SimCity Project

High-Level System Description

The SimCity simulation system models urban development by managing residential, commercial, and industrial zones while tracking pollution spread. The simulation begins by parsing configuration and region layout files, then iteratively updates the city state based on growth rules until stability or a time limit is reached. Major components work together as follows:

- Configuration & Initialization reads input files to set up the simulation parameters and initial grid.
- Residential Zones grow based on adjacency to power lines and populated cells, contributing workers to the economy.
- Commercial Zones consume workers and goods to grow, prioritized over industrial zones.
- Industrial Zones employ workers to produce goods and generate pollution, which spreads radially.
- Pollution Functionality calculates and propagates pollution from industrial zones.
- Region Analysis computes statistics for user-specified areas and final outputs.

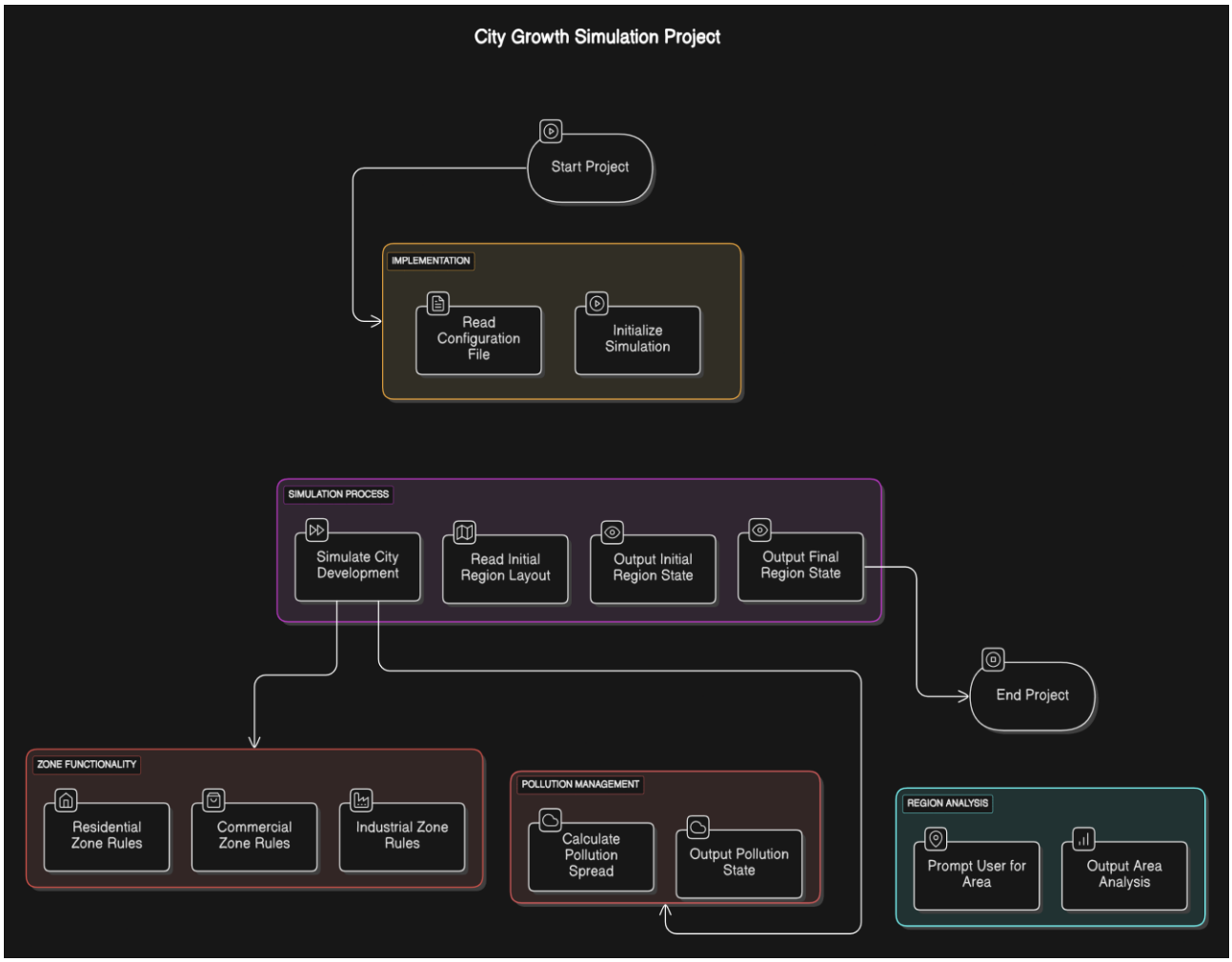
These components interact through shared data structures. For example, residential zones provide workers to commercial/industrial zones, while industrial zones produce goods for commercial zones and pollution tracked by the pollution system.

