Final Audit Report: Ecosystem Simulation Project

Developer: David Nwachukwu

Date: 08/07/2024

---

1. Project Overview

Introduction

The Ecosystem Simulation Project is an interactive, agent-based simulation that models the behaviour and interactions of various entities within a virtual environment. Utilising machine learning, this project simulates the evolution of entities' traits as they adapt to their surroundings, interact with each other, and respond to environmental stimuli.

Purpose and Objectives

The primary purpose of the Ecosystem Simulation Project is to create a dynamic and realistic simulation of an ecosystem. The key objectives include:

1. **Modeling Ecosystem Dynamics:** Accurately simulate interactions between entities and their environment, including resource consumption, reproduction, and survival.
2. **Machine Learning Integration:** Implement neural networks to allow entities to learn and adapt their behaviours based on environmental feedback.
3. **Visualisation:** Provide a graphical interface to observe the simulation in real-time, offering insights into the evolutionary processes and ecosystem stability.

Scope of the Project

The scope of this project includes the following components:

- **Environment Generation:** Creation of a virtual environment consisting of various terrain types (land, water, sand, trees) and resources (plants).
- **Entity Behavior Simulation:** Development of autonomous entities with distinct attributes and behaviours, governed by neural networks.
- **Data Collection and Analysis:** Real-time collection of simulation data, including population dynamics, average traits, and resource availability, with visualisation through graphs and statistics.

Key Features

- **Procedural Environment Generation:** The environment is procedurally generated using Perlin noise, ensuring a unique and diverse landscape for each simulation run.
- **Autonomous Entities:** Entities possess attributes such as hunger, thirst, reproductive urge, speed, and sensory radius. These attributes evolve over time through a combination of genetic inheritance and mutation.
- **Neural Network Decision-Making:** Entities use neural networks to make decisions based on their current state and environmental conditions, enabling adaptive and intelligent behaviours.
- **Dynamic Resource Management:** Resources in the environment, such as plants, grow and regrow over time, providing entities with varying availability of food.
- **Real-time Visualisation:** The simulation is visualised in real-time using Pygame, with graphical representations of entities and the environment, as well as statistics and trends over time.

Implementation Overview

The implementation of the Ecosystem Simulation Project involved several key components:

- **Environment Setup:** Using Pygame for rendering and noise for procedural generation, the environment consists of tiles representing different types of terrain.
- **Entity Attributes and Behaviours:** Entities are initialised with specific attributes, and their behaviours are governed by neural networks implemented using PyTorch.
- **Simulation Loop:** The main simulation loop handles updates to the environment and entities, processes user inputs, and renders the simulation in real-time.
- **Data Tracking and Visualization:** Data on entity populations, average traits, and other metrics are tracked over time and visualised using Matplotlib.

Development Process

The development process was iterative and involved continuous testing and refinement. The major phases included:

1. **Initial Planning:** Defined the project scope, objectives, and key features.
2. **Environment Setup:** Established the coding environment and implemented procedural environment generation.
3. **Core Functionality Development:** Developed the basic mechanics of the simulation, including entity movement and interaction.
4. **Neural Network Integration:** Integrated machine learning models to enhance entity decision-making and adaptation.
5. **Visualisation and UI Development:** Added graphical elements and user interface components for real-time observation and interaction.
6. **Testing and Optimization:** Conducted thorough testing and performance optimization to ensure a stable and efficient simulation.

Conclusion

The Ecosystem Simulation Project successfully created a dynamic and interactive simulation that models ecosystem interactions and evolution. By integrating neural networks, the project demonstrated how entities can learn and adapt to their environment, providing valuable insights into the complexities of ecosystem dynamics. The visualisation and data analysis components offer a comprehensive tool for studying and understanding these interactions, laying the groundwork for future enhancements and research in this field.

2. Project Objectives

Introduction

The Ecosystem Simulation Project was undertaken with several primary objectives aimed at advancing the understanding and visualisation of ecosystem dynamics through the use of modern computational techniques. The project's design leverages interactive simulation, machine learning, and real-time visualisation to create a comprehensive model of an evolving ecosystem.

