Object-Oriented Programming
Project Report

Cellular-Automata-Based Ecosystem Simulation

Group Members:
Huzaifa Iqbal (31563)
Ali Hamza (30618)

Table of Contents

Introduction

Preface:

This project has been one of the most fascinating and intellectually rewarding experiences we have undertaken in our academic journey. From the very beginning, we were driven by a desire to bridge the gap between theory and real-world phenomena, looking not just to code, but to understand, replicate, and interact with the world around us through the lens of Object-Oriented Programming (OOP).

The idea of simulating a living, breathing ecosystem wasn't just exciting; it was immersive. It pushed us beyond the boundaries of regular programming problems and into the rich, complex domain of nature itself. We were no longer just dealing with variables and classes, but we were tracing the algorithms of life. How do animals behave? How do species survive, compete, and coexist? **In attempting to model this digitally, we found ourselves asking not just "how do we program this?" but rather, "how does nature program itself?"**

This report documents the design, implementation, and analysis of an Ecosystem Simulation developed as part of our OOP project. The simulation models a dynamic environment where plants, herbivores, and carnivores interact, reproduce, and compete for survival under the influence of seasonal changes. The project serves as a practical application of core OOP principles: encapsulation, abstraction, inheritance, and polymorphism, while demonstrating how complex systems can be effectively structured using modern C++.

In developing this simulation, we not only applied what we had learned in class but also explored new dimensions of learning, from biological patterns and environmental behavior to ethical modeling and randomness in systems. It gave us a deeper appreciation for the interconnectedness of systems, natural or computational, and showed us how powerful OOP can be when used to model the real world.

This project was more than just an assignment; it was a journey of curiosity, creativity, and problem-solving, and we hope this report reflects the same spirit with which we built it.

Purpose:

This project was originally envisioned as a research-driven simulation, not just a programming exercise, but an opportunity to explore and replicate the incredible complexity of our natural ecosystem using the lens of Object-Oriented Programming. From the start, we recognized that ecosystems are not linear, predictable systems; they are dynamic, adaptive, and deeply interwoven. Designing such a simulation meant stepping into the shoes of both a programmer and a naturalist.

We chose this project because it offered immense room for exploration and creativity. There is no single right way to model life. Its only patterns, probabilities, and principles. This gave us the flexibility to research biological systems, interpret animal behavior, understand plant life cycles, and translate all of that into structured, logical code. It became not only a technical challenge, but a journey into how life itself works.

The primary objective of this project was to:

- Design a realistic ecosystem simulation where entities follow biological and environmental rules.
- Implement OOP best practices to create modular, maintainable, and scalable code.
- Analyze challenges in simulating dynamic interactions and propose effective, logical solutions.
- Provide a foundation for future enhancements, such as additional species, more environmental variables, or graphical user interfaces.

Target Audience:

Beyond personal learning, this project was also designed with the end user in mind. Whether you're a student, teacher, or researcher, this simulation aims to provide a window into the natural world. It can assist in understanding environmental dynamics without needing to conduct complex field experiments or sift through layers of raw data. Through visualization and controlled experimentation, users can observe patterns, test behaviors, and draw insights in a simplified yet powerful environment.

This project is intended for a diverse audience that includes:

- **Computer Science students** learning about OOP and simulation modeling.
- **Educators** seeking compelling and practical case studies for teaching object-oriented design.
- **Developers and hobbyists** interested in agent-based modeling, simulations, or game design.
- **Zoologists and environmental researchers** looking to simulate animal behavior patterns and ecosystem dynamics in a controlled digital environment.

Acknowledgements:

I would like to thank Miss Samreen and the TA Syed Taha for their guidance during this project. Their feedback was invaluable in refining the design and addressing critical issues.

This report not only fulfills academic requirements but also serves as a reference for anyone interested in computational ecology or object-oriented system design. The accompanying code is open-source and extensible, inviting further experimentation and improvement.

