



FINAL PROJECT REPORT: Eco City Builder

Cohort C - Group 10

Amy Addo - 74012028

Lisa Baer - 47592028

Joram Hanson - 25432028

Tetteh Quayenortey - 56612028

Ashesi University

Object-Oriented Programming

Lecturer - Dr Daniel Addo

Faculty Intern- John Mensah

9th December 2025

Background and Introduction

In recent human history, urban areas have become significant contributors to resource consumption and environmental degradation, accounting for approximately 75% of global resource consumption, 65% of energy use, and over 70% of carbon emissions as of 2024 (Figen, 2024). He adds that the rapid pace of urbanisation, particularly in developing countries, has exacerbated issues in these urban areas, such as housing shortages, inadequate infrastructure, and increased vulnerability to climate change. From this work alone, we can begin to see the effects cities have on our environment and way of life. Rashed further confirms this observation, stating the 20216 World Cities Report revealed that “the current urbanisation model is unsustainable in many respects, puts many people at risk, creates unnecessary costs, negatively affects the environment (2023).

This is where the concept of smart cities comes in. Eco-City Builder is an educational game that aims to address this issue by encouraging players to build a sustainable city with minimal impact on its ecosystem. We aim to give users the chance to explore various effects on a city and to leave a lasting impression, where users begin to implement positive, eco-friendly concepts learned in their lives. This text-based simulation game models the creation and management of a city, preferably one that is environmentally friendly and sustainable. Being created in Java, we have given users the ability to control various buildings within the city, thereby impacting measured metrics in the game.

By utilising game logic and various object-oriented programming concepts such as inheritance, we created a program consisting of a “Building” superclass and several subclasses which have varying effects on the city’s statistics. These subclasses include:

- Power plants (Renewable and Non-renewable)
- Public Services (City hall, Fire station etc.)
- Parks
- Residencies
- Shops (Malls and Retail Centres)
- Schools
- Public Transport
- Factories

Each subclass contains methods that modify key city metrics, such as energy balance, pollution levels, happiness, and overall well-being. The program continuously updates these metrics through a game loop, where users can add or remove buildings and track the effects and changes in their city using a scoreboard.

Additionally, users will have access to a list of achievements and experiences to earn from, serving as motivation to create various combinations within the game. Through this project, our primary goal is to develop an environmentally conscious game that empowers users to maintain a less polluted environment while maximising the happiness and well-being of the city’s residents.

We initially designed the game as a text-based format to allow users to interact with it. However, we later decided to implement a graphical user interface (GUI) to enhance its appeal to gamers. This change also helps players visualize the message our game aims to convey.

“Building” Class

The Buildings class serves as the abstract superclass for all building types in the Eco-City Builder simulation. It defines the common structure, attributes, and behaviors shared by every building in the game. Each building contributes to the city’s overall sustainability through four key metrics: “energyBalance”, “pollution”, “happiness”, and “wellBeing”. These fields represent the extent a building impacts the environment and lives of its residents. The class also stores a “buildingName” for clear and easy subclass identification. The Buildings class provides getter methods for all metrics for the City class to access the values when adding or removing buildings. Additionally, it includes a toString() method that formats and displays the effects of a building in a way that is comprehensible. This helped us to debug and trace various outcomes. By making Buildings an abstract class, only specified subclasses (like SolarPowerPlant, Factory etc.) can be instantiated. This helped us to create buildings with consistent structure, whilst allowing each subclass to define its own unique values. Overall, the Buildings class acts as our template for building the city.

“City” Class

The City class contains key methods and functions for the city. It tracks and updates the total city’s metrics as totalPollution, totalEnergyBalance, totalHappiness, and totalWellBeing. These values change automatically whenever buildings are added or removed. The City class also stores the number of units for each building type using the BuildingCount property. This prevents users from exceeding the maximum number of buildings for each building type. When a building is added, the metrics for that building, defined in the Buildings superclass, are multiplied by the number of units and applied to the

city's totals. Removing a building does the opposite. Other methods, such as `remainingFor()` and `getCount()`, helped us to validate the metrics and to not exceed possible limits. Overall, the `City` class helped us to encapsulate the calculations in the city and allowed for scalability.