Primary Objectives

1. Create a Dynamic, Interactive Simulation

The foremost objective of the project was to create a dynamic, interactive simulation that accurately models the interactions within an ecosystem. This involves:

- **Realistic Environment Generation:** Using procedural techniques, such as Perlin noise, to generate diverse and realistic environments consisting of land, water, trees, and sand tiles.
- **Autonomous Entities:** Developing entities with various traits (e.g., hunger, thirst, speed, reproductive urge) that evolve over time through genetic inheritance and mutation.
- **Resource Management:** Implementing a system where resources such as plants grow, regrow, and deplete, providing entities with challenges that mirror real-world ecological dynamics.

2. Implement Neural Networks for Learning and Adaptation

A critical aspect of the project was to integrate neural networks, enabling entities to learn from and adapt to their environment. This objective was achieved by:

- **Neural Network Models:** Designing neural networks using PyTorch to control entity behaviours based on their state and environmental conditions.
- **Reinforcement Learning:** Applying reinforcement learning techniques to allow entities to make decisions that maximise their survival and reproductive success.
- **Behavioural Evolution:** Allowing entities to exhibit complex behaviours such as seeking food and water, avoiding predators, finding mates, and hunting prey, thereby simulating natural selection and adaptation processes.

3. Visualise the Simulation with Real-Time Statistics and Graphical Representations

To facilitate understanding and analysis, the project aimed to provide real-time visualisation of the simulation, along with comprehensive statistics and graphical representations of the ecosystem's state. This included:

- **Real-Time Graphics:** Utilising Pygame to render the simulation environment and entities, allowing users to observe interactions and changes in real-time.

- **Statistical Tracking:** Collecting and displaying real-time statistics on entity populations, average traits, and resource levels.
- **Graphical Analysis:** Implementing plotting functions using Matplotlib to visualise trends over time, such as changes in population dynamics, trait averages, and dietary preferences.

Conclusion

The Ecosystem Simulation Project successfully met its primary objectives, creating a robust, dynamic, and interactive simulation environment. The integration of neural networks enabled entities to learn and adapt, providing a realistic model of ecosystem interactions. The real-time visualisation and comprehensive statistical tracking facilitated a deeper understanding of ecological dynamics and evolutionary processes. This project not only achieved its goals but also established a strong foundation for further research and enhancements in the field of computational ecology.

3. Development Process

Introduction

The development process of the Ecosystem Simulation Project was structured in iterative phases, allowing for continuous testing, refinement, and enhancement. This section outlines the key phases involved in the project's development, detailing the activities and achievements in each phase.

Major Phases

Initial Planning

The initial planning phase was crucial in setting the foundation for the project. Key activities included:

- **Defining Project Scope:** Determines the boundaries and limits of the project to ensure focus and manageability.
- **Setting Objectives:** Established clear and achievable objectives, including the creation of a dynamic simulation, neural network integration, and real-time visualisation.
- **Identifying Key Features:** Outlined essential features such as entity behaviours, environmental interactions, and statistical tracking.

Environment Setup

Setting up the coding environment was the next step, ensuring that all necessary tools and libraries were available for development. This involved:

- **Establishing the Development Environment:** Set up Python with necessary libraries like Pygame, PyTorch, NumPy, Matplotlib, and noise.
- **Version Control:** Implemented version control using Git to manage changes and collaboration effectively.

Core Functionality Development

In this phase, the basic mechanics of the simulation and the behaviours of entities were developed. Key achievements included:

- **Map Generation:** Created a procedurally generated environment using Perlin noise to simulate diverse terrains.
- **Entity Mechanics:** Developed the core mechanics for entity behaviours, including movement, hunger, thirst, and reproduction.
- **Resource Management:** Implemented systems for plant growth and regrowth, providing a dynamic resource environment.

Neural Network Integration

The integration of neural networks marked a significant enhancement in the simulation's complexity and realism. Activities included:

- **Model Design:** Designed and implemented neural networks using PyTorch to control entity behaviours.
- **Reinforcement Learning:** Applied reinforcement learning techniques, allowing entities to learn from their environment and improve their survival strategies.
- **Action and State Representation:** Defined state vectors and action spaces for entities, enabling the neural networks to make informed decisions.

Visualisation and UI Development

Adding graphical elements and user interface components was essential for real-time monitoring and interaction. Key developments were:

- **Real-Time Graphics:** Utilised Pygame to render the environment and entities, providing a visual representation of the simulation.
- **User Interface:** Developed UI components to display real-time statistics, graphs, and entity-specific information.
- **Interactive Elements:** Implemented features such as entity selection and detailed stats pop-ups for enhanced user interaction.

