Smart Autonomous Taxi & Traffic System

Complete System Documentation

Table of Contents

System Overview

What is This System?

The **Smart Autonomous Taxi & Traffic System** is an intelligent simulation platform that demonstrates AI-powered urban mobility management. It uses **Agent-Based Modeling (ABM)** where each entity (taxi, passenger, traffic light) acts as an autonomous agent making intelligent decisions.

Key Features

- **Autonomous Taxi Agents** - Self-navigating taxis with intelligent routing
- **Priority-Based Passenger System** - Emergency, VIP, and Regular passengers
- **Ride Sharing Support** - Multiple passengers per taxi (based on taxi type)
- **Adaptive Traffic Lights** - Intelligent traffic flow management
- **Real-Time Dashboard** - Live visualization and analytics
- **Performance Monitoring** - Metrics, revenue tracking, and system health

---

Architecture & Components

System Architecture

Core Components

1. Taxi Agent (`agents/taxi.py`)

- Autonomous decision-making
- Pathfinding using NetworkX
- Priority-based passenger selection
- Ride sharing support
- Traffic awareness

2. Passenger Agent (`agents/passenger.py`)

- Priority levels (Emergency, VIP, Regular)
- Wait time tracking
- Automatic removal if waiting too long

3. Traffic Light Agent (`agents/traffic_light.py`)

- Adaptive timing based on traffic density
- Direction-based control (horizontal/vertical)
- Automatic optimization

4. City Model (`model/city_model.py`)

- Coordinates all agents
- Manages simulation state
- Handles agent spawning
- Collects statistics

5. Dashboard (`dashboard/app.py`)

- Real-time visualization
- Interactive charts and metrics
- Agent detail views
- Export functionality

Taxi Types & Fleet

Three Taxi Types

1. Economy Taxi

- **Capacity**: 1 passenger
- **Fleet Distribution**: 60% (most common)
- **Speed**: Normal (1 unit per step)
- **Icon**:
- **Use Case**: Standard single-passenger rides
- **Colors**:
  - Idle: Yellow (#fbbf24)
  - Picking Up: Light Orange (#fb923c)
  - Transporting: Green (#10b981)

2. Premium Taxi

- **Capacity**: 2 passengers (ride sharing)
- **Fleet Distribution**: 30%
- **Speed**: Normal (1 unit per step)
- **Icon**:
- **Use Case**: Small groups, ride sharing for 2
- **Colors**:
  - Idle: Orange (#f59e0b)
  - Picking Up: Orange (#f97316)
  - Transporting: Dark Green (#059669)

3. Luxury Taxi

- **Capacity**: 3 passengers (maximum ride sharing)
- **Fleet Distribution**: 10% (rarest)
- **Speed**: Normal (1 unit per step)
- **Icon**:
- **Use Case**: Large groups, maximum efficiency
- **Colors**:
  - Idle: Dark Orange (#d97706)
  - Picking Up: Red-Orange (#ea580c)
  - Transporting: Darker Green (#047857)

Fleet Distribution Example

With 10 taxis:

- **Economy**: 6 taxis (60%)
- **Premium**: 3 taxis (30%)
- **Luxury**: 1 taxi (10%)

Total Capacity: 6×1 + 3×2 + 1×3 = 15 passengers

Key Differences

Feature	Economy	Premium	Luxury
Capacity	1	2	3
Fleet %	60%	30%	10%
Ride Sharing			
Max Passengers/Trip	1	2	3

Important Note: All taxi types move at the same speed. The difference is **capacity**, not speed.