Note to Readers:

For optimal understanding, read this report alongside the source code and README file, which provide technical implementation details and execution instructions.

.

Disclaimer:

Through extensive research and careful analysis, we have developed this Ecosystem Simulation project by applying our strongest technical capabilities and problem-solving skills. After conducting thorough investigations into ecological modeling and object-oriented design principles, we formulated the core logic and architecture that drive this simulation.

Our implementation reflects a thoughtful synthesis of biological behaviors, computational efficiency, and software engineering best practices. Every Component, from entity interactions to seasonal effects, has been meticulously designed and tested to create a realistic yet manageable simulation environment.

Simulation Rules and User Guide

The “Ecosystem Simulation” models a dynamic ecosystem consisting of plants, herbivores, and carnivores. The simulation runs over multiple months, with each month representing a time step where entities interact, reproduce, and compete for survival based on seasonal changes.

The project leverages key OOP principles such as encapsulation, abstraction, inheritance, and polymorphism, to create a structured and maintainable simulation. The ecosystem evolves dynamically, with entities responding to environmental factors such as seasons, food availability, and predator-prey relationships.

Before the simulation begins, the user will be guided through the setup process, where they configure the initial conditions and parameters of the ecosystem. This guide outlines how the simulation works, what is expected from the user, and how to interpret the outputs.

Simulation Initialization

When you start the program, you will be prompted to configure the simulation environment with the following inputs:

- **Hemisphere Selection:**

The user must choose between:

- 'N' for Northern Hemisphere
- 'S' for Southern Hemisphere

This affects seasonal changes, which in turn influence plant growth, animal behavior, and survival rates.

- **Simulation Duration:**

Enter the number of years you want the simulation to run.

Limit: Maximum duration is 10 years.

Internally, the simulation runs month-by-month (12 months per year), so a 10-year simulation covers 120 cycles.

- **Initial Populations:**

You will be asked to enter the number of:

- Plants (Recommended: 230–280)

- Herbivores (Recommended: 90–140)
- Carnivores (Recommended: 5–20)

Important Note: The total combined population (Plants + Herbivores + Carnivores) must not exceed 400, which represents the total available cells in the grid.

Grid Representation

The ecosystem is modeled as a 20×20 grid, giving a total area of 400 square kilometers. Each cell represents 1 km².

Every month, the simulation outputs the current state of the grid.

The grid uses a symbolic key to show the placement of different entities:

Symbol	Meaning
P	Plant
H	Male Herbivore
h	Female Herbivore
C	Male Carnivore
c	Female Carnivore
*	Empty Cell (no life)

Simulation Timeline

The simulation runs month-by-month, displaying:

- The grid (ecosystem map).
- Monthly statistics: births, deaths, reproduction events, plant spread, etc.
- Seasonal information: Each month is tagged with its respective season (Winter, Spring, Summer, Autumn), influenced by the chosen hemisphere.

12 months = 1 year. After each month, the simulation continues unless stopped or ended due to extinction scenarios.

Statistics and Events

At the end of each month, detailed statistics are shown:

- **Population Overview:** Current number of plants, herbivores, and carnivores.
- **Deaths:** From aging, starvation, predation, or seasonal stress.
- **Births:** Herbivores and carnivores born from successful reproduction.
- **Plant Spread:** Number of new plants spread naturally.
- **Migration:** Animals entering or leaving the ecosystem.

- **Season & Month:** Active month and its corresponding season.

Additionally, special monthly events such as mating, immigration, emigration, notable deaths, and survival milestones are printed to enrich the simulation experience.

Simulation Ending Conditions

The simulation ends if any of the following occurs before the user-defined duration:

- All animals have died.
- All plants have died, leaving herbivores and carnivores to starve.
- All herbivores have died, leaving carnivores with no prey.
- User quits manually by pressing Q during the simulation.
- Maximum simulation time is reached.

