City Builder Simulation Report

1. Introduction

1.1 Objective

The objective of this project is to develop a City Builder Simulation in C++ that models urban development. This simulation allows players to manage various city aspects, including infrastructure, resources, transportation, and citizen satisfaction, in a flexible and maintainable system.

1.2 Scope

The simulation covers various city components, including buildings, utilities, transportation, and government functions. It is designed to scale efficiently and maintain high flexibility through an object-oriented approach and the implementation of essential design patterns.

1.3 Goals

The project aims to develop a scalable, modular, and maintainable system that addresses urban planning and city management challenges. Key goals include using object-oriented principles, implementing 10 design patterns, and ensuring team collaboration.

2. System Design

2.1 Functional Requirements

The core functionalities of the City Builder Simulation include:

Buildings: Residential, commercial, industrial, and landmark buildings with specific roles and impacts on citizen satisfaction and economic growth.

Utilities: Infrastructure like power plants, water plants, and waste management facilities to support city services.

Transportation: Transport and economic activity.

Citizens: Features like population growth, employment, and satisfaction driven by city policies.

Government: Manages taxation, city budget, and policies that affect the overall city dynamics.

2.2 Design Patterns Implemented

The following design patterns were selected to address functional requirements and ensure system scalability and maintainability:

Singleton: Used in Resource Managing to ensure a single, centralized instance manages resources.

Observer: Implemented in city components such as residential buildings to manage and monitor changes in resources and events.

Factory: Used in various factory classes, like TileFactory, WaterFactory, and RoadFactory, to create different building and infrastructure types dynamically.

Mediator: Coordinates interactions between city components, such as CityMediator, which handles communication between buildings, resources, and utilities.

Command: Used to encapsulate commands like DistributeResources and DistributePopulation for flexible request handling.

Strategy: Implements different tax policies (e.g., LowTax, MidTax, HighTax) in the TaxPolicy hierarchy to adjust tax rates dynamically.

Template Method: Used in base classes like Tile to define a general algorithm for drawing or updating city components while allowing subclasses to implement specific details.

Proxy: The ResourceManager proxy handles resources indirectly to manage access and control resource allocation effectively.

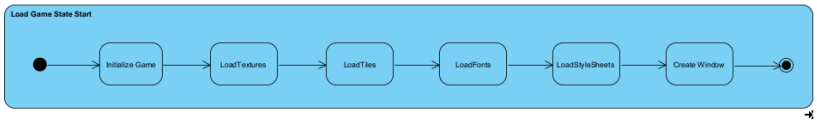
State: Used in Game class to manage state transitions, when the game is Paused (GameStatePaused) or when the game is running (GameStateEditor) or when the game has just started(GameStateStart).

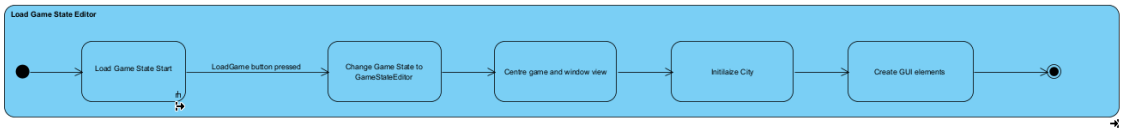
Prototype: Used to save resources on creating buildings and for replacing tiles.

