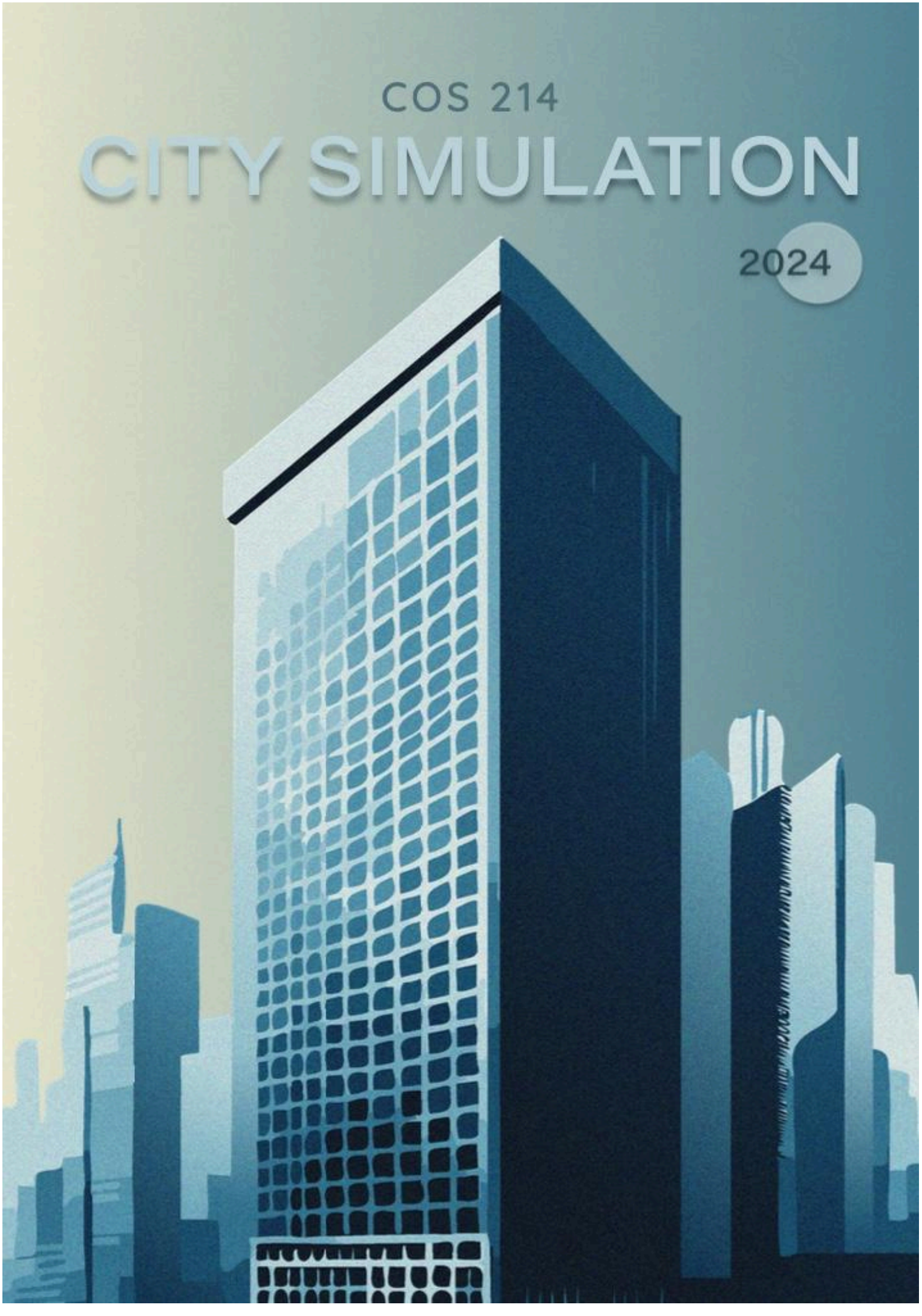
COS 214

CITY SIMULATION

2024



COS 214 Project Report

Group name: idk

(Nickname: Arrays of Sunshine)

u23587832 Hannah Koorbanally u23673941 Louise Bruwer u23530996 Kiara Hodgson
u23605376 Brendan du Plooy u21439631 Euan Botha u21437883 Nolan Kühn u23782219 Ruan le Roux

GITHUB REPOSITORY:
https://github.com/COS214-Project-2024/idk

LINK TO DOCUMENT:
https://docs.google.com/document/d/1w2eHsOXzge7_VIquZCeK1Lx0pm3v5aoK9sUKoSMN8PU/edit?usp=sharing

Table of Contents

Section 1 : Research Brief

Urban Development

Urban development transforms cities to drive economic & social growth. It focuses on improving living standards while addressing environmental concerns that arise from high population density. Key goals include enhancing quality of life through accessible services, housing & efficient transportation. Government, S. (2024)

Key Principles for City Management

We found particularly relevant the principles of Acquiring sufficient funding (which includes financing running costs, as well as levying taxes) as well as that of Accountable governance, as per Nusca, A. (2024). Our government will be made to be as efficient and have as positive of an impact on citizens as possible.