All Simulation Features in a Nutshell

✓ Hemisphere selection (Northern/Southern)
✓ Seasonal dynamics (Spring, Summer, Autumn, Winter)
✓ 20×20 grid representing 400 km²
✓ Customizable initial populations
✓ Monthly simulation updates (12 per year)
✓ Gender-specific organisms (Male/Female Herbivores & Carnivores)
✓ Predator-prey interaction logic
✓ Reproduction system with pregnancy and nursing
✓ Energy and aging mechanics
✓ Adaptive movement across 8 directions
✓ Plant growth and spread system
✓ Distance-based hunting and survival probabilities
✓ Carnivore competition and hunting priority
✓ Death due to starvation, age, or failed hunts
✓ Monthly statistical reports
✓ Simulation end conditions (extinction, max years, manual quit)

Project Timeline And Milestones

PHASE	TASKS	COMPLETION STATUS
Planning	• Define project scope and requirements • Research OOP principles for simulation design	Completed
Design	• Create class hierarchy (Entity, Animal, Plant, etc.) • Define interactions between entities	completed
Implementation	• Implement base classes (Entity, Animal, Plant) • Add derived classes (Herbivore, Carnivore) • Implement Grid and Simulation logic	completed
Testing	• Debug movement, reproduction, and death mechanics • Validate seasonal effects on behavior	completed
FSMs	• Using FSM for season changes	incomplete
Hemisphere	• Different Hemishphere apply different seasonal logic	Incomplete (southern is little problematic)
Finalization	• Optimize performance. • Write project report • Write README file	completed

Milestones and Development Journey

Our ecosystem simulation was developed step by step, growing from a simple prototype to a feature-rich model. Each phase helped us learn, improve, and make the simulation more realistic and meaningful.

1. Basic Prototype (100-Cell Grid)

We started with a simple 10×10 grid (100 cells). It included basic features like:

- Plant growth
- Herbivore and carnivore movement
- Predator-prey interaction
- Gender-specific reproduction and pregnancy tracking
- Monthly statistics

This version was not optimized but helped us test core logic. Since the map was small, the ecosystem usually collapsed within a year—which was expected, as animals quickly consumed all resources.

2. Full Simulation (400-Cell Grid)

Next, we expanded to a 20×20 grid (400 cells). This allowed for:

- Longer-lasting simulations
- Better movement and hunting behavior
- More balanced population growth

We also improved the display and layout to clearly show each month's grid and statistics.

3. Realism and Feature Enhancements

We added new features to make the simulation more lifelike:

- Seasons (Spring, Summer, Autumn, Winter)
- Hemisphere selection (Northern or Southern)
- Migration behavior and survival chances based on distance

We tested and adjusted values based on real-life research to make everything feel more natural.

4. Finite State Machines (FSM) In the final phase, we experimented with Finite State Machines to handle seasonal changes and animal behaviors. Although this part wasn't fully complete, it introduced new complexity and showed us how systems can be built around states and transitions.

Detailed Code Explanation

Core Classes & Their Responsibilities

1. Entity (Base Class)

- Purpose: Abstract base class for all entities (plants, herbivores, carnivores).
- Key Attributes:
- r, c (position on grid)
- EntityType (PLANT, HERBIVORE, CARNIVORE)
- alive (tracks if the entity is active)
- Key Methods:
- update() (abstract, called every simulation step)
- isAlive(), kill() (lifecycle management)

2. Plant (Derived from Entity)

- Purpose: Represents vegetation that grows and spreads.
- Behavior:
- Spreads to adjacent cells based on season.
- Dies due to old age or harsh weather (winter/autumn).
- Key Methods:
- update() (handles growth, death, and spreading).

3. Animal (Derived from Entity)

- Purpose: Base class for all animals (herbivores & carnivores).
- Key Attributes:
- energy, maxEnergy (starvation mechanics)
- gender, age, maxAge (reproduction & aging)

- visionRange (how far they can see food/predators)
- Key Methods:
- attemptEat(), move(), attemptReproduce() (core virtual functions).