“Game” Class

The `Game` class is the main part of the `Eco City Builder` program. It controls how the game runs and how the player interacts with it. This class starts the game by creating a `City` object to track all buildings and statistics, and uses a `Scanner` to get input from the player. The game runs in a loop that shows a menu of building options. The player can add or remove buildings, and the class updates the city's energy, pollution, happiness, and well-being accordingly. It also checks that the player enters valid numbers and prevents exceeding building limits. After each action, the class prints the effects of the buildings and shows the current totals. When the player finishes, the class calculates percentage scores for eco-health, energy, happiness, and well-being, and then determines a final city ending based on these scores. This ending reflects the balance and success of the player's city. Overall, the `Game` class manages the game's flow, tracks player actions, and provides feedback and outcomes.

Subclasses

Each subclass inherits from the `Buildings` superclass and defines its own effects. Taking from real life effects, each building type contributes to the city in a different way. For instance, `Green Power Plants` affect energy levels but not pollution levels by much and `Coal Power Plants` greatly affects power output but significantly increase pollution. Each class `Schools`, `Shops`, `Factories`, `Public Transport`, `Public Services`, `Green Power Plant`, `Coal Power Plant`, `Park` and `Residences` have various impacts and reflect the environmental and social consequences of urban planning decisions. Their metric effects are explicitly

stated in their constructors for each subclass to change the city's sustainability profile. Within this class we utilised inheritance to realistically model how diverse building types affect the statistics of a city.

Important Methods

Command Line Interface

The following methods comprise the Game class, and their functionalities are explained below.

1. `public static void main(String[] args)`

Purpose: This is the entry point of the game and manages the entire gameplay flow.

Responsibilities:

- Creates a Scanner for reading user input.
- Instantiates a City object that tracks buildings and city statistics.
- Runs the main game loop, allowing the player to:
 - Choose a building type.
 - Add or remove buildings.
 - View effects on energy, pollution, happiness, and well-being.
- Ends the game by calling `finish(city)` and closes the scanner.

Key Features:

Input validation, menu display, and coordination of building effects.

2. public static int getValidCount(Scanner sc, int min, int max, String message)

Purpose:

Ensures the player enters a valid number of buildings to add or remove.

Parameters:

- Scanner sc: Reads input from the user.
- int min: Minimum allowed value (usually 1).
- int max: Maximum allowed value (either the remaining slots or current count).
- String message: Prompt displayed to the player.

Functionality:

Loops until a valid integer is entered, rejecting non-integer input and numbers outside the valid range.

Returns:

A validated integer representing the quantity to add or remove.

3. public static void printAddEffects(Buildings b, int qty)

Purpose:

Displays the effect of adding buildings on city metrics.

Parameters:

- Buildings b: The building being added.

- int qty: Number of buildings added.

Functionality:

Multiplies the buildings individual effects by qty and prints:

- Energy balance

- Pollution (minimum 0)

- Happiness

- Well-being

4. public static void printRemoveEffects(Buildings b, int qty)

Purpose:

Displays the effect of removing buildings on city metrics.

Parameters:

Same as printAddEffects.

Functionality:

Prints the negative effects of removing buildings, ensuring that pollution does not fall below 0.

5. public static void printCityTotals(City city)

Purpose:

Displays current overall city statistics.

Parameters:

-City city: The city object tracking totals.

Functionality:

Retrieves total pollution, energy, happiness, and well-being from the city and displays these totals in a formatted output. Pollution is clamped to 0 if it is negative.

6. public static void finish(City city)

Purpose:

Calculates final percentages and determines the city's closing narrative.

Parameters:

-City city

Functionality:

Retrieves raw totals from the city, clamps pollution to non-negative values, and converts raw totals into percentages using a fixed target value of 30. It utilizes `clampPercent()` to ensure percentages remain within the 0–100% range. It prints a final report of raw values and percentages and determines the city's ending based on the percentage combinations:

- UTOPIA: Best-case scenario; a clean, happy, balanced city.

- ECO PARADISE: A strong city with good statistics.
- JOYFUL CHAOS: Happy citizens despite environmental issues.
- INDUSTRIAL COLLAPSE: Pollution and energy shortages dominate.
- EMPTY SHELL: A clean city but with unhappy citizens.
- TECHNOCRATIC STATE: Highly efficient city with low happiness/well-being.
- BALANCED REALITY: Average or mixed outcomes.

7. private static double clampPercent(double v)

Purpose:

Ensures percentage values remain within the range of 0–100%.