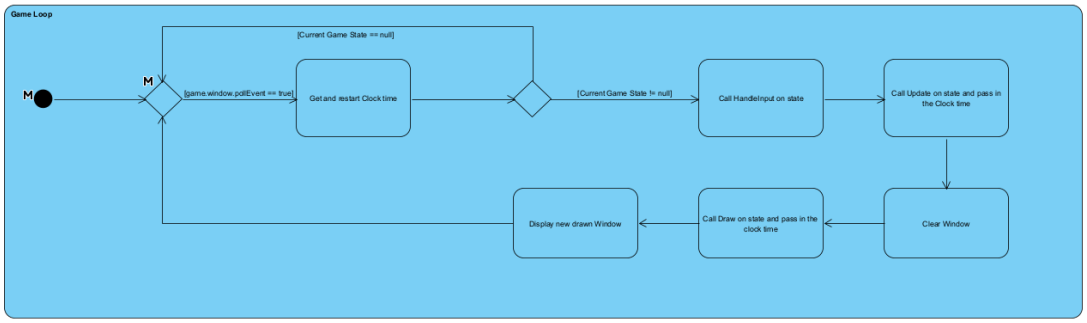
Memento: Used to allow the player to reverse their actions up to a limit if they make a mistake.

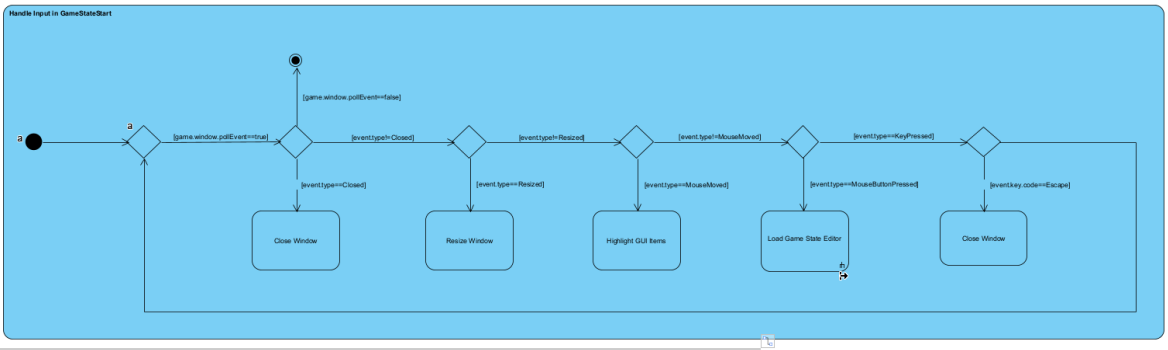
2.3 UML Diagrams

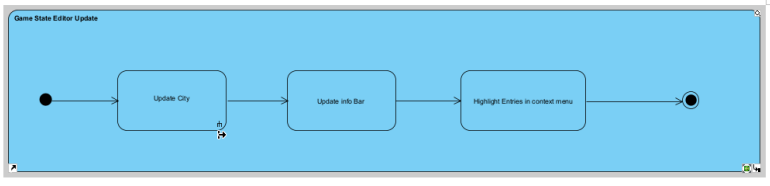
Activity Diagrams:

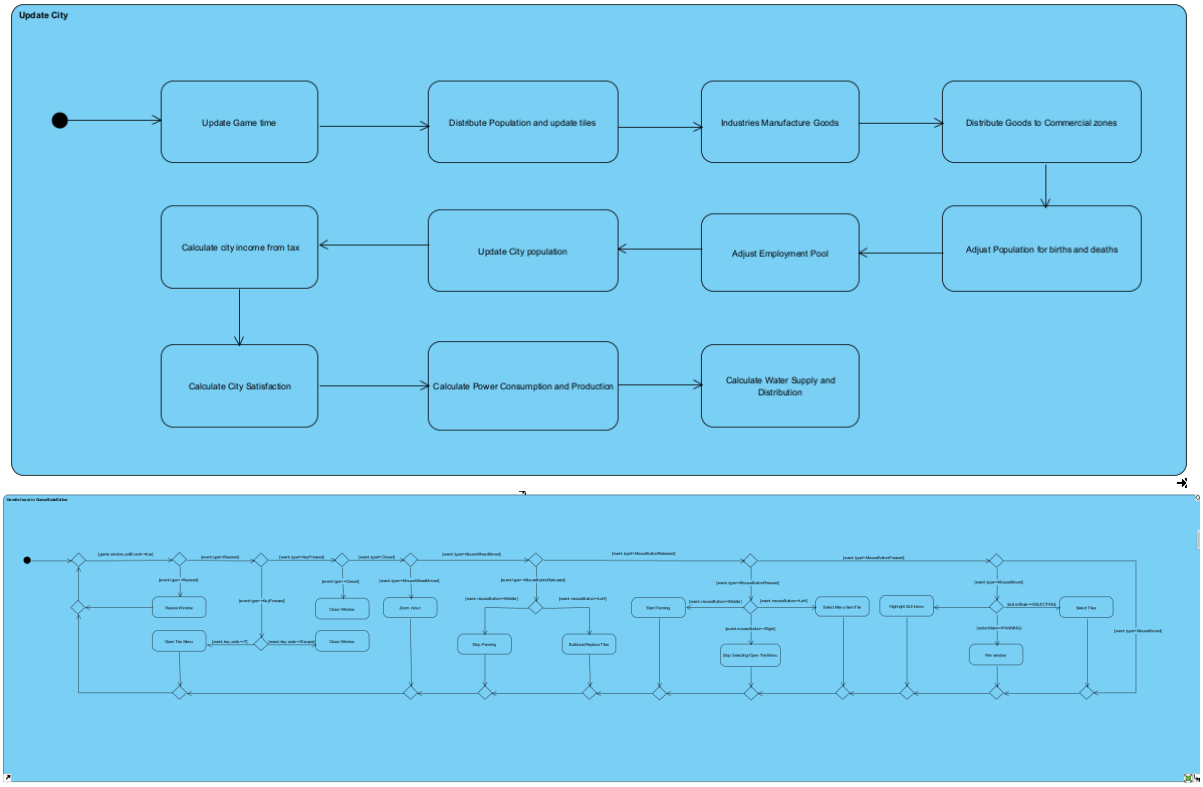




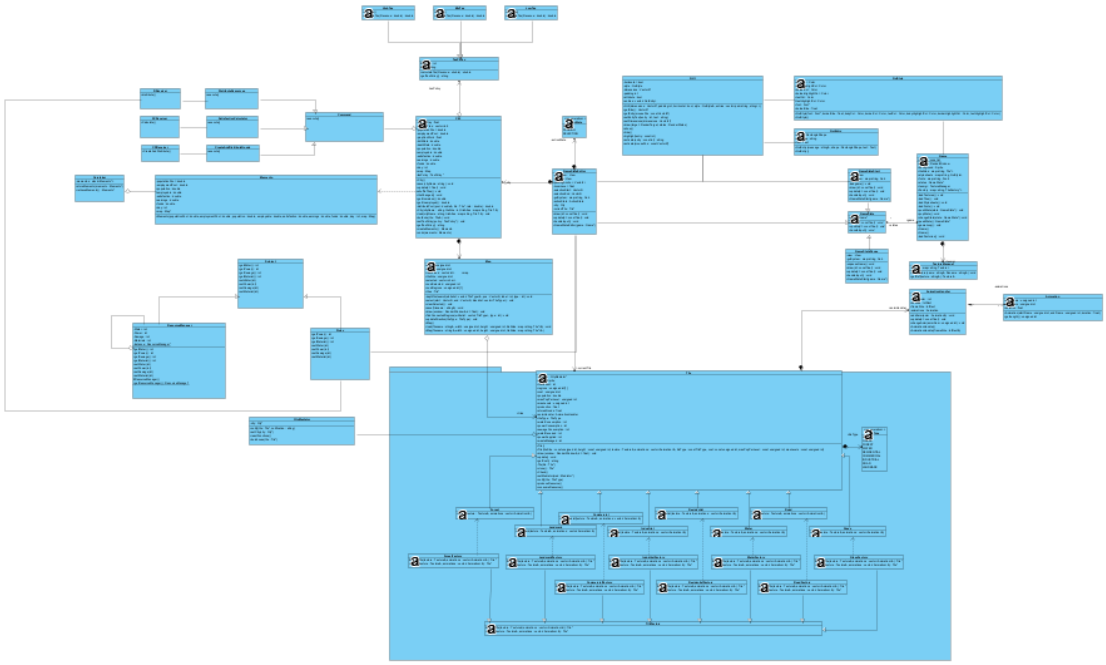








Class Diagram:



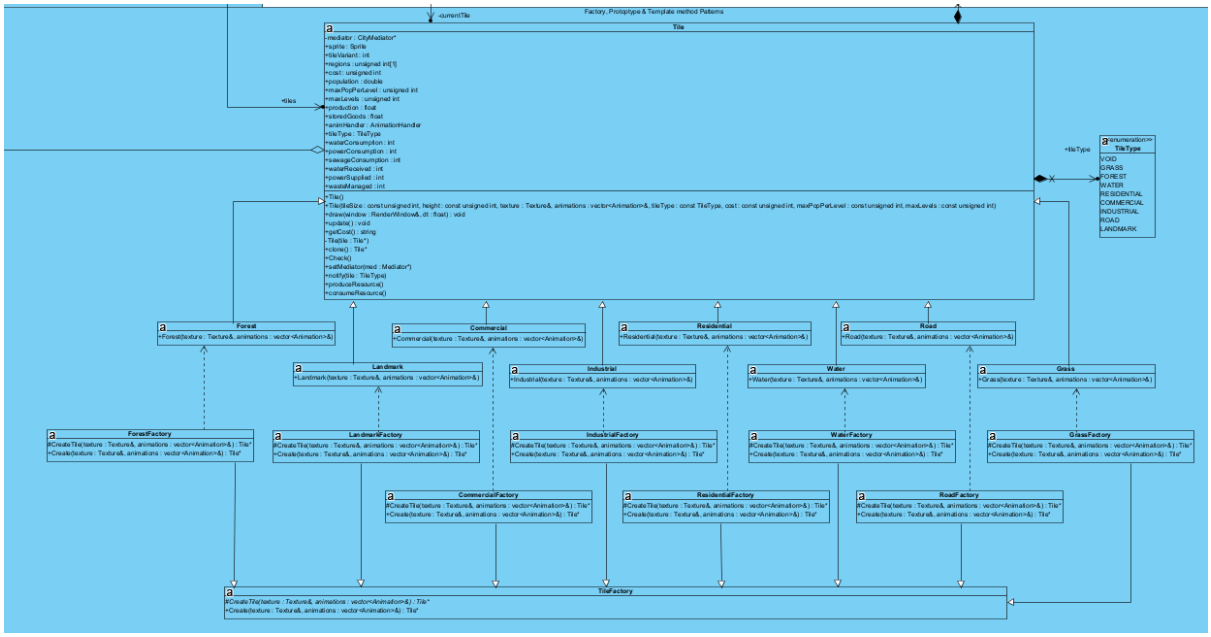


Figure 1: Factory, Prototype And template method

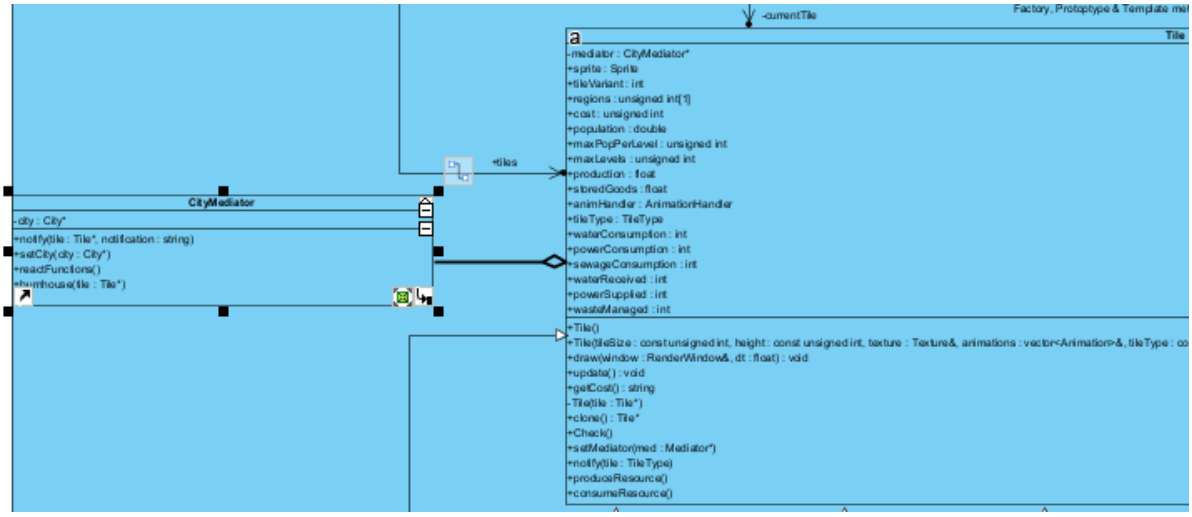


Figure 2: Observer and Mediator pattern

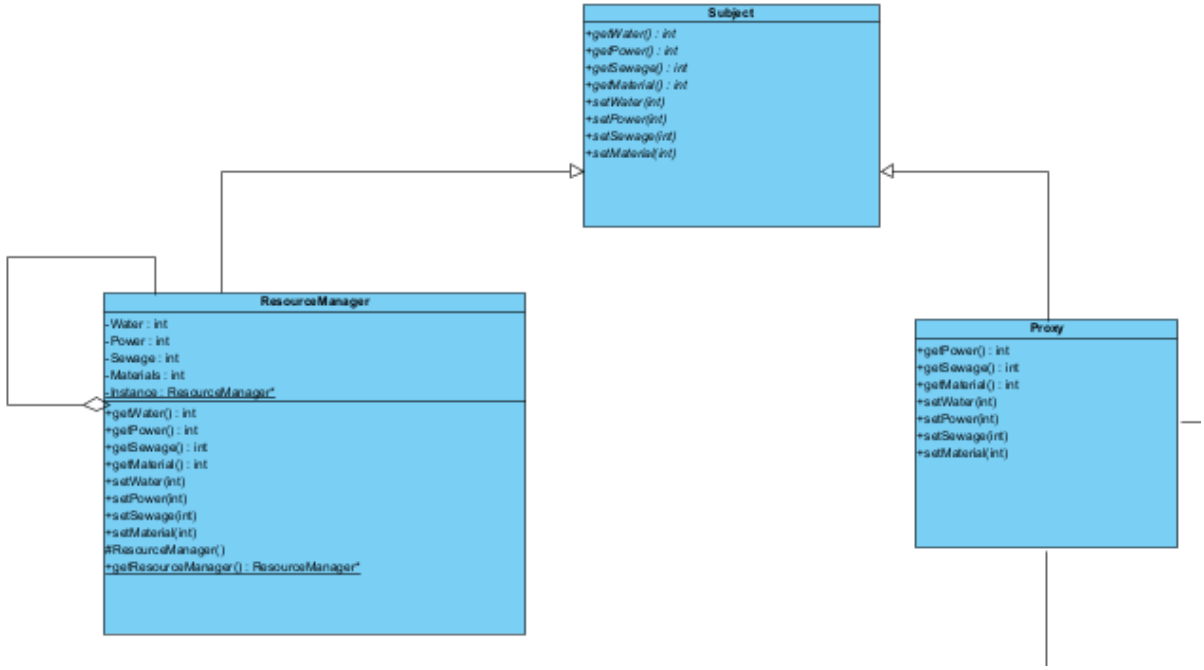


Figure 3: Singleton and Proxy Pattern

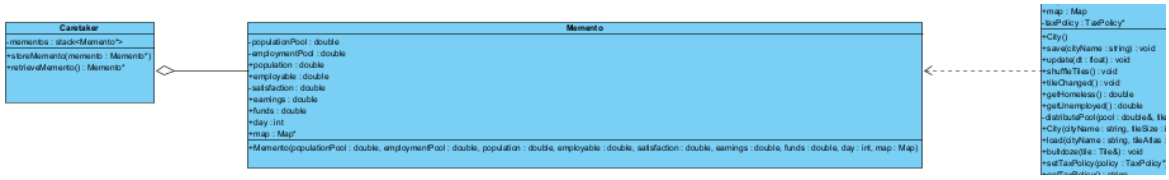


Figure 4: Memento Pattern

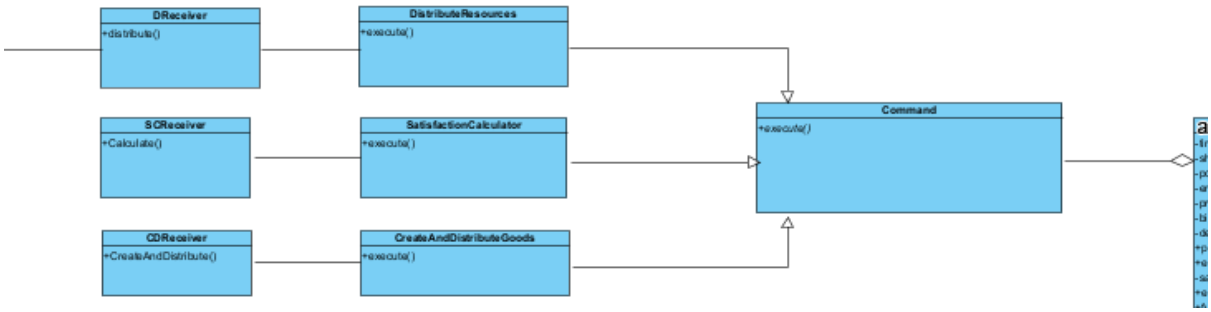


Figure 5: Command Pattern

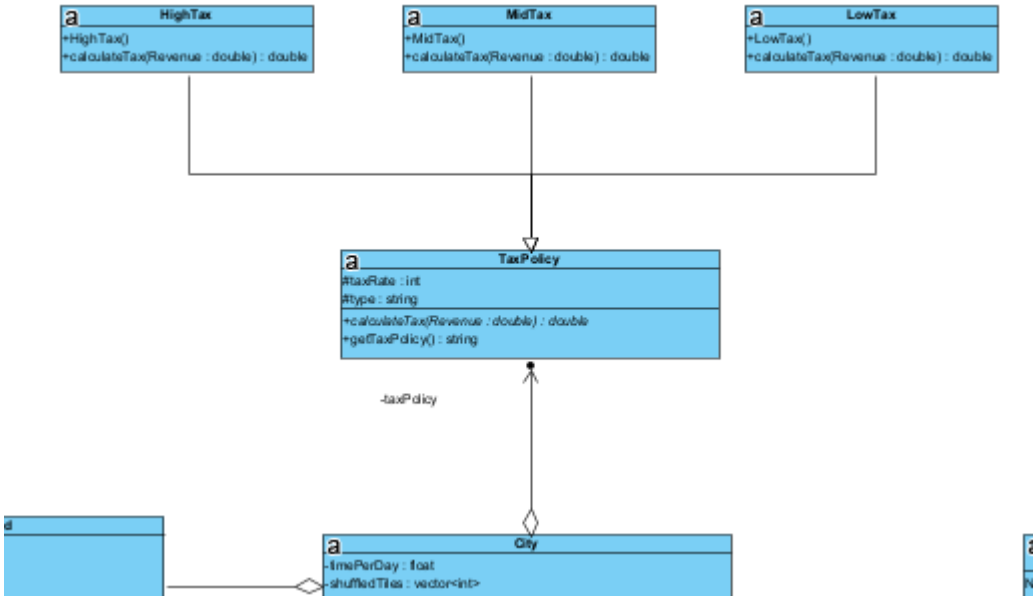


Figure 6: Strategy Pattern

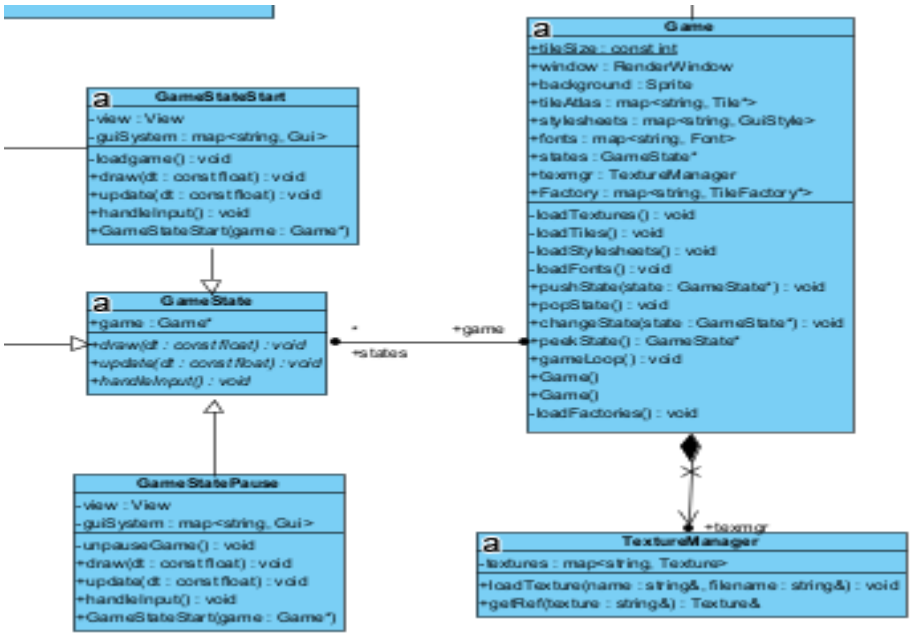


Figure 7: State Pattern

3. Implementation

3.1 Coding Standards and Best Practices