Role of Various Components In The City

Buildings

Buildings serve multiple roles in a city. Residential properties, for example, house citizens, influencing their satisfaction and contributing to economic development. While each building type has a primary function (e.g., providing space for business, production, or residence), it also impacts the city's economy by contributing to GDP (Architecture Courses, 2024).

Utilities

According to the Council of Scientific and Industrial Research (CSIR), residential utilities support domestic needs by providing resources like electricity and materials, while collective services handle waste management and sewer systems. Maintaining these utilities is crucial for citizen satisfaction and efficient public services.

Transportation

Many factors can influence the method of transport citizens take. These factors include traffic, cost of vehicles/travel & the time taken to travel. The time taken for travel, in particular, may impact the citizen satisfaction of the city. Both private and public transportation varies depending on the given scenario. Factors such as traffic, cost of vehicle for public and private ownership and the time of travel influence which method is most efficient to travel across the city. Transit, P. (2021)

The transportation component is used within our system to influence factors such as citizen satisfaction, impact from a given scenario (car accident) and the generation of income from public modes of transport (impact on Bank and Government). The transportation section therefore has an economic, functional and qualitative role based on the effective management of the city.

Citizens

The concept of “citizen-centricity” came up in our research. This concerns the prioritising of citizens’ demands in the designing and delivery stages of public services (Berntzen et al., 2016; Kamalia & Nor, 2017)

Government

From our research, the government manages all aspects of the city. Examples are managing citizen satisfaction and welfare by ensuring residents have services & adequate housing, as well as Resource Management, where local governments are key managers of essential services like water & energy. Hoeflich de Duque, S. (2023)

Resources

Urban areas depend heavily on resources to support their populations and activities. However, rapid urbanisation drives up resource consumption, complicating efforts to uphold sustainability principles. Therefore, effective resource management is essential for achieving sustainable urban development, focusing on maximising resource use while minimising waste and environmental impact. Zucaro, A., Maselli, G. and Ulgiati, S. (2021)

Taxes

Taxes play a key role in how residents feel about their government and their own finances. Changes in tax rates affect people's bank accounts and their satisfaction with public services. There's often a gap between what citizens want from their local government and what they're willing to pay in taxes. When people see improvements in services that match their tax dollars, it boosts their overall happiness and financial situation. Glaser, M. A., & Hildreth, W. B. (1999)

City Growth

Population growth significantly impacts taxes by increasing the demand for essential services like housing. As cities expand, they may need to raise tax rates or introduce new taxes to fund these services. On the other hand, a larger population can also broaden the tax base, potentially allowing for lower rates. Bahl, R., Holland, D. & Linn, J., (2015)

Influence on design

The research guided our design choices by emphasizing how core city components impact both functionality and quality of life, leading to assumptions and design decisions that align with urban development goals. For instance, we assumed that each city component (e.g., buildings, utilities, transportation) contributes to citizen satisfaction and economic health. This guided our decision to model these components with flexibility in mind, allowing for efficient changes in response to evolving city dynamics.

Each pattern helped maintain modularity and adaptability, supporting efficient management of city functions, resources, and services as per the principles outlined in the research brief.

Assumptions and decisions

- Citizen satisfaction is a key metric; it is directly influenced by cost of living, employment rate and government decisions like taxation.
- Utilities are essential services (e.g., electricity, water, waste management) assumed to operate continuously, with failure rates simulated based on resource availability and demand.
- The government component acts as a central mediator, managing things like taxes, public services, and resource distribution across the city.
- Tax strategies are employed based on economic and population growth
- Randomised situations like accidents can occur, with response forces then assisting. The outcome of assistance is, however, not always perfect. This affects citizen satisfaction.
- Citizens and government both have a bank account with a balance. The player of the simulation operates from the government money and may use a cheatcode to increase this money if desired.

References

- Government, S. (2024) *Urban development*, *Portal GOV.SI*. Available at: https://www.gov.si/en/policies/environment-and-spatial-planning/prostor-2/urban-development/ (Accessed: 29 October 2024).

- Korppoo, K. (2015) *Designing game analytics for a city-builder game*, *Trepo*. Available at: https://trepo.tuni.fi/handle/10024/97480 (Accessed: 22 October 2024).

- Nusca, A. (2024) Four principles of Effective City Management, ZDNET. Available at: https://www.zdnet.com/article/four-principles-of-effective-city-management/ (Accessed: 22 October 2024).

- Villagomez, E. (2023) *S101s - describing building types: Why they matter*, *Spacing Vancouver*. Available at: https://spacing.ca/vancouver/2023/09/11/s101s-describing-building-types-why-they-matter/ (Accessed: 22 October 2024).

- Transit, P. (2021) Public vs. private transportation: Advantages & Disadvantages, Safe and Reliable Bus Charter Rentals in Ventura County. Available at: http://www.pegasustransit.com/post/public-vs-private-transportation-advantages-disadvantages (Accessed: 28 October 2024).