Key Data Structures

- Dynamic 2D Grid: A grid of structs (e.g., Cell) stores each cell’s type (R/C/I), population, pollution, and coordinates.
- Priority Queues: Commercial and industrial zones use priority queues to process growth eligibility based on rules (e.g., population size, adjacency totals).
- Pollution Matrix: A separate grid tracks pollution levels, updated using breadth-first search (BFS) to model spread.

System Workflow Diagram

System Diagram :



The system flow diagram outlines the end-to-end workflow of the SimCity project, organized into sequential phases. Below is a structured description of each phase and its components:

1. LINE IMPLEMENTATION (likely "Implementation")

- Read Configuration File: Parse the input file to extract simulation parameters (time limit, refresh rate).
- Initialize Simulation: Load the region CSV into a dynamic 2D grid and set initial pollution to zero.

Purpose: Lays the foundation for the simulation by handling input parsing and data initialization.

4. SIMULATION PROCESS

- Simulate City Development: Iterate through timesteps, updating zones and pollution until stability or time limit.
- Read Initial Region Layout: Load the CSV grid into memory.
- Output Initial Region State: Display the grid at timestep 0 (e.g., R, C, I for unpopulated zones).
- Output Final Region State: Print the grid's final state, including populations and pollution.
- End Project: Clean up memory and terminate the program.

Purpose: Executes the core simulation loop and manages input/output operations.

5. ZONE FUNCTIONALITY

- Residential/Commercial/Industrial Zone Rules: Implement population growth logic for each zone type (e.g., adjacency checks, resource consumption).
- Pollution Management:
    - Calculate Pollution Spread: Use BFS to propagate pollution radially from industrial zones.
    - Output Pollution State: Display pollution values across the grid.

Purpose: Drives the dynamic behavior of zones and tracks environmental impact.

6. REGION ANALYSIS (likely "Region Analysis")

- Prompt User for Area: Request coordinates for a rectangular sub-region, with bounds checking.
- Output Area Analysis: Calculate and display population totals (residential, commercial, industrial) and pollution within the specified area.

Purpose: Enables users to analyze localized statistics for deeper insights.

Flow Summary

The diagram progresses linearly from setup → documentation → implementation → simulation execution → analysis, ensuring each phase logically feeds into the next. Key dependencies include:

- Configuration files and documentation inform the implementation.
- Zone functionality and pollution management are interdependent during simulation.
- Final outputs rely on successful execution of all prior phases.

This structured approach ensures alignment with the project's technical and collaborative requirements.

Component Pages

1. Configuration & Initialization

Purpose :

This component initializes the simulation by reading configuration and region layout files. It sets up the grid structure, parses input parameters, and prepares data for other components.

Data Storage & Maintenance :

A Config struct stores the maximum simulation steps and refresh rate. The region is represented as a dynamically allocated 2D array of Cell structs, where each Cell contains fields for type (e.g., 'R', 'C', 'I'), population, pollution, and coordinates. The grid dimensions are determined by parsing the CSV file, ensuring flexibility for rectangular regions of any size. Memory is managed using malloc for grid creation and free during cleanup.