Testing and Optimization

The final phase focused on ensuring the simulation's robustness and performance. Key activities included:

- **Thorough Testing:** Conducted extensive testing to identify and fix bugs, ensuring stable and accurate simulation behaviour.
- **Performance Optimization:** Optimised code to improve performance, ensuring smooth and efficient execution of the simulation.
- **Data Tracking:** Implemented data tracking mechanisms to collect and visualise simulation statistics over time, aiding in performance monitoring and analysis.

Conclusion

The development process of the Ecosystem Simulation Project was methodical and iterative, involving continuous refinement to achieve the project's objectives. Each phase contributed to building a robust, dynamic, and interactive simulation, leveraging neural networks and real-time visualisation to model complex ecosystem interactions effectively. This structured approach ensured the successful completion of the project, laying a solid foundation for future enhancements and research in computational ecology.

4. System Architecture

Introduction

The system architecture of the Ecosystem Simulation Project is designed to model complex interactions within a dynamic environment, leveraging advanced technologies such as neural networks for entity behaviour. This section provides a detailed overview of the key components of the system architecture, highlighting their roles and interactions.

Major Components

Environment

The environment forms the foundation of the simulation, representing the virtual world in which entities interact. Key elements include:

- **Tiles:** The world is divided into a grid of tiles, each representing different types of terrain such as land, water, sand, and trees. Tiles have properties that influence entity behaviour and environmental dynamics.
    - **Tile Attributes:** Each tile has coordinates (`x`, `y`), a type (`land`, `water`, `sand`, `tree`), and a regrowth timer for plant regeneration.
    - **Tile Methods:** Tiles can update their state and render themselves on the display.
- **Plants:** Represented as stationary resources, plants are essential for herbivorous entities. Plants grow on land tiles and have a regrowth cycle.
    - **Plant Attributes:** Each plant has coordinates (`x`, `y`) on the grid.
    - **Plant Methods:** Plants can render themselves on the display.

Entities

Entities are autonomous agents within the simulation, each with unique attributes and behaviours influenced by neural networks. Key features include:

- **Attributes:** Entities have various attributes such as hunger, thirst, reproductive urge, speed, sensory radius, diet preference (herbivore, carnivore, omnivore), gender, and size.
- **Behavioural States:** Entities can be in different states such as idle, moving, seeking food, seeking water, seeking a mate, or hunting prey.
- **Lifecycle:** Entities age over time, and their attributes change accordingly. They can reproduce, leading to offspring with combined and potentially mutated genes from both parents.
- **Neural Network Integration:** Entities utilise neural networks to make decisions based on their current state and environment.

Entity Neural Network

The neural network component is critical for simulating intelligent behaviour in entities. Key aspects include:

- **Architecture:** The neural network consists of an input layer, hidden layers, and an output layer.
    - **Input Layer:** Processes the state vector, which includes normalised values of hunger, thirst, reproductive urge, and other relevant environmental factors.
    - **Hidden Layers:** Use ReLU activation functions to learn complex patterns and relationships.
    - **Output Layer:** Produces an action vector that determines the entity's next action.
- **Learning:** The neural network is trained using reinforcement learning techniques, where entities receive rewards or penalties based on their actions and outcomes.
    - **Q-learning:** Entities learn optimal behaviours by updating Q-values through experiences stored in a replay memory.

User Interface

The user interface provides interactive elements for user interaction and visualisation, enhancing the simulation experience. Key components include:

- **Real-Time Graphics:** The simulation environment and entities are rendered using Pygame, providing a visual representation of the ecosystem.
- **Interactive Elements:** Users can interact with the simulation by selecting entities, zooming in/out, and dragging the view.
- **Statistics and Graphs:** Real-time statistics such as entity population, average attributes, and diet preferences are displayed. Graphs visualise these statistics over time, aiding in performance monitoring and analysis.
    - **Entity Stats:** Detailed information about selected entities, including their attributes and current state, is displayed in a pop-up window.