- Hoeflich de Duque, S. (2023) *Sustainable development through local governments*, *Urbanet*. Available at: https://www.urbanet.info/local-governments-sustainable-urbanisation/ (Accessed: 29 October 2024).

- Zucaro, A., Maselli, G. and Ulgiati, S. (2021) *Insights in Urban Resource Management: A comprehensive understanding of unexplored patterns*, *Frontiers*. Available at: https://www.frontiersin.org/journals/sustainable-cities/articles/10.3389/frsc.2021.807735/full (Accessed: 29 October 2024).

- Glaser, M. A., & Hildreth, W. B. (1999). Service Delivery Satisfaction and Willingness to Pay Taxes: Citizen Recognition of Local Government Performance. *Public Productivity & Management Review*, *23*(1), 48–67. https://doi.org/10.2307/3380792

- Bahl, R., Holland, D. & Linn, J., (2015). *Urban growth and local taxes in less developed countries*. Available at: https://scholarspace.manoa.hawaii.edu/server/api/core/bitstreams/4b852e95-4d3b-4c3c-a9ba-e166f883d452/content [Accessed 29 Oct. 2024]
-
- Architecture Courses (2024) *Building Types*. Available at: https://www.architecturecourses.org/learn/building-types (Accessed: 2 November 2024)

Section 2: Design Pattern Application Report

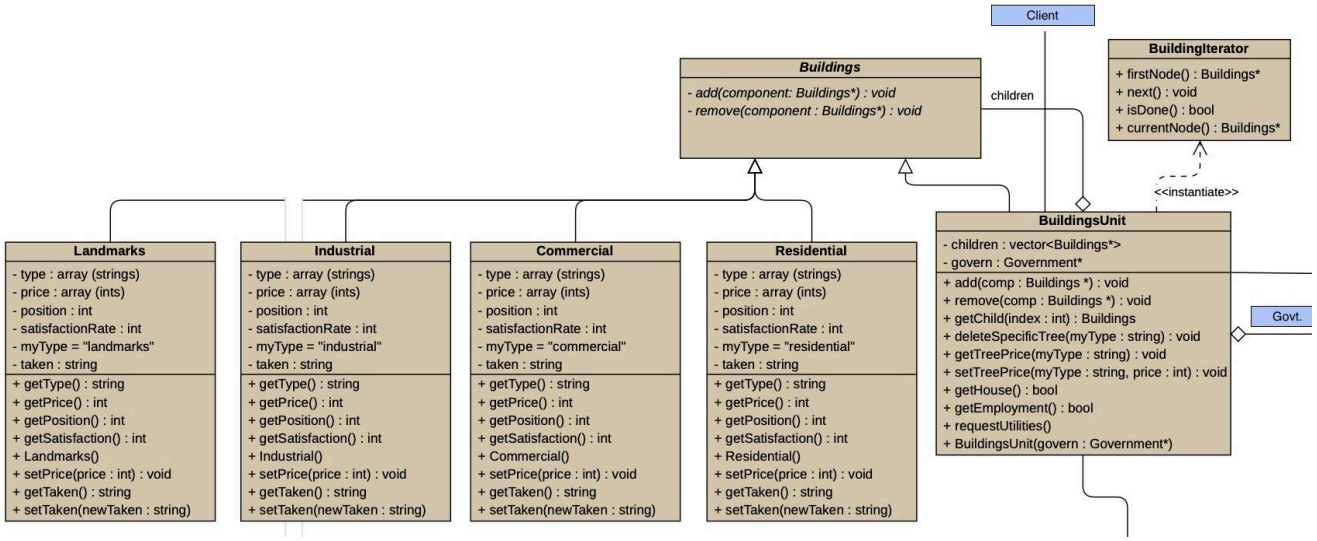
Iterator

The Iterator Design Pattern provides a structured way to traverse elements of an aggregate object sequentially without exposing its underlying structure. This pattern is particularly useful for components that need to access or process a sequence of elements, such as methods, fields, or functions, without needing to understand how those elements are organised internally.

We used this pattern for our **Buildings** component to efficiently locate and access specific types of building units.

In this context, the Iterator pattern is implemented within our BuildingIterator class, providing the ability to navigate through a list of BuildingUnit objects, thus enabling the retrieval of a specific unit based on set requirements.