Priority System

Three Priority Levels

1. Emergency (Highest Priority)

- **Priority Score**: 3
- **Spawn Rate**: 5% of passengers
- **Max Wait Time**: 100 steps
- **Revenue Multiplier**: 2.0x ($20.00 per ride)
- **Icon**:
- **Color**: Red (#ef4444)
- **Marker Size**: Large (28px)
- **Use Case**: Urgent medical/emergency situations

2. VIP (High Priority)

- **Priority Score**: 2
- **Spawn Rate**: 15% of passengers
- **Max Wait Time**: 300 steps
- **Revenue Multiplier**: 1.5x ($15.00 per ride)
- **Icon**:
- **Color**: Orange (#f59e0b)
- **Marker Size**: Medium (24px)
- **Use Case**: Premium/preferred customers

3. Regular (Standard Priority)

- **Priority Score**: 1
- **Spawn Rate**: 80% of passengers
- **Max Wait Time**: 500 steps
- **Revenue Multiplier**: 1.0x ($10.00 per ride)
- **Icon**:
- **Color**: Blue (#3b82f6)
- **Marker Size**: Small (20px)
- **Use Case**: Standard passengers

Priority Assignment

When a passenger spawns:

```
rand = random.random()
if rand < 0.05:          # 0-5% = Emergency
    priority = "emergency"
elif rand < 0.20:        # 5-20% = VIP
    priority = "vip"
else:                    # 20-100% = Regular
    priority = "regular"
```

Priority Impact on Assignment

Selection Criteria:

1. **Priority Score** (PRIMARY) - Higher priority always selected first
2. **Distance** (SECONDARY) - Only matters when priorities are equal

Example:

- Emergency passenger at distance 10 → Selected
- Regular passenger at distance 1 → Not selected (even though closer)

Priority ALWAYS overrides distance!

Priority Impact on Revenue

Base fare: $10.00

- **Emergency**: $10 × 2.0 = **$20.00**
- **VIP**: $10 × 1.5 = **$15.00**
- **Regular**: $10 × 1.0 = **$10.00**

Passenger Assignment Mechanism

How Taxis Find Passengers

The system uses a **decentralized "pull" model** where taxis actively search for passengers (not a central dispatch system).

Assignment Process

Step 1: Passenger Spawns

- Passenger appears at random road position
- Status: `"waiting"`
- Position added to grid

Step 2: Taxis Scan (Every Simulation Step)

All idle taxis simultaneously scan for passengers:

```
for agent in self.model.schedule.agents:
    if isinstance(agent, Passenger) and agent.status == "waiting":
        if len(self.passengers) < self.capacity:  # Check capacity
            distance = calculate_distance(taxi.position, passenger.position)
            priority_score = get_priority_score(passenger.priority)
            # Store for evaluation
```

Step 3: Taxi Evaluates & Selects

Each taxi independently:

1. Collects all available passengers (status = "waiting")
2. Calculates distance to each
3. Gets priority score for each
4. Sorts by: **Priority (descending), then Distance (ascending)**
5. Selects the best match

Step 4: Taxi Commits (Assignment Happens)

```
self.status = "picking_up"  # Taxi commits!
pickup_location = selected_passenger.position
self.path = calculate_path(taxi.position, pickup_location)
```

This is the moment of assignment: Taxi sets status to `"picking_up"` and calculates route.

Step 5: Taxi Moves to Passenger

- Taxi follows calculated path
- Moves one position per step
- Checks if reached passenger location

Step 6: Pickup Complete

```
if self.position == pickup_location:
    self.passengers.append(passenger)
    passenger.status = "in_taxi"
    self.status = "transporting"
```

Assignment Rules

1. **Priority Overrides Distance**
    - Emergency always selected before VIP
    - VIP always selected before Regular
    - Distance only matters for same priority
2. **Capacity Constraints**
    - Economy: Can only pick 1 passenger
    - Premium: Can pick up to 2 passengers
    - Luxury: Can pick up to 3 passengers
3. **Competition Handling**
    - Multiple taxis can choose the same passenger
    - First taxi to arrive wins
    - Other taxis re-evaluate next step

How to Identify Assignment in Dashboard

1. **Visual Connection Lines**
    - Colored line connects taxi to passenger
    - Line color = passenger priority color
2. **Passenger Tooltip**
    - Shows "Taxi #X coming!" if assigned
3. **Click Passenger**
    - Modal shows "Taxi Assignment" section
    - Displays: Taxi ID, Type, Status, Position, Distance
4. **Click Taxi**
    - Modal shows "Picking Up" section
    - Shows which passenger taxi is heading to

Distance & Proximity Detection

Two-Stage Proximity System

Stage 1: Initial Selection (Manhattan Distance)

Manhattan Distance Formula:

```
Distance = |x1 - x2| + |y1 - y2|
```

Example:

```
Taxi at (5, 3)
Passenger at (8, 7)

Manhattan Distance = |5-8| + |3-7| = 3 + 4 = 7 units
```

Why Manhattan Distance?