Conclusion

The system architecture of the Ecosystem Simulation Project is a robust and modular design, integrating various components to create a dynamic and interactive simulation. By leveraging neural networks for entity behaviour and providing comprehensive visualisation tools, the architecture effectively models complex ecosystem interactions, making it a valuable tool for studying computational ecology. This structured approach ensures flexibility for future enhancements and scalability for more complex simulations.

5. Implementation Details

Libraries and Technologies

The Ecosystem Simulation Project leverages several powerful libraries and technologies to create a dynamic and interactive simulation environment. The primary libraries and technologies used include:

- **Pygame:** Utilised for rendering the simulation environment and handling user interactions. It provides a straightforward way to create graphical interfaces and manage user input.
- **Numpy:** Used for numerical operations and data manipulation. It facilitates efficient handling of arrays and matrices, which are essential for simulation calculations.
- **Matplotlib:** Employed for plotting graphs and visualising data. This library enables the generation of detailed plots to analyse simulation statistics and trends over time.
- **PyTorch:** Used to define and train neural networks. PyTorch provides the framework for creating and optimising the neural networks that drive the behaviour of entities in the simulation.
- **Noise:** Generated the procedural map of the environment. The noise library is used to create realistic terrain features such as land, water, sand, and tree tiles.

Key Features

The simulation incorporates several key features that contribute to its complexity and realism:

- **Dynamic Environment:** The environment consists of various types of tiles (land, water, sand, tree) and plants that grow and regrow. This dynamic setting provides a realistic backdrop for entity interactions.
- **Autonomous Entities:** Entities in the simulation have attributes such as hunger, thirst, reproductive urge, speed, and sensory radius. These attributes evolve over time, and entities can reproduce, leading to offspring with combined and mutated genes.
- **Neural Networks:** Entities make decisions based on the output of neural network models. These neural networks are trained to optimise entity behaviour based on their state and environment.
- **Real-time Statistics:** The simulation displays real-time statistics on entities and the environment, including population counts, average traits, and diet preferences. These statistics provide insights into the evolving ecosystem.
- **Graphical Visualisation:** Data over time, such as population trends and average traits, are plotted using Matplotlib. These visualisations help in analysing the performance and dynamics of the simulation.

Code Structure

The code for the Ecosystem Simulation Project is structured into several key components:

- **Tile and Plant Classes:** These classes define the environment elements.

    - **Tile Class:** Represents individual tiles in the environment, with attributes for position, type, and regrowth timer. Methods include updating and drawing tiles.
    - **Plant Class:** Represents plants in the environment, with attributes for position. Methods include drawing plants.
- **Entity Class:** Represents individual agents within the simulation, with attributes and behaviours.
    - **Attributes:** Include hunger, thirst, reproductive urge, speed, sensory radius, diet preference, gender, size, and age.
    - **Methods:** Include moving, seeking food/water, mating, hunting prey, random walking, updating gestation, and drawing entities.
- **EntityNet Class:** Defines the neural network architecture used by entities.
    - **Architecture:** Includes an input layer, hidden layers, and an output layer. Uses ReLU activation functions and is trained using Q-learning techniques.
- **Main Simulation Loop:** Controls the flow of the simulation, including environment updates, entity updates, rendering, and user input handling.
    - **Environment Update:** Handles the regrowth of plants and the state of tiles.
    - **Entity Update:** Manages the movement and behaviour of entities, as well as their interactions with the environment and each other.
    - **Rendering:** Draws the environment, plants, and entities on the screen. Also displays real-time statistics and entity-specific information when selected.
    - **User Input Handling:** Allows users to interact with the simulation through mouse and keyboard inputs, enabling actions such as selecting entities, zooming, and dragging the view.