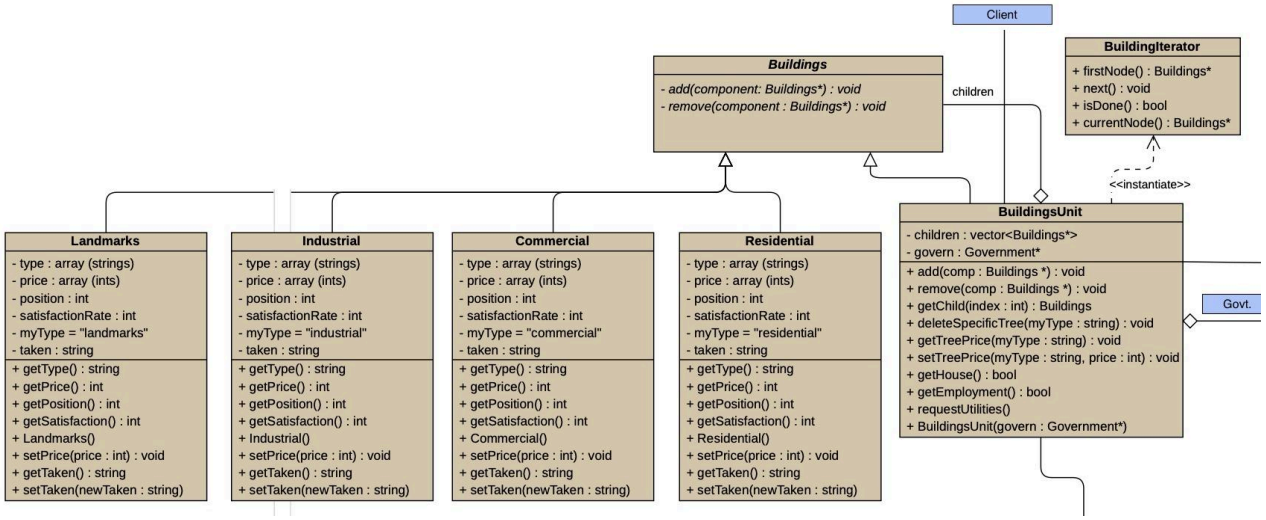
This approach makes it straightforward for other classes or functions to find and retrieve the necessary BuildingUnit object, ensuring that the process remains clear and unobtrusive, without adding complexity to the existing code structure.



Composite

The Composite Design Pattern organises objects into tree-like structures to represent part-whole hierarchies, allowing clients to treat single objects and collections of objects uniformly. This pattern is particularly useful for managing entities that share common behaviours while varying in their specific implementations, making it an effective approach for modular systems with diverse components.

In our system, the Composite Design Pattern was chosen to structure the Buildings component, enabling both individual building types and groups of buildings to be handled consistently. By applying this pattern, different building types - Residential, Industrial, Commercial, and Landmarks—can function as standalone units or as part of larger structures, aligning with the varied needs of citizens. Here, BuildingUnit serves as the composite, with each building type acting as a leaf to provide some additional specific functions. The Buildings component offers **general operations like adding or removing buildings**, which the BuildingUnit hierarchy implements to ensure **consistent interaction** across the system. This setup allows each building type to integrate seamlessly with other parts of the city simulation, including government functions and banking, without requiring special handling.

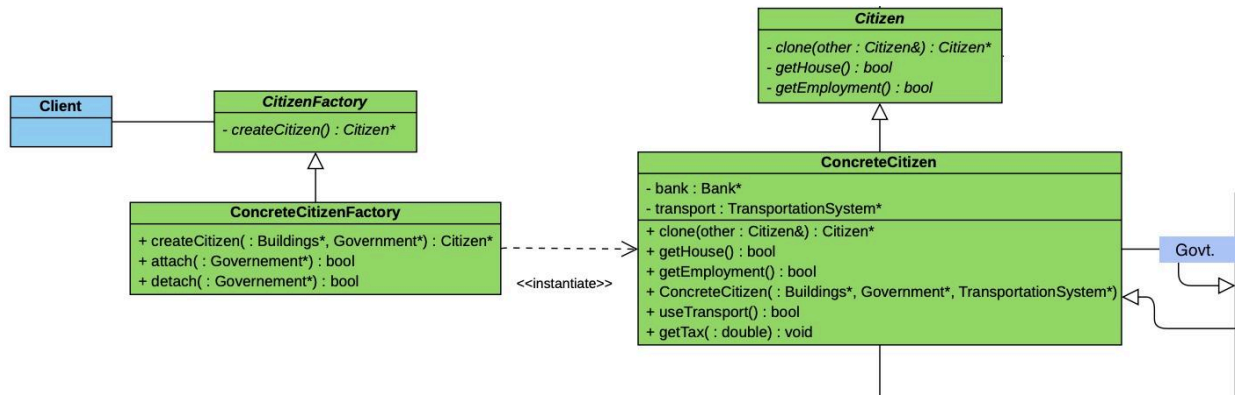


Factory Method

The Factory Method pattern provides an interface for creating an object but allows subclasses to decide which specific class to instantiate. This enables classes to delegate instantiation to subclasses, promoting flexibility by decoupling the client from the concrete products being created

In this system, a Factory object is instantiated in the main, responsible for continuously creating Citizen objects without additional parameters. Once a citizen is created, it can be copied repeatedly to represent scenarios like twins. Each Citizen can then request housing or employment, which triggers the Buildings unit to check a list for availability, where available entries are marked as taken = false. When a match is found, the status updates to taken = true, and the system notifies the citizen that housing or employment has been secured.