Parameters:

- double v: It contains the calculated percentage.

Functionality:

Returns 0 if v is less than 0, returns 100 if v is greater than 100, and returns v unchanged if it is within the valid range. This method is used by finish() when computing final percentages.

These totals are used by the Game class to calculate percentages, display the city status, and determine the final city ending.

Overall Summary

The Game class serves as the central controller of the Eco City Builder game. It manages user input and the game loop, coordinates adding and removing buildings and their effects, displays live updates of city statistics, calculates final percentages, and determines the city ending. It utilizes helper methods for input validation, printing effects, and clamping values. The Game class does not handle the actual building data; that is managed by the City class and Buildings subclasses. The Game acts as the interface between the player and the simulation.

City Class Overview

Constructor: City()

This constructor initializes a new City object by setting the maximum allowed number of buildings for each type. These limits are stored in the buildingLimits map. For instance, the city can have up to 3 Solar Power Plants, 2 Coal Power Plants, 3 Parks, and so on. This establishes the rules regarding how many of each building type can exist within the city.

Method: canAdd(String name, int qty)

This method checks whether a specified quantity of a particular building type can be added without exceeding the predefined limit. It retrieves the current number of that building type in the city from buildingCounts and obtains the maximum allowed quantity from buildingLimits. The method

returns true if adding the specified quantity will not exceed the limit; otherwise, it returns false. This ensures that players cannot add more buildings than permitted.

Method: remainingFor(String name)

This method calculates how many additional units of a specific building type can still be added to the city. It does this by subtracting the current count of that building from its maximum allowed count. The method returns the remaining number that can be added, which is useful in the game for prompting the user about how many buildings they still have the capacity to place.

Method: addBuilding(Buildings b, int qty)

This method adds a specified number of buildings to the city and updates the city's metrics accordingly. It retrieves the name and current count of the building type, ensuring that the quantity to be added does not exceed the set limit. The method updates buildingCounts to reflect the new total for that building type. Additionally, it adjusts the city metrics by incorporating the effects of the new buildings on pollution, energy balance, happiness, and well-being. This ensures that the city's totals remain accurate and consistent with the buildings added.

Method: removeBuilding(Buildings b, int qty)

This method removes a specified number of buildings from the city and reverses their effects on the city's metrics. It retrieves the name and current count of the building type, capping the quantity to be removed if it exceeds the current count. The method updates buildingCounts to reflect the reduced number of that building type and subtracts the building's effects from the city totals for pollution, energy, happiness, and well-being. This allows players to adjust their city while ensuring that metrics are updated correctly when buildings are removed.

Method: getCount(String name)

This method returns the current number of a specific building type in the city. If no buildings of that type exist, it returns 0. This method is used to display current counts and enforce building limits.

Methods: `getTotalPollution()`, `getTotalEnergy()`, `getTotalHappiness()`, `getTotalWellBeing()`

These four methods provide the raw totals of the respective city metrics: pollution, energy balance, happiness, and well-being.

The City class serves as a central system for managing buildings, enforcing regulations, and maintaining current metrics about the city's status. It is responsible for all logic related to adding or removing buildings, as well as updating the city's overall statistics.

Graphical User Interface

The GUI aspect of this game allows users to see real time effects of the city planning. Instead of typing into the console like in the CLI implementation, users are allowed to use drop down menus such as in figure 1 to select the building type and buttons to remove and add buildings and to finish the game. We used the swift package to create the GUI as it was the easiest means we discovered to create the GUI.

To allow the image to respond to user input, a JFrame was created to host all aspects of the GUI. This frame contained several image layers made of panes that changed images using switch cases depending on the number of buildings a user added. These layers had a transparent background and a background layer was created at the utmost bottom of these layers.

A separate class was created to handle more interactive aspects of the application such as the buttons and drop down menu.

It is to note that this aspect is not the main component of our game and was created as an aesthetic appeal for the user, hence the limited focus on its implementation.