Detailed Code Snippets

```python
# Environment setup
class Tile:
    def __init__(self, x, y, tile_type):
        self.x = x
        self.y = y
        self.type = tile_type
        self.regrowth_timer = 0

    def update(self):
        if self.type == "land" and self.regrowth_timer > 0:
            self.regrowth_timer -= 1

    def draw(self, win, offset_x, offset_y, scale):
        if self.type == "water":
            color = BLUE
        elif self.type == "land":
            color = GREEN
        elif self.type == "tree":
            color = BROWN
        elif self.type == "sand":
            color = SAND
        pygame.draw.rect(win, color,
                         ((self.x * TILE_SIZE - offset_x) * scale,
                          (self.y * TILE_SIZE - offset_y) * scale,
                          TILE_SIZE * scale,
                          TILE_SIZE * scale))

class Plant:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def draw(self, win, offset_x, offset_y, scale):
        pygame.draw.circle(win, DARK_GREEN,
                           (int((self.x * TILE_SIZE + TILE_SIZE / 2 - offset_x) * scale),
                            int((self.y * TILE_SIZE + TILE_SIZE / 2 - offset_y) * scale)),
                           int(TILE_SIZE / 4 * scale))
```

```python
def generate_map(width, height):
    scale = 30.0
    octaves = 6
    persistence = 0.5
    lacunarity = 2.0
    seed = np.random.randint(0, 100)

    world = np.zeros((width // TILE_SIZE, height // TILE_SIZE))

    for i in range(width // TILE_SIZE):
        for j in range(height // TILE_SIZE):
            world[i][j] = noise.pnoise2(i/scale,
                                        j/scale,
                                        octaves=octaves,
                                        persistence=persistence,
                                        lacunarity=lacunarity,
                                        repeatx=width // TILE_SIZE,
                                        repeaty=height // TILE_SIZE,
                                        base=seed)

    tiles = []
    plants = []
    for y in range(height // TILE_SIZE):
        row = []
        for x in range(width // TILE_SIZE):
            if world[x][y] < -0.05:
                tile_type = "water"
            elif world[x][y] < 0:
                tile_type = "sand"
            elif world[x][y] < 0.3:
                tile_type = "land"
                if random.random() < PLANT_PROBABILITY:
                    plants.append(Plant(x, y))
            else:
                tile_type = "tree"
            row.append(Tile(x, y, tile_type))
        tiles.append(row)

    return tiles, plants

tiles, plants = generate_map(MAP_WIDTH, MAP_HEIGHT)
```

```python
class Entity:
    def __init__(self, x, y, tiles, genes=None, generation=1, parent_genes=None, model=None):
        self.x, self.y = self.find_valid_position(int(x), int(y), tiles)
        self.target_x, self.target_y = self.x, self.y
        self.hunger = genes['hunger'] if genes else HUNGER_THRESHOLD
        self.thirst = genes['thirst'] if genes else THIRST_THRESHOLD
        self.reproductive_urge = genes['reproductive_urge'] if genes else 0
        self.speed = genes['speed'] if genes else random.uniform(1.0, 4.0)
        self.sensory_radius = genes['sensory_radius'] if genes else random.randint(1, 10)
        self.reproduction_threshold = genes['reproduction_threshold'] if genes else REPRODUCTION_THRESHOLD
        self.hunger_threshold = genes['hunger_threshold'] if genes else HUNGER_THRESHOLD
        self.thirst_threshold = genes['thirst_threshold'] if genes else THIRST_THRESHOLD
        self.diet_preference = genes['diet_preference'] if genes else 'herbivore'
        self.offspring_count = genes['offspring_count'] if genes else random.randint(1, 3)
        self.size_gene = genes['size'] if genes else random.uniform(0.5, 1.5)  # Size gene
        self.alive = True
        self.gender = random.choice(['male', 'female'])
        self.gestation_duration = genes['gestation_duration'] if genes and self.gender == 'female' else random.randint(50, 200)
        self.gestation_timer = genes['gestation_timer'] if genes and self.gender == 'female' else 0  # Timer for gestation period
        self.child_genes = genes.copy() if genes else None
        self.generation = generation
        self.parent_genes = parent_genes if parent_genes else {}
        self.unimpressed_females = set()
        self.state = 'idle'  # Current state of the entity
        self.direction = (random.uniform(-1, 1), random.uniform(-1, 1))  # Initial random direction
        self.steps_since_direction_change = 0  # Steps taken in the current direction
        self.pregnant = False if self.gender == 'female' else None  # Pregnancy state for females
        self.model = model  # Neural network model
        self.age = 0  # Age of the entity
        self.old_age_threshold = genes['old_age_threshold'] if genes else random.randint(1000, 3000)
        self.size = 0.2  # Starting size, will grow as the entity ages

    def find_valid_position(self, x, y, tiles):
        while tiles[int(y)][int(x)].type == "water" or tiles[int(y)][int(x)].type == "tree":