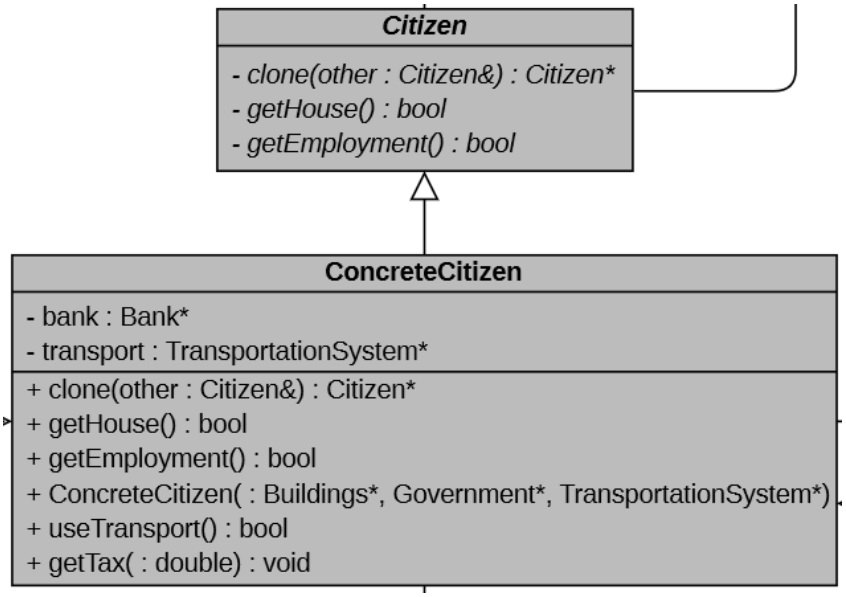
The Factory Method pattern simplifies this process by organising citizen creation in a centralised and manageable way, decoupling Citizen creation from specific details and allowing Buildings to handle housing and job assignments. This separation reduces the complexity in the main application logic, enabling efficient resource allocation and tracking, while keeping object life cycles consistent and orderly across the system.



Prototype

Seeing that all citizens share the same fundamental properties such as a place of residence as well as making use of resources such as transportation, it would be beneficial to make use of the Prototype design pattern to ensure that the citizens created come from a prototypical instance. However, along with the use of the Factory Method design pattern, the properties instantiated for an existing Citizen can be copied onto the newly created citizen, emulating the concept of a family where family members would share the same properties such as the same house.

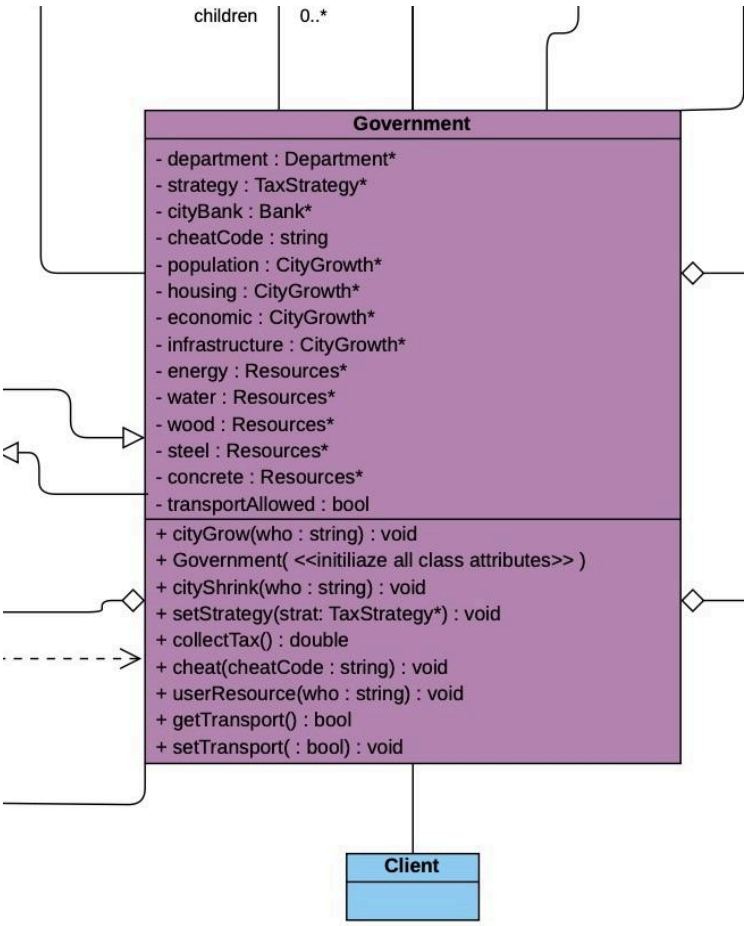
In the creation of new citizens, it would be timely and un-effective to create citizens that derive from each other from scratch through another design pattern, however, with the Prototype design pattern, one is able to ensure that uniform creation is possible. Seeing that all citizens share the same fundamental makeup (their members: to make use of a bank, as well as a form of transport), making a copy of a prototype and its uniquely populated members (through the Factory Method) would be most efficient in the creation of citizen objects that share the same properties. This makes use of DRY coding, to ensure that code is not repeated unnecessarily.