Modularization: Each component, such as buildings, utilities, and citizens, has dedicated classes and methods, ensuring a clear separation of concerns.

Naming Conventions: Consistent naming conventions follow the format PascalCase for classes and camelCase for methods and variables.

Documentation: All header files (e.g., Tile.h, TaxPolicy.h, City.h) have Doxygen comments for auto-generated API documentation.

3.2 Core Modules and Classes

Tile and TileFactory: Base class Tile represents city elements (e.g., roads, water plants) with subclasses like Residential, Commercial, and corresponding factories (ResidentialFactory, CommercialFactory) that streamline object creation.

Resource Management: ResourceManager ensures efficient resource allocation using a Singleton pattern, handling essential city resources such as water, electricity, and waste.

Citizen and Satisfaction Management: Citizen satisfaction is tracked through SatisfactionCalculator and managed by the Observer pattern, adapting to policy and infrastructure changes.

Commands and Utilities: Commands like DistributeResources and utilities like WaterPlant allow modular and reusable operations across the simulation.

3.3 Data Flow

Resource Allocation: The ResourceManager handles resource flow between production facilities (e.g., power plants) and consumer buildings (e.g., residential areas).

Government Policies: Government decisions, such as setting tax rates through TaxPolicy subclasses, influence citizen satisfaction and city funding.

3.4 Optimization Techniques

Lazy Loading: For certain resources and tiles, loaded on-demand to reduce initial loading times and memory usage.

Google Docs link

https://docs.google.com/document/d/13O-WcW3G7TEObcm24XkvyR_JHSZCENM-Rt4Gxgs1qqY/edit?usp=sharing