4. Herbivore (Derived from Animal)

- Purpose: Eats plants and flees from carnivores.
- Behavior:
- Moves towards plants when hungry.
- Reproduces if energy and mate are available.
- Dies if starved or eaten.

5. Carnivore (Derived from Animal)

- Purpose: Hunts herbivores for food.
- Behavior:
- Chases nearby herbivores.
- Reproduces less frequently than herbivores.
- Dies if no prey is available.

6. Grid (Manages the World)

- Purpose: Handles entity placement, movement, and interactions.
- Key Methods:
- addEntity(), removeEntity(), moveEntity() (grid operations).
- findNearbyEntities() (used for hunting/reproduction).

7. Simulation (Orchestrates the Ecosystem)

- Purpose: Controls the simulation loop, seasons, and statistics.
- Key Methods:
- runMonth() (updates all entities, handles seasons).
- handleMigration() (animals move in/out based on season).

OOP Principles Applied

Encapsulation

- Each class encapsulates its own data and behavior.

- Example: Animal hides energy management (energy, maxEnergy) and exposes only necessary methods (attemptEat()).

Abstraction

- The Entity class provides an abstract update() method, forcing derived classes to implement their own logic.

- Simulation interacts with entities without knowing their internal details.

Inheritance

- Plant and Animal inherit from Entity.

- Herbivore and Carnivore inherit from Animal.

- Reduces code duplication (e.g., movement logic in Animal).

Polymorphism

- The update() method behaves differently for each entity:

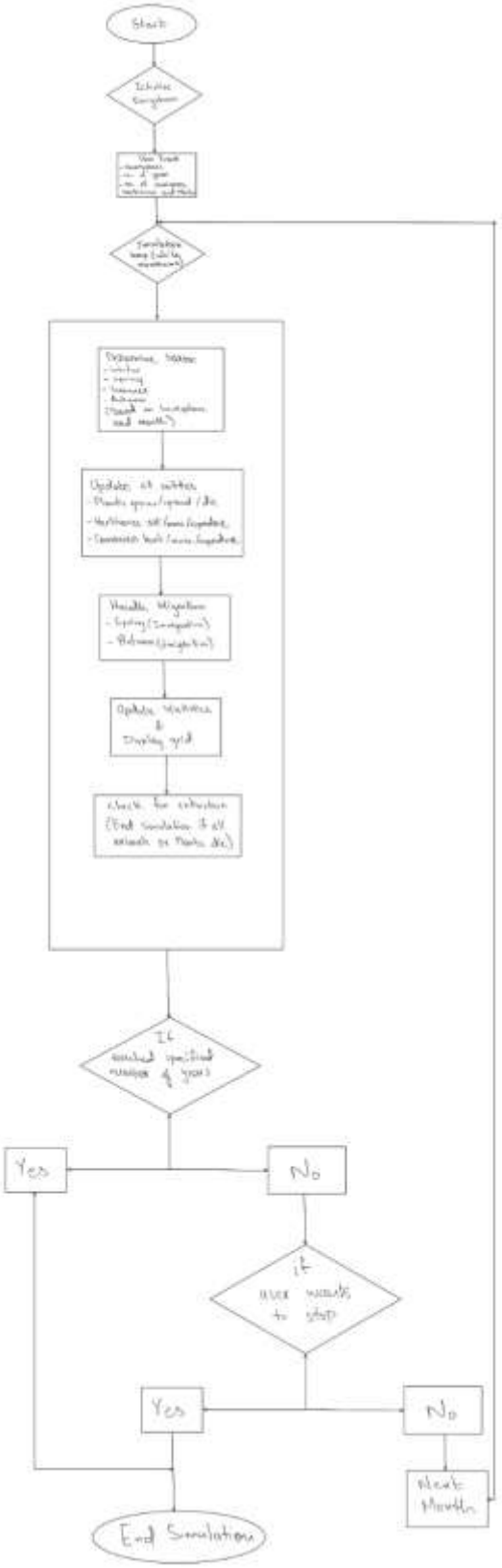
- Plant grows/spreads.