Appendix

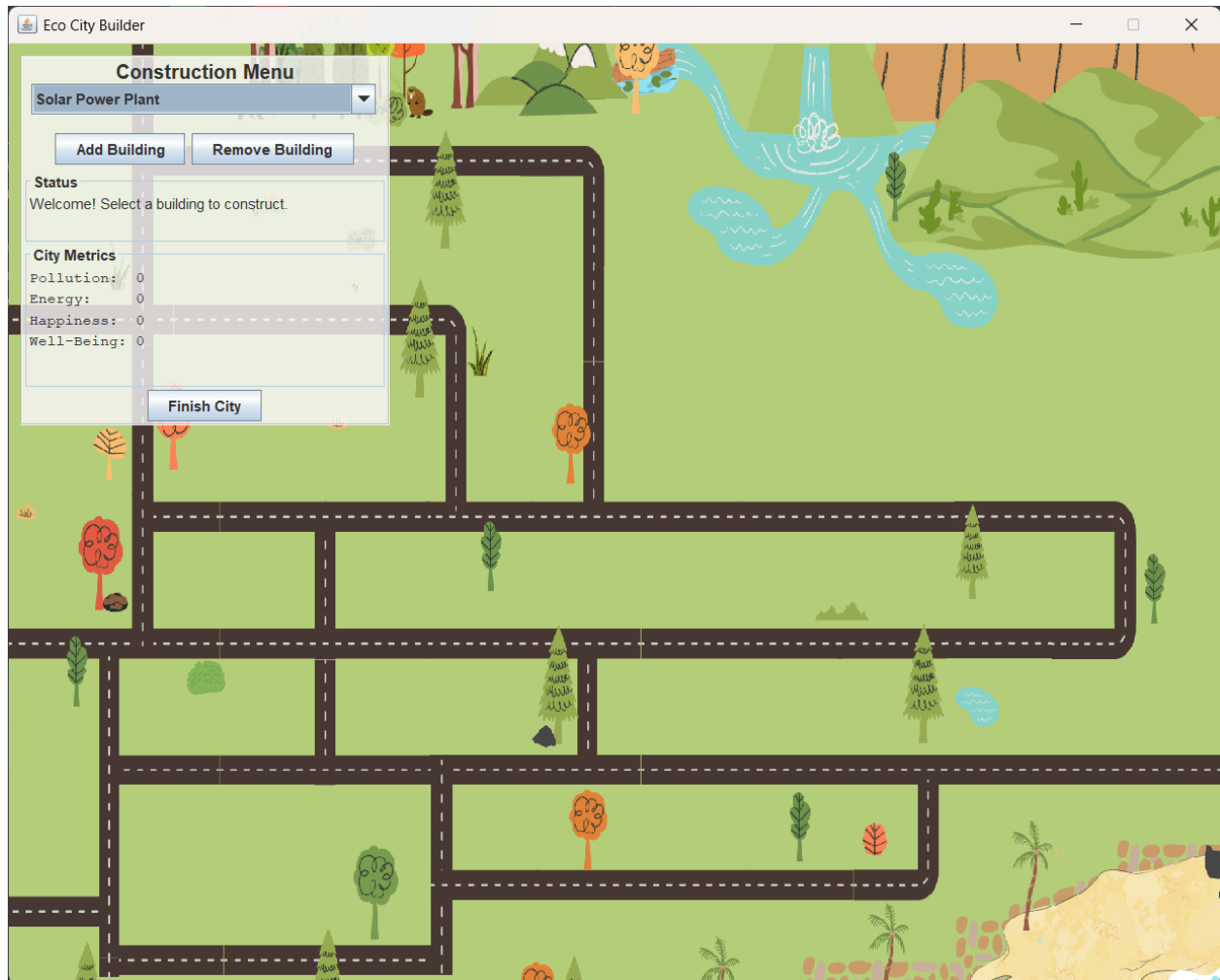


Figure 1

References

BroCode. (2020, September 14). *Java Swing GUI Full Course* ☕.

<https://www.youtube.com/watch?v=Kmgo00avvEw&t=142s>

Figen , S. (2024, August 29). *The Importance of SDG11 for Sustainable Urban Development*.

Winssolutions.<https://www.winssolutions.org/the-importance-of-sdg11-for-sustainable-urban-development-explained/>

Rashed, A. H. (2023, February 10). *The Impacts of Unsustainable Urbanization on the Environment*.Www.intechopen.com;IntechOpen.

<https://www.intechopen.com/chapters/86023>

United Nations. (2022). *Sustainable cities and human settlements* | *Department of Economic and SocialAffairs*.Sdgs.un.org.

<https://sdgs.un.org/topics/sustainable-cities-and-human-settlements>