Guía Práctica: Sistema de Evacuación Zombie

Comenzando

Puedes ejecutar este proyecto directamente en tu máquina o usando Docker. Elige el método que mejor se adapte a tus necesidades.

Opción 1: Instalación Local

Fase 1: Configuración y Familiarización

1. Clona el repositorio:

```
git clone <repository_url>
cd zombie_playgroung
```

2. Crea y activa un entorno virtual:

```
python -m venv .venv
source .venv/bin/activate  # Para Linux/Mac
# o
.venv\Scripts\activate  # Para Windows
```

3. Instala las dependencias:

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

PROF

Fase 2: Entendiendo el Sistema

1. Ejecuta una simulación de ejemplo con la política predeterminada:

```
python3 run_simulation.py
```

- Observa la visualización
- Estudia el registro de eventos
- Toma nota de las condiciones de éxito/fracaso

2. Examina las interfaces principales en `public/lib/interfaces.py`:

- `CityGraph`: Estructura de la ciudad
- `ProxyData`: Datos del entorno

- `ResourceTypes`: Tipos de recursos
- `PolicyResult`: Resultados de la política

3. Revisa la implementación de ejemplo en `public/examples/random_policy.py`

Fase 3: Exploración de Datos

1. Ejecuta múltiples simulaciones para entender la variabilidad:

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

2. Estudia los datos generados en:

```
data/policies/EvacuationPolicy/experiments/exp_<timestamp>/
├── summary.json              # Resultados generales
├── cities/                   # Escenarios individuales
│   └── city_<id>/
│       ├── definition.json   # Diseño de la ciudad
│       ├── proxy_data.json   # Datos ambientales
│       └── mission_results.json  # Resultados de la misión
└── visualizations/           # Gráficos de análisis
```

3. Analiza los patrones de datos proxy:

- Indicadores de nodos
- Indicadores de bordes
- Correlaciones con resultados

Fase 4: Ciclo de Desarrollo

PROF

1. Crea tu política en `public/student_code/solution.py`

2. Prueba escenarios individuales:

```
python3 run_simulation.py
```

Los inputs estan definidos en este lugar. Las tres cosas que va a recibir tu policy es city, proxy_data, max_resources

```python
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 with nodes and edges
            - city.graph: NetworkX graph with the city layout
            - city.starting_node: Your starting position
            - city.extraction_nodes: List of possible extraction points

        proxy_data: Information about the environment
            - proxy_data.node_data[node_id]: Dict with node indicators
            - proxy_data.edge_data[(node1,node2)]: Dict with edge indicators

        max_resources: Maximum total resources you can allocate

    Returns:
        PolicyResult with:
        - path: List[int] - List of node IDs forming your evacuation path
        - resources: Dict[str, int] - How many of each resource to take:
                    {'explosives': x, 'ammo': y, 'radiation_suits': z}
                    where x + y + z <= max_resources
    """
```

- Obtén retroalimentación rápida
- Depura situaciones específicas
- Comprende casos de fallo

3. Ejecuta pruebas masivas:

```
# Sin análisis detallado (más rápido)
python3 run_bulk_simulations.py --skip-city-analysis

# Con análisis completo
python3 run_bulk_simulations.py
```

PROF

4. Analiza resultados:

- Revisa tasas de éxito
- Estudia el uso de recursos
- Examina registros de eventos
- Analiza visualizaciones

5. Itera y mejora basándote en los datos

Fase 5: Análisis Avanzado

1. Crea visualizaciones personalizadas:

- Extiende `public/visualization/city_analysis.py`
- Agrega nuevas métricas a `public/visualization/bulk_analysis.py`

2. Agrega registros personalizados:

- Rastrea métricas adicionales
- Crea nuevos gráficos de análisis
- Genera reportes personalizados

3. Experimenta con diferentes escenarios:

- Modifica tamaños de ciudad
- Ajusta número de ejecuciones
- Cambia semillas aleatorias

Fase 6: Optimización de Rendimiento

1. Perfila tu solución:

- Tiempo de toma de decisiones
- Eficiencia de recursos
- Optimalidad de rutas

2. Ejecuta pruebas a gran escala:

```
# Aumenta el número de simulaciones
python3 run_bulk_simulations.py --n-runs 100
```

