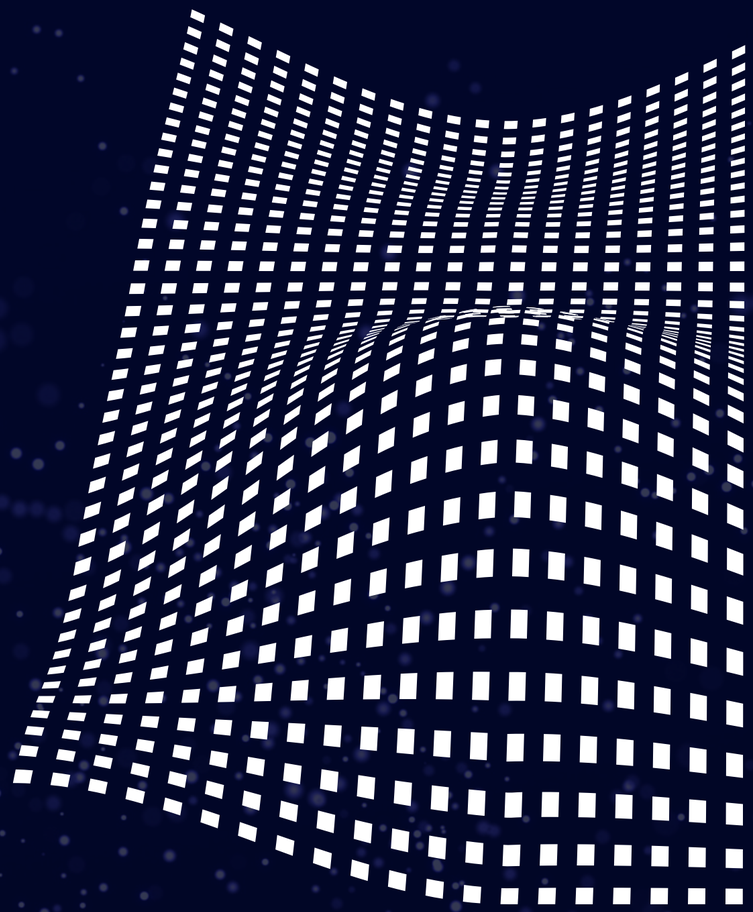
How did we test?

What is Measured:

- We calculate the average wait time for ride requests
- Defined as:
  - wait time = pickup time - request time

How It Works:

- Each time a taxi picks up a passenger, the wait time is recorded
- All wait times are stored in a list
- At any moment, average_wait_time reports the mean



Experiment:

Taxi Simulation (Parameter: 25 taxis, 2000 ticks)

Random:

Experiment #	Average Wait Time
1st	98.02
2nd	99.86
3rd	89.03
4th	106.86
5th	86.08
6th	73.86
7th	86.74
8th	80.29
9th	91.03
10th	81.24

Nearest:

Experiment #	Average Wait Time
1st	63.45
2nd	51.94
3rd	59.42
4th	61.01
5th	72.67
6th	64.14
7th	64.43
8th	65.78
9th	63.15
10th	56.85

Smart:

Experiment #	Average Wait Time
1st	45.01
2nd	38.02
3rd	52.15
4th	57.17
5th	50.42
6th	44.92
7th	42.16
8th	45.22
9th	49.64
10th	38.44

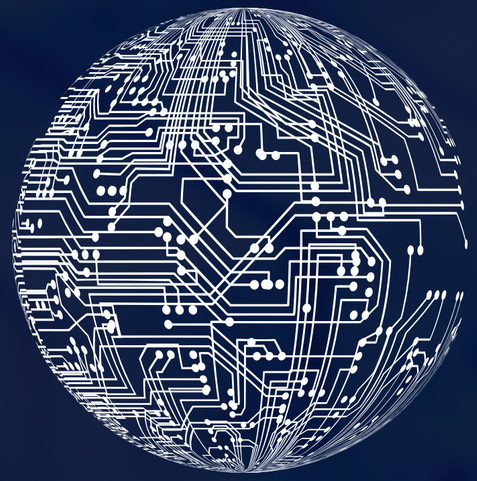
Results:

Final Average Wait Time

Strategy	Average Wait Time
Random	89.3
Nearest	62.3
Smart	46.3

Improvements obtained:

From	To	Improvement (%)
Random	Nearest	30.25
Nearest	Smart	25.63
Random	Smart	48.13



Future Work

- Dijkstra’s algorithm
- Real-time traffic updates
- Passenger behavior (ex: ride cancellation)
- Implementation of Traffic Lights