Technical Guide: Zombie Evacuation System

Getting Started

Prerequisites

- Python 3.8 or higher
- Required packages: networkx, numpy, matplotlib, pandas

Installation

1. Clone the repository
2. Install dependencies:

```
pip install -r requirements.txt
```

System Overview

The evacuation system consists of several components:

1. City environment representation
2. Environmental data collection
3. Policy implementation
4. Simulation and evaluation
5. Results analysis and visualization

Core Components

CityGraph

PROF

Represents the city layout as a graph:

- Nodes: Locations in the city
- Edges: Pathways between locations
- Attributes: Distances, coordinates

```
class CityGraph:
    graph: networkx.Graph        # The city layout
    starting_node: int          # Starting position
    extraction_nodes: List[int] # Possible extraction points
```

ProxyData

Contains environmental sensor readings:

- Node data: Conditions at each location
- Edge data: Conditions along pathways

```
class ProxyData:
    node_data: Dict[int, Dict]  # Data for each node
    edge_data: Dict[Tuple[int, int], Dict]  # Data for each edge
```

ResourceTypes

Available resource types:

- `explosives`: Clear blocked pathways
- `ammo`: Handle hostile encounters
- `radiation_suits`: Protect from radiation

Implementing Your Solution

The EvacuationPolicy Class

Your solution should be implemented in the `EvacuationPolicy` class in `public/student_code/solution.py`:

```
def plan_evacuation(self, city: CityGraph, proxy_data: ProxyData,
                    max_resources: int) -> PolicyResult:
    """
    Plan the evacuation route and resource allocation.

    Args:
        city: The city layout
        proxy_data: Environmental data
        max_resources: Maximum total resources available

    Returns:
        PolicyResult with:
        - path: List[int] - Sequence of nodes to visit
        - resources: Dict[str, int] - Resource allocation
    """
```

PROF

Environmental Indicators

Node Indicators

- `radiation_readings`: Radiation levels (0-1)
- `thermal_readings`: Heat signatures (0-1)
- `seismic_activity`: Structural instability (0-1)
- `signal_strength`: Communications quality (0-1)

- `population_density`: Activity levels (0-1)
- `emergency_calls`: Distress signals (0-1)
- `structural_integrity`: Building condition (0-1)

Edge Indicators

- `structural_damage`: Path blockage (0-1)
- `signal_interference`: Communications disruption (0-1)
- `movement_sightings`: Activity detection (0-1)
- `debris_density`: Obstacle levels (0-1)
- `hazard_gradient`: Environmental changes (0-1)

Testing Your Solution

Single Run Test

Test your policy on a single scenario:

```
python3 run_simulation.py
```

Bulk Testing

Test across multiple scenarios:

```
python3 run_bulk_simulations.py
```

Add `--skip-city-analysis` flag to skip detailed per-city analysis:

```
python3 run_bulk_simulations.py --skip-city-analysis
```

PROF

Understanding Results

Mission Results

Each evacuation attempt produces:

- Success/failure status
- Path taken
- Resources used
- Time taken
- Detailed event log

Bulk Analysis

Multiple runs generate:

- Success rates across different city sizes
- Resource usage patterns
- Environmental correlations
- Performance visualizations

Visualizations

City Layout

- Node colors indicate status
- Edge thickness shows path taken
- Icons show resource usage
- Event log shows mission progress

Analysis Plots

- Success rate by city size
- Resource efficiency
- Environmental correlations
- Time-distance relationships
- Resource impact analysis

Tips for Development

1. Start Simple
    - Test basic pathfinding first
    - Add resource management gradually
    - Validate on single scenarios before bulk testing
2. Use Available Tools
    - NetworkX for graph operations
    - Matplotlib for custom visualizations
    - Pandas for data analysis
3. Debug Effectively
    - Check event logs for failure points
    - Analyze resource usage patterns
    - Test edge cases with different city sizes