Mediator

Define an object that encapsulates the interactions between a set of objects, promoting loose coupling by preventing direct references and allowing interactions to vary independently.

In our system, the Mediator design pattern centralizes communication by designating the Government as the intermediary through which all components interact. This pattern mirrors real-world governance, where processes must pass through government oversight to ensure all activities and members are tracked and recorded. The Government acts as the central channel, managing the communication between components to prevent direct references among them. This approach maintains loose coupling, as components do not interact independently but instead communicate exclusively through the Government, allowing it to monitor all system interactions. This centralized structure ensures the Government's awareness and oversight of all processes affecting the city, fulfilling its role in coordinating and tracking interactions across the system.

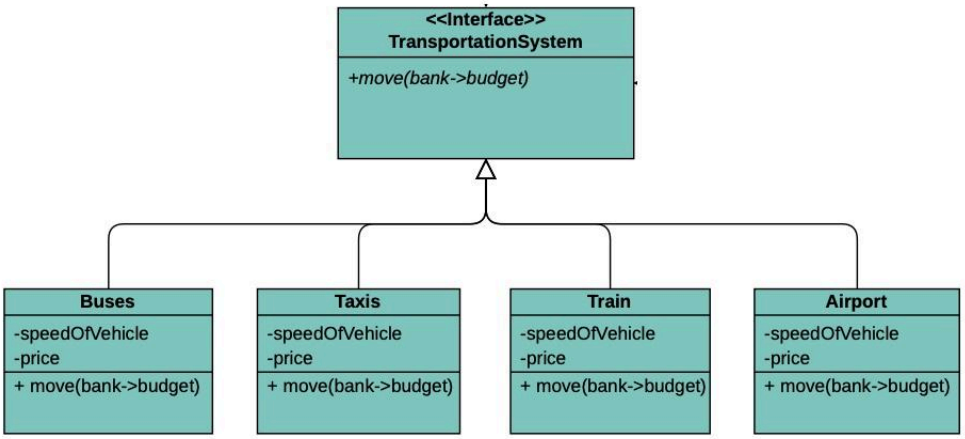


Template Method (done)

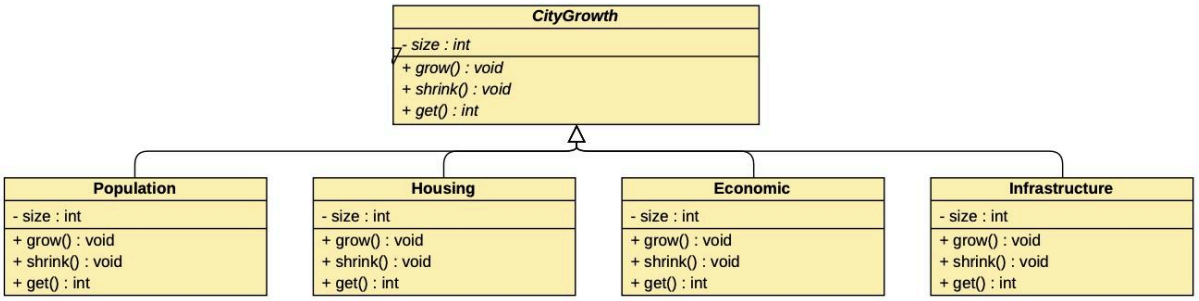
Defines the skeleton of an algorithm in operation by deferring some steps to its subclasses which will then redefine the steps of the algorithm without changing it structure

For the transport component, this approach works best because all the different modes of transport share a common method with similar implementations. Defining a primitive operation in a common abstract class, therefore, enhances code reuse and reduces duplication. Additionally, if a mode of transport needs to be added or removed in the future, this will be much easier with the template method implemented. This reasoning also applies to the city growth component. Different factors affect city growth, but they all have similar methods with similar implementations. Therefore implementing the template method is most efficient.

Transport



City Growth

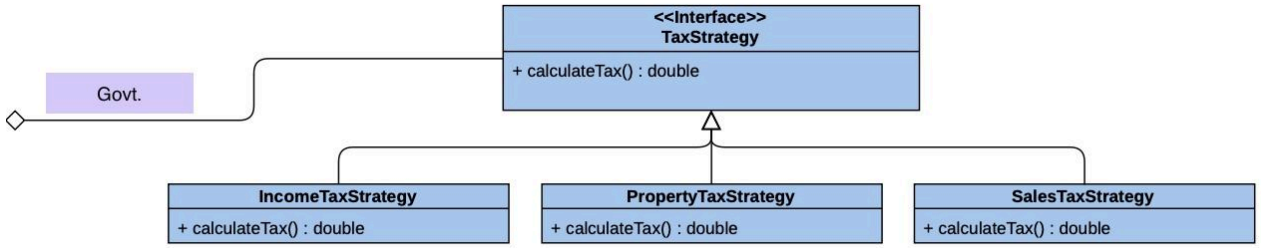


Strategy

The Strategy pattern defines a group of related algorithms, encapsulates each one, and makes them interchangeable. This allows the algorithm to vary independently from the clients using it, providing flexibility and reducing dependencies on specific implementations.