- Fast calculation (O(1))
- Good approximation for grid-based movement
- Used only for initial selection/ranking

Stage 2: Exact Position Check (After Pathfinding)

After selecting a passenger, taxi:

1. Uses NetworkX pathfinding to calculate actual route
2. Moves step-by-step along path
3. Checks: `taxi.position == passenger.position` (exact match)
4. When positions match → Pickup occurs

Why Two Stages?

- **Manhattan Distance**: Fast heuristic for selection
- **Pathfinding**: Accurate route through road network
- **Exact Position Check**: Precise pickup confirmation

Note: Manhattan distance and actual path distance can differ because pathfinding follows road network, not straight lines.

Pathfinding & Routing

Pathfinding Algorithm

Uses **NetworkX shortest_path** algorithm:

- Finds shortest path through road network
- Respects road connectivity
- Handles obstacles and road structure

Route Planning for Multiple Passengers

Nearest-Neighbor Approach:

1. Start from current position
2. Find nearest destination
3. Plan route to that destination
4. Repeat for remaining destinations

Traffic Light Awareness

Taxis check traffic lights before moving:

- If red light in movement direction → Wait
- If green light → Proceed
- Allows adaptive traffic flow

Visualization & Dashboard

Dashboard Components

1. Simulation Grid

- Real-time visualization of all agents
- Color-coded by type and status
- Interactive clicking for details

2. Key Metrics Cards

- Total Fleet
- Active Taxis
- Waiting Passengers
- Passengers Served

- Average Wait Time
- Utilization Rate

3. Charts

- **Wait Time Analysis**: Trend over time
- **Taxi Utilization**: Percentage of active taxis
- **Traffic Density**: Road congestion levels

4. Performance Score

- Overall system performance (0-100)
- Based on wait times, utilization, coverage

5. Real-Time Alerts

- High wait times
- Emergency passengers waiting
- Low utilization warnings

6. Analytics & Insights

- Trend analysis
- Efficiency recommendations
- Top performer tracking

7. Cost Analysis

- Total revenue
- Cost breakdown (fuel, maintenance, base costs)
- Net profit
- Revenue by taxi type

Visual Indicators

Taxi Colors by Status

- **Yellow/Orange**: Idle (looking for passengers)
- **Orange/Red**: Picking Up (heading to passenger)
- **Green**: Transporting (carrying passengers)

Passenger Colors by Priority

- **Red**: Emergency
- **Orange**: VIP
- **Blue**: Regular

Connection Lines

- Shows taxi-to-passenger assignments
- Color matches passenger priority
- Thickness varies by priority

Performance Metrics

Key Metrics Tracked

1. **Average Wait Time**
    - Average time passengers wait before pickup
    - Lower is better
2. **Taxi Utilization**
    - Percentage of taxis that are active
    - Optimal range: 50-80%
3. **Traffic Density**
    - Percentage of road cells occupied
    - Higher = more congestion
4. **Total Passengers Served**
    - Cumulative count of completed trips
5. **Revenue**

    - Total earnings from all trips
    - Breakdown by taxi type and passenger priority
6. **Performance Score**
    - Composite metric (0-100)
    - Factors: Wait time, utilization, coverage, waiting passengers

Performance Scoring

Formula:

```
Score = (Wait_Score × 0.4) + (Util_Score × 0.3) + (Coverage_Score × 0.2) - (Wait_Penalty × 0.1)
```

Status Levels:

- 75-100: Excellent (Green)
- 50-74: Good (Yellow)
- 0-49: Needs Improvement (Red)