- Herbivore seeks plants.

- Carnivore hunts herbivores.

- Achieved via virtual functions and dynamic dispatch.

Flowchart



Challenges & Solutions

Throughout this project, we encountered significant challenges that required deep research into animal behavior and ecological systems. Determining realistic parameters for our simulation proved particularly demanding - every value in our code, from energy consumption to movement patterns, needed careful consideration based on biological research of herbivore and carnivore species.

The most complex challenge emerged in implementing the reproduction system. As we lacked prior expertise in animal reproductive cycles, we had to extensively study gestation periods, mating behaviors, and offspring care across species to develop a biologically plausible model. This aspect of the simulation required multiple iterations to balance realism with computational feasibility.

Key Research Areas:

- Behavioral patterns of predator-prey relationships
- Energy requirements across species
- Seasonal impact on reproduction cycles
- Population dynamics in confined ecosystems

Through persistent research and testing, we developed a system that, while simplified, maintains important ecological principles while functioning effectively within our simulation framework.

Challenge: Entity Interaction Logic

-Problem: Ensuring animals correctly detect food/prey while avoiding infinite loops.

-Solution: Used Grid::findNearbyEntities() with vision range to optimize searches.

Challenge: Seasonal Effects

-Problem: Adjusting behavior based on seasons (e.g., plants dying in winter).

-Solution: Added Season enum and modified update() methods to check current season.

Challenge: Performance with Large Grids

-Problem: Slow updates with many entities.

-Solution: Used smart pointers (std::shared_ptr) and optimized grid traversal.

Challenge: Plant growth properly with seasonal changes:

-Problem: Plant growth was not stable and unexpected in different seasons

-Solution: Proper research, multiple testing, and setting values and growth rates accordingly.

Testing & Validation

General Test Cases

1. Herbivore eats plant
2. Carnivore hunts herbivore
3. Plant spreads more in summer and spring and less in autumn and winter
4. Animals starve if no food
5. Animals immigrate and emigrate in spring and autumn
6. Animals reproduce more in summer and less in winter

Issues Discovered & Fixes

- Issue: Plants not spreading properly according to the different seasons.
- Fix: Changed plant spread chance and improved plant spread logic.
- Issue: Herbivores and carnivores not reproducing at all.
- Fix: changed reproduction logic so that less energy is required for reproduction and if male and female are close by and if they have enough energy then they reproduce.

Edge Test Cases:

- **Boundary Conditions**

Ensured that entities at the edges of the grid do not move outside the valid 20×20 area.

- **Plant Spreading**

Verified that plant spread occurs only to valid and empty adjacent cells.

- **Reproduction Rules**

Confirmed that two animals of the same gender cannot reproduce.

Checked behavior when total population is already at the maximum (400 entities).

- **Starvation Scenarios**

Simulated extreme cases like carnivores with no herbivores, or herbivores with no plants, leading to eventual die-off.

Input & Runtime Validation Test Cases

- **Population Limit Enforcement**

Prevented users from entering a total starting population greater than 400.

Ensured that even during simulation, total population never exceeds the grid limit.

- **Input Format Validation**

Required the number of years and entity populations to be valid integers.

- **Termination Conditions**

Simulation ends if all herbivores or all plants die.

Simulation stops automatically after reaching the user-specified number of years.

Conclusion

This project successfully implements an OOP-based ecosystem simulation with dynamic interactions between plants, herbivores, and carnivores. Key takeaways:

- Proper class hierarchy design simplifies complex behaviors.
- Polymorphism allows flexible entity updates.
- Seasonal mechanics add realism to the simulation.

Future improvements could include:

- More detailed weather systems (rain, drought).
- Additional species (omnivores)
- further breakdown herbivores and omnivores into specific animals (lion, cow)
- Add water and water animals.
- Add disease spread amongst animals
- Add human intervention (hunting)
- Properly implement FSM Mechanics
- More control for users.
- Graphical visualization (SDL/SFML).