In the tax strategy component, the Strategy pattern allows the government to manage various tax calculations through an abstract taxStrategy class. Three derived classes - propertyTaxStrategy, incomeTaxStrategy, and salesTaxStrategy - each implement the calculateTax() method, enabling different tax types to be calculated without modifying government.

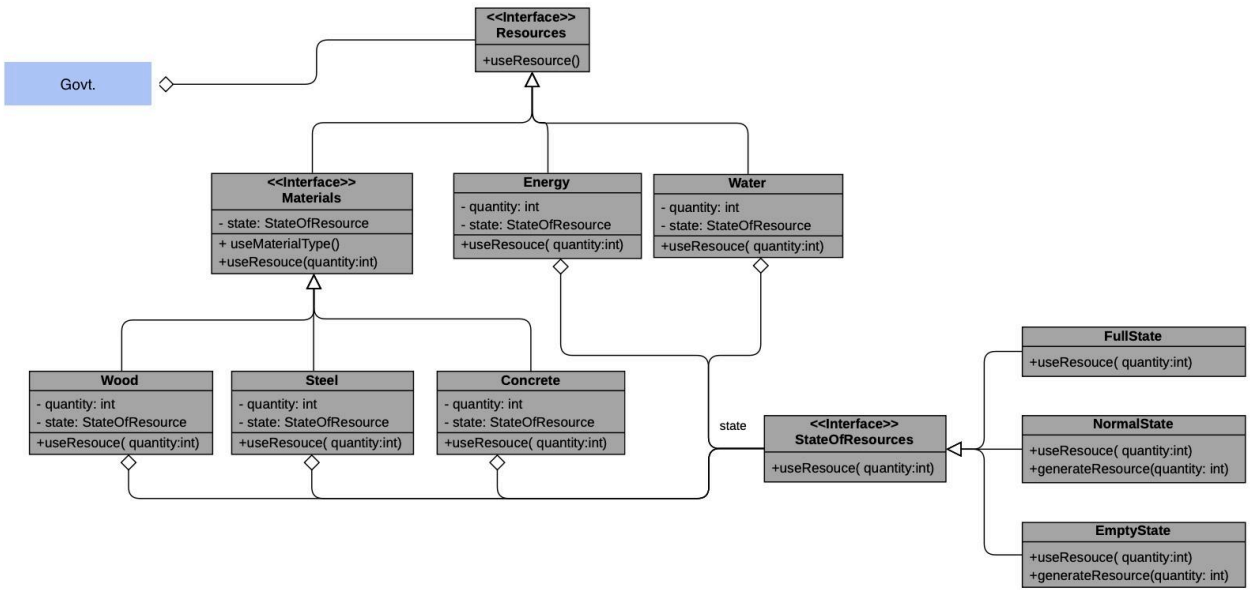
This approach simplifies the design by removing complex conditionals and enables the government to set or change tax strategies dynamically via setStrategy(). Encapsulating each tax calculation ensures robustness, as any changes or additions to tax algorithms don't impact the government class, keeping the system flexible and easier to maintain.



State

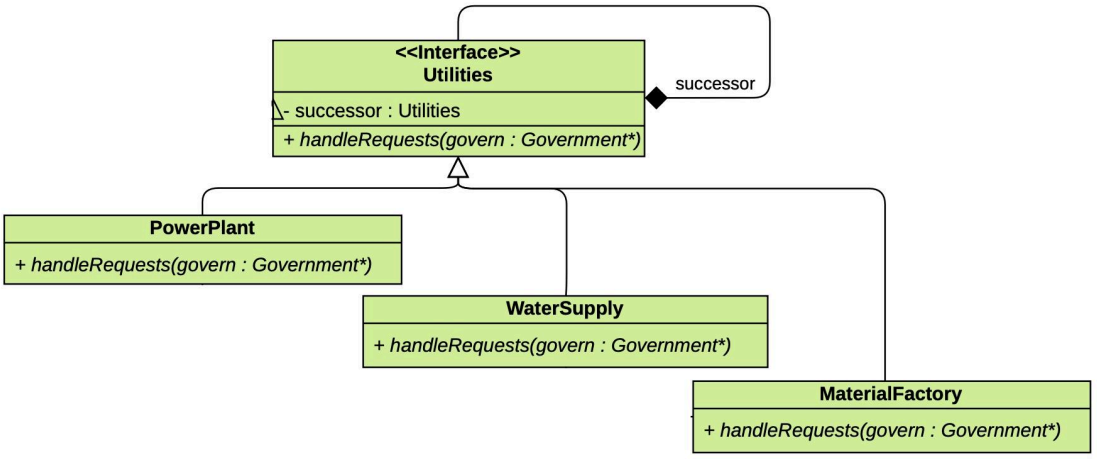
The State pattern allows an object to alter its behaviour when its internal state changes, making the object appear to change its class. This pattern was chosen because it enables resources to dynamically modify their behaviour based on their current state. For resources, particularly in managing their usage, the State pattern allows interactions to adapt according to the quantity available. Since the amount of a resource affects how other classes utilise it, this pattern enables resources to adjust behaviour in response to their quantity state.