Technical Details

Technology Stack

- **Python 3.11+**
- **Mesa** - Agent-based modeling framework
- **NetworkX** - Graph-based pathfinding
- **Plotly Dash** - Interactive web dashboard
- **Dash Bootstrap Components** - UI components
- **Plotly** - Charts and visualizations
- **NumPy & Pandas** - Data processing

File Structure

```
AI Agent/
├── agents/
│   ├── taxi.py              # Taxi agent logic
│   ├── passenger.py         # Passenger agent logic
│   └── traffic_light.py     # Traffic light agent
├── model/
│   └── city_model.py        # Main simulation model
├── dashboard/
│   └── app.py              # Dashboard application
├── analytics/
│   └── data_collector.py   # Data collection
├── utils/
│   └── pathfinding.py      # Pathfinding utilities
├── dashboard_main.py       # Entry point
├── run_agent.bat          # Windows launcher
└── requirements.txt       # Dependencies
```

Simulation Flow

1. **Initialization**
    - Create road network
    - Spawn taxis at random positions
    - Place traffic lights at intersections
2. **Simulation Loop** (Each Step)
    - Passengers spawn (based on spawn rate)
    - All agents execute `step()` method:
        - Taxis: Find passengers, move, pick up, transport
        - Passengers: Update wait time
        - Traffic lights: Adjust timing based on traffic
    - Dashboard updates (every 500ms)
3. **Agent Actions**
    - Taxis scan for passengers

- Select best passenger (priority + distance)
- Calculate path using NetworkX
- Move along path
- Pick up when reached
- Transport to destination
- Drop off and become idle

Key Algorithms

1. **Passenger Selection**
   - Sort by priority (descending), then distance (ascending)
   - O(n log n) where n = number of passengers
2. **Pathfinding**
   - NetworkX shortest_path algorithm
   - O(V + E) for unweighted graphs
3. **Route Planning**
   - Nearest-neighbor heuristic
   - O(n²) for n destinations

Important Concepts Summary

Decentralized Assignment

- No central dispatcher
- Each taxi decides independently
- "First commitment wins" approach

Priority System

- Emergency > VIP > Regular
- Priority always overrides distance
- Affects wait time limits and revenue

Ride Sharing

- Enabled by taxi capacity
- Premium (2) and Luxury (3) can share
- Economy (1) cannot share

Two-Stage Proximity

- Manhattan distance for selection (fast)
- Exact position match for pickup (precise)

Visual Feedback

- Colors indicate status and priority
- Connection lines show assignments
- Charts show trends and performance

How to Use the System

Starting the Simulation

1. Run `run_agent.bat` (Windows) or `python dashboard_main.py`
2. Browser opens to `http://127.0.0.1:8050`
3. Dashboard loads with initial state

Controls

- **Start Button**: Begin simulation (enables interval updates)
- **Pause Button**: Stop simulation (disables updates)
- **Step Button**: Advance one simulation step
- **Reset Button**: Reset simulation to initial state
- **Speed Slider**: Adjust simulation speed (0-20)

Interacting with Dashboard

- **Hover**: See tooltips with agent information
- **Click Agent**: Open detailed modal with full information
- **Speed Control**: Adjust simulation speed for better observation
- **Export**: Download data as CSV or JSON

Tips for Analysis

1. **Set Speed to Slow/Medium** for better observation
2. **Click Agents** to see detailed information
3. **Watch Connection Lines** to see assignments
4. **Monitor Charts** for trends
5. **Check Performance Score** for overall health

Conclusion

The Smart Autonomous Taxi & Traffic System demonstrates:

- **Intelligent Agent Behavior**: Autonomous decision-making
- **Priority Management**: Critical cases handled first
- **Resource Optimization**: Efficient fleet utilization
- **Real-Time Analytics**: Performance monitoring
- **Scalable Architecture**: Handles multiple agents efficiently

This system provides valuable insights into:

- Urban mobility optimization
- Priority-based resource allocation
- Multi-agent coordination
- Real-time decision support systems

Agent Developer: Muhammad Zubair