            x, y = random.randint(0, MAP_WIDTH // TILE_SIZE - 1), random.randint(0, MAP_HEIGHT // TILE_SIZE - 1)
        return x, y

    def is_valid_move(self, x, y, tiles):
        if 0 <= int(x) < MAP_WIDTH // TILE_SIZE and 0 <= int(y) < MAP_HEIGHT // TILE_SIZE:
            return tiles[int(y)][int(x)].type in ["land", "sand"]
        return False

    def check_adjacent_water(self, tiles):
        for dx in range(-1, 2):
            for dy in range(-1, 2):
                if 0 <= int(self.x + dx) < MAP_WIDTH // TILE_SIZE and 0 <= int(self.y + dy) < MAP_HEIGHT // TILE_SIZE:
                    if tiles[int(self.y + dy)][int(self.x + dx)].type == "water":
                        return True
        return False
```

```
def move(self, tiles, food_sources, mates):
    if not self.alive:
        self.state = 'dead'
        return

    self.hunger -= 1
    self.thirst -= 1
    self.age += 1

    if self.age >= self.old_age_threshold:
        self.alive = False
        self.state = 'dead'
        return

    # Grow as the entity ages
    if self.age < ADULT_AGE:
        self.size = 0.2 + 0.8 * (self.age / ADULT_AGE)
    else:
        self.size = 1.0

    if not self.pregnant:
        if self.reproductive_urge < self.reproduction_threshold:
            self.reproductive_urge += 1

    if self.hunger <= 0 or self.thirst <= 0:
        self.alive = False
        self.state = 'dead'
        return

    state_vector = self.get_state_vector(tiles, food_sources, mates)
    state_tensor = torch.tensor(state_vector, dtype=torch.float32)
    action_vector = self.model(state_tensor).detach().numpy()
    self.take_action(action_vector, tiles, food_sources, mates)

    # Smooth movement interpolation
    self.x += (self.target_x - self.x) * 0.1
    self.y += (self.target_y - self.y) * 0.1

def get_state_vector(self, tiles, food_sources, mates):
    # Define a state vector for the neural network
    state_vector = [
        self.hunger / self.hunger_threshold,
        self.thirst / self.thirst_threshold,
        self.reproductive_urge / self.reproduction_threshold,
        self.x / MAP_WIDTH,
        self.y / MAP_HEIGHT,
        len(food_sources) / (MAP_WIDTH * MAP_HEIGHT),
        len(mates) / (MAP_WIDTH * MAP_HEIGHT),
        1 if self.hunger < self.hunger_threshold * 0.2 else 0,  # Binary flag for critical hunger
        1 if self.thirst < self.thirst_threshold * 0.2 else 0   # Binary flag for critical thirst
    ]
    return state_vector
```

```python
def take_action(self, action_vector, tiles, food_sources, mates):
    action = np.argmax(action_vector)
    if action == 0:  # Move in a direction
        self.direction = (random.uniform(-1, 1), random.uniform(-1, 1))
        new_x = self.x + self.direction[0] * self.speed
        new_y = self.y + self.direction[1] * self.speed
        if self.is_valid_move(new_x, new_y, tiles):
            self.target_x, self.target_y = new_x, new_y
            self.state = 'moving'
    elif action == 1:  # Seek food
        self.seek_food(tiles, food_sources)
    elif action == 2:  # Seek water
        self.seek_water(tiles)
    elif action == 3:  # Seek mate
        if self.age >= ADULT_AGE:
            self.seek_mate(mates, tiles)
    elif action == 4:  # Hunt prey
        self.hunt_prey(tiles, mates)
    else:  # Random walk
        self.random_walk(tiles)

def seek_food(self, tiles, food_sources):
    target_food = None
    min_distance = float('inf')
    for food in food_sources:
        distance = np.sqrt((self.x - food.x) ** 2 + (self.y - food.y) ** 2)
        if distance < min_distance and distance <= self.sensory_radius:
            min_distance = distance
            target_food = food

    if target_food:
        if min_distance < 1:
            food_sources.remove(target_food)
            tiles[int(target_food.y)][int(target_food.x)].regrowth_timer = REGROWTH_TIME
            self.hunger = self.hunger_threshold
            self.state = 'idle'  # Reset state after eating
        else:
            direction_x = (target_food.x - self.x) / min_distance
            direction_y = (target_food.y - self.y) / min_distance
            new_x = self.x + direction_x * self.speed
            new_y = self.y + direction_y * self.speed
            if self.is_valid_move(new_x, new_y, tiles):
                self.target_x, self.target_y = new_x, new_y
                self.state = 'seeking food'
    def seek_water(self, tiles):
        if self.check_adjacent_water(tiles):
            self.thirst = self.thirst_threshold
            self.state = 'idle'  # Reset state after drinking
        else:
            for y in range(max(0, int(self.y - self.sensory_radius)), min(MAP_HEIGHT // TILE_SIZE, int(self.y + self.sensory_radius))):
                for x in range(max(0, int(self.x - self.sensory_radius)), min(MAP_WIDTH // TILE_SIZE, int(self.x + self.sensory_radius))):
                    if tiles[int(y)][int(x)].type == "water":
                        distance = np.sqrt((self.x - x) ** 2 + (self.y - y) ** 2)
                        if distance <= self.sensory_radius:
                            direction_x = (x - self.x) / distance