3. Genera reportes completos:

- Tasas de éxito por condición
- Patrones de uso de recursos
- Correlaciones ambientales

Comandos Útiles

PROF

Uso Básico

```
# Simulación individual con visualización
python3 run_simulation.py

# Pruebas masivas sin análisis de ciudad
python3 run_bulk_simulations.py --skip-city-analysis

# Pruebas masivas con análisis completo
python3 run_bulk_simulations.py
```

Opciones Adicionales

```
```

```
# Establecer semilla aleatoria
python3 run_simulation.py --seed 42

# Cambiar tamaño de ciudad
python3 run_simulation.py --nodes 50

# Nombre personalizado de experimento
python3 run_bulk_simulations.py --experiment-name "prueba_1"
```

Ubicación de Datos

Resultados de Simulación

```
data/policies/EvacuationPolicy/experiments/
└── exp_<timestamp>/
    ├── summary.json
    ├── cities/
    │   └── city_<id>/
    │       ├── definition.json
    │       ├── proxy_data.json
    │       ├── mission_results.json
    │       └── visualizations/
    └── visualizations/
```

Salidas de Análisis

```
data/policies/EvacuationPolicy/
└── experiments/
    └── exp_<timestamp>/
        └── visualizations/
            ├── success_rates.png
            ├── resource_efficiency.png
            ├── proxy_correlations.png
            ├── time_distance.png
            └── resource_impact.png
```

PROF

Consejos de Desarrollo

1. Usa control de versiones para tus implementaciones
2. Mantén notas sobre lo que aprendes de cada experimento
3. Crea casos de prueba sistemáticos
4. Documenta tu código de análisis personalizado
5. Respalda resultados importantes

Opción 2: Usando Docker

Prerequisitos

- Docker instalado en tu sistema

Inicio Rápido con Docker

1. Construye la imagen de Docker:

```
docker build -t zombie-sim .
```

2. Ejecuta una simulación individual:

```
docker run -v $(pwd):/app -v $(pwd)/data:/app/data zombie-sim
```

3. Ejecuta con diferentes comandos:

```
# Simulaciones masivas
docker run -v $(pwd):/app -v $(pwd)/data:/app/data zombie-sim python3
run_bulk_simulations.py

# Sin análisis de ciudad
docker run -v $(pwd):/app -v $(pwd)/data:/app/data zombie-sim python3
run_bulk_simulations.py --skip-city-analysis

# Con parámetros específicos
docker run -v $(pwd):/app -v $(pwd)/data:/app/data zombie-sim python3
run_simulation.py --nodes 50 --seed 42
```

PROF

Desarrollo con Docker

La configuración de Docker incluye montajes de volumen que permiten:

- Editar código en tu máquina local y ver cambios inmediatamente
- Preservar datos entre ejecuciones
- Acceder a las visualizaciones localmente

1. Inicia una sesión interactiva:

```
docker run -it --rm -v $(pwd):/app -v $(pwd)/data:/app/data zombie-sim
bash
```

2. Ejecuta comandos dentro del contenedor:

```
root@container:/app# python3 run_simulation.py
root@container:/app# python3 run_bulk_simulations.py
```

3. Accede a los datos generados:

- Todos los datos estarán disponibles en tu directorio local `data/`
- Las visualizaciones se pueden ver directamente desde tu sistema de archivos local

Consejos para Docker

1. Elimina contenedores no utilizados:

```
docker container prune
```

2. Reconstruye la imagen después de cambios en dependencias:

```
docker build --no-cache -t zombie-sim .
```

3. Ver logs del contenedor:

```
docker logs <container_id>
```

4. Ejecuta con variables de entorno específicas:

```
docker run -e PYTHONPATH=/app -v $(pwd):/app -v $(pwd)/data:/app/data
zombie-sim python3 run_simulation.py
```

PROF

Nota para usuarios de Windows: Reemplaza `$(pwd)` con `%cd%` en CMD o `${PWD}` en PowerShell.