This pattern was implemented to manage the resource flow more effectively: depending on the quantity of a specific resource, certain methods—such as generateResource() and useResource()—are called to meet the demands of other components, like Utilities. When a utility requests a specific quantity of a resource, the request is processed by the Government, which in turn requests the exact quantity from the resource. Based on availability, different methods are triggered: if the resource quantity is insufficient, generateResource() is called to increase stock, followed by useResource() to utilise the available amount. If enough of the resource is available, generateResource() is bypassed. The various states of full, empty, or normal dictate how a resource behaves, making the state diagram an appropriate structure for this functionality.



Chain of Responsibility

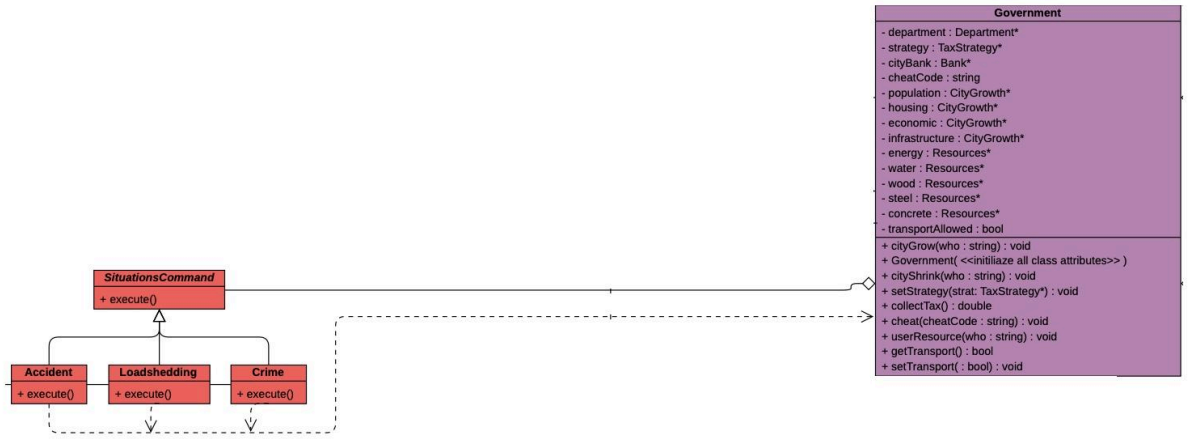
This design pattern helps avoid tightly coupling the sender of a request to its receiver by allowing multiple objects to handle the request, passing it along a chain until one takes responsibility. In our city system, this pattern is particularly well-suited because the Utilities class must supply each BuildingUnit with resources such as power and water. By decoupling the sender (BuildingUnit) from the receiver (the specific utility), each resource request can be routed along a chain, allowing only the utility responsible for generating that resource to handle the request. This approach supports modular design and efficient processing of varied resource requests. Each BuildingUnit needs to be equipped with specific resources, and this is achieved by calling the responsible utility, which manages both the creation and distribution of the requested resource. The decoupling achieved with this pattern means that each request is directed only to the relevant utility, ensuring that each resource is provided by the correct handler. This allows the system to manage multiple requests independently and efficiently, with each utility taking responsibility only for the resources it controls.



Command

The Command pattern encapsulates a request as an object, allowing clients to be parameterized with different requests, enabling queuing or logging of requests, and supporting undoable operations. By wrapping requests in command objects, this pattern separates the object that initiates the operation from the one that performs it, offering greater flexibility and control over when and how requests are executed.

The Command pattern enhances the system's flexibility by allowing the Government to issue different commands - such as handling accidents, managing loadshedding, or addressing crime - through a common interface. This setup allows for easier modification of operations in response to new requirements or changes in command handling. Encapsulating each situation as a command also allows the government to log, queue, or even undo specific operations if needed. By decoupling the Government from the specifics of each action, the design keeps the code modular, making it easy to add new commands in the future while maintaining clear boundaries between request initiation and execution.



Section 3: GitHub Branching

We decided to use **feature branching** as our GitHub branching strategy to streamline collaboration in our team of seven. Each feature branch allowed team members to work on specific aspects of the city simulation project independently without affecting the main branch. This setup helped us avoid conflicts and ensured that each feature was fully developed and tested before being merged and integrated. Additionally, by keeping our main branch stable, we could ensure continuous integration and periodic reviews, where each member's code underwent peer review before merging. This approach helped maintain high code quality and a stable project structure as we built out the simulation's complex interactions and dependencies.