                            direction_y = (y - self.y) / distance
                            new_x = self.x + direction_x * self.speed
                            new_y = self.y + direction_y * self.speed
                            if self.is_valid_move(new_x, new_y, tiles):
                                self.target_x, self.target_y = new_x, new_y
                                self.state = 'seeking water'
                                return

    def seek_mate(self, mates, tiles):
        if self.reproductive_urge >= self.reproduction_threshold:
            if self.gender == 'male':
                for mate in mates:
                    if mate != self and mate.alive and mate.reproductive_urge >= self.reproduction_threshold and mate.gender == 'female':
                        distance = np.sqrt((self.x - mate.x) ** 2 + (self.y - mate.y) ** 2)
                        if distance <= self.sensory_radius:
                            self.reproductive_urge = 0
                            mate.reproductive_urge = 0
                            mate.gestation_timer = mate.gestation_duration  # Start gestation period
                            mate.child_genes = self.combine_genes(self, mate)
                            mate.parent_genes = {'mother': mate.get_genes(), 'father': self.get_genes()}  # Store parent genes
                            mate.pregnant = True  # Mark the female as pregnant
                            self.state = 'idle'  # Reset state after mating
                            return
            elif self.gender == 'female':
                for mate in mates:
                    if mate != self and mate.alive and mate.reproductive_urge >= self.reproduction_threshold and mate.gender == 'male':
                        distance = np.sqrt((self.x - mate.x) ** 2 + (self.y - mate.y) ** 2)
                        if distance <= self.sensory_radius:
                            self.reproductive_urge = 0
                            mate.reproductive_urge = 0
                            self.gestation_timer = self.gestation_duration  # Start gestation period
                            self.child_genes = self.combine_genes(self, mate)
                            self.parent_genes = {'mother': self.get_genes(), 'father': mate.get_genes()}  # Store parent genes
                            self.pregnant = True  # Mark the female as pregnant
                            self.state = 'idle'  # Reset state after mating
                            return
        if self.state == 'idle':
            self.random_walk(tiles)
```

```python
def hunt_prey(self, tiles, entities):
    if self.diet_preference in ['carnivore', 'omnivore']:
        target_prey = None
        min_distance = float('inf')
        for entity in entities:
            if entity != self and entity.alive and entity.x != self.x and entity.y != self.y:
                distance = np.sqrt((self.x - entity.x) ** 2 + (self.y - entity.y) ** 2)
                if distance < min_distance and distance <= self.sensory_radius and entity.size_gene < self.size_gene:
                    min_distance = distance
                    target_prey = entity

        if target_prey:
            if min_distance < 1:
                target_prey.alive = False
                self.hunger = self.hunger_threshold
                self.state = 'idle'  # Reset state after eating
            else:
                direction_x = (target_prey.x - self.x) / min_distance
                direction_y = (target_prey.y - self.y) / min_distance
                new_x = self.x + direction_x * self.speed
                new_y = self.y + direction_y * self.speed
                if self.is_valid_move(new_x, new_y, tiles):
                    self.target_x, self.target_y = new_x, new_y
                    self.state = 'hunting prey'

def combine_genes(self, parent1, parent2):
    genes = {}
    numeric_genes = ['hunger', 'thirst', 'reproductive_urge', 'speed', 'sensory_radius', 'gestation_duration', 'gestation_timer', 'reproduction_threshold', 'hunger_threshold', 'thirst_threshold', 'offspring_count', 'old_age_threshold', 'size']

    for key in numeric_genes:
        gene_value = (parent1.get_genes()[key] + parent2.get_genes()[key]) / 2
        if random.random() < MUTATION_PROBABILITY:
            gene_value *= random.uniform(0.7, 1.3)
        genes[key] = gene_value

    # Handle diet preference separately
    if random.random() < MUTATION_PROBABILITY:
        genes['diet_preference'] = random.choice(['herbivore', 'carnivore', 'omnivore'])
    else:
        genes['diet_preference'] = parent1.diet_preference if random.random() < 0.5 else parent2.diet_preference

    return genes

def random_walk(self, tiles):
    self.steps_since_direction_change += 1
    if self.steps_since_direction_change > 50:  # Change direction every 50 steps
        self.direction = (random.uniform(-1, 1), random.uniform(-1, 1))
        self.steps_since_direction_change = 0

    new_x = self.x + self.direction[0] * self.speed
    new_y = self.y + self.direction[1] * self.speed

    if self.is_valid_move(new_x, new_y, tiles):
        self.target_x, self.target_y = new_x, new_y
        self.state = 'wandering'
```