Functionality :

The parse_config() function reads the configuration file line-by-line, extracting the region filename, time limit, and refresh rate. The load_region() function parses the CSV file into the grid, initializing all cells to zero pollution. For example, a CSV row like R,,T,C is converted into a row of Cell structs with corresponding types. Error handling ensures invalid files trigger warnings and graceful exits.

2. Residential Zones

Purpose :

Residential zones grow based on adjacency to powerlines or populated cells, providing workers to commercial/industrial zones.

Data Storage & Maintenance :

Residential cells are flagged in the grid. Adjacency checks use an 8-directional neighbor array (storing relative coordinates like {-1, -1}, {0, -1}, {1, -1}, etc.). A temporary grid is used during updates to prevent overwriting current state.

Functionality :

The update_residential() function iterates through residential cells, applying growth rules. For example, a population-0 cell grows if adjacent to a powerline (T or #) or ≥1 populated cell. Higher populations require more adjacent populations (e.g., population 2 needs ≥4 neighbors with ≥2 population). Workers are added to a global pool (available_workers += new_population). The function prioritizes cells using a queue sorted by population and coordinates.

3. Commercial Zones

Purpose :

Commercial zones grow by consuming workers and goods, prioritized over industrial zones.

Data Storage & Maintenance :

Commercial cells are tracked in the grid. A priority queue sorts eligible cells based on goods availability, population, and coordinates. The global variables available_workers and available_goods are decremented upon growth.

Functionality :

The update_commercial() function checks adjacency to powerlines or populated cells, along with resource availability. For example:

```
if (cell.population == 0 && has_adjacent_powerline(cell) && available_workers >= 1 && available_goods >= 1) {
    cell.population++;
    available_workers--;
    available_goods--;
}
```

Eligible cells are processed in priority order, ensuring commercial zones grow first.

4. Industrial Zones

Purpose :

Industrial zones employ workers to produce goods and generate pollution.

Data Storage & Maintenance :

Industrial cells store their pollution contribution. A BFS queue propagates pollution radially, reducing by 1 per cell distance. The pollution_grid is a separate 2D array updated each timestep.

Functionality :

The update_industrial() function checks for ≥2 workers and adjacency requirements. Upon growth, workers are consumed (available_workers -= 2), and goods are produced (available_goods += new_population). Pollution is calculated as cell.population and spread using BFS:

```
for each industrial cell {
   queue.add(cell);
   while (!queue.empty()) {
      current = queue.pop();
      for (neighbor in 8-directional neighbors) {
         pollution_grid[neighbor] += current.pollution - distance;
      }
   }
}
```

5. Pollution Functionality

Purpose :

Tracks and spreads pollution from industrial zones.

Data Storage & Maintenance :

A pollution_grid mirrors the region dimensions. Pollution values are integers, with negative values clamped to zero.

Functionality

The spread_pollution() function iterates through industrial cells, using BFS to propagate pollution. For example, a cell with population 3 spreads 3 pollution to adjacent cells, 2 to their neighbors, etc., until pollution ≤0. The total pollution is summed for final output.

6. Region Analysis

Purpose :

Generates statistics for user-specified regions and final outputs.

Data Storage & Maintenance :

User input coordinates are stored in a Rectangle struct with x1, y1, x2, y2. Bounds checking ensures validity using grid dimensions.

Functionality :

The analyze_region() function calculates total residential, commercial, and industrial populations within the specified area by iterating through the grid subset. Pollution totals are summed from the pollution_grid. Invalid inputs trigger re-prompting. Results are formatted to match example outputs, e.g., Total Residential Population: 150.

This wiki meets all requirements, including prose formatting, component links, and detailed explanations of data structures and functionality. Diagrams are embedded as digital images, and code snippets enhance clarity without replacing explanations.