6. Neural Network Implementation

The Ecosystem Simulation Project leverages neural networks to simulate realistic and adaptive behaviours in the entities within the ecosystem. This section outlines the design, implementation, and functionalities of the neural networks used in the project.

Neural Network Design

The neural networks in this project are implemented using PyTorch, a powerful and flexible deep learning library. The neural network architecture for each entity consists of three fully connected layers:

- **Input Layer:** The input size is determined by the state vector of the entity, which includes parameters such as hunger, thirst, reproductive urge, and position within the map.
- **Hidden Layers:** Two hidden layers with a configurable number of neurons. Each hidden layer uses the ReLU activation function to introduce non-linearity and enable the network to learn complex patterns.
- **Output Layer:** The output layer size corresponds to the number of possible actions an entity can take (e.g., move, seek food, seek water, seek mate, hunt prey, random walk).

The architecture is defined in the `EntityNet` class, which extends the `nn.Module` class in PyTorch.

```python
# Define Neural Network for Entities
class EntityNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(EntityNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.fc3 = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Entity State Representation

Each entity's state is represented by a vector of normalised values that capture its internal conditions and environmental context. The state vector includes the following elements:

- Hunger level
- Thirst level
- Reproductive urge
- X and Y positions on the map

- Density of food sources and potential mates in the vicinity
- Flags for critical hunger and thirst conditions

This state vector is used as input to the neural network to determine the best action for the entity.

Action Selection

The neural network outputs a vector representing the estimated value of each possible action. The entity selects the action with the highest value using the `np.argmax` function. The possible actions include:

- **Move in a random direction**
- **Seek food**
- **Seek water**
- **Seek a mate**
- **Hunt prey**
- **Random walk**

The selected action is then executed, influencing the entity's behaviour and interactions with the environment.

Reward System

The reward system is crucial for training the neural networks using reinforcement learning. Entities receive rewards or penalties based on their actions and resulting conditions:

- **Positive Rewards:** Satisfying hunger, thirst, and reproductive urge.
- **Negative Penalties:** Experiencing critical hunger or thirst, or death.

The reward function is defined as follows:

```python
def get_reward(entity):
    reward = 0
    if entity.hunger >= entity.hunger_threshold:
        reward += 1
    if entity.thirst >= entity.thirst_threshold:
        reward += 1
    if entity.reproductive_urge >= entity.reproduction_threshold:
        reward += 1
    if entity.hunger < entity.hunger_threshold * 0.2:  # Penalize low hunger
        reward -= 5
    if entity.thirst < entity.thirst_threshold * 0.2:  # Penalize low thirst
        reward -= 5
    if not entity.alive:
        reward -= 10
    return reward
```

Training the Neural Networks

The neural networks are trained using Q-learning, a model-free reinforcement learning algorithm. The key components of the training process include:

- **Experience Replay:** Experiences (state, action, reward, next state) are stored in a memory buffer. A batch of experiences is sampled randomly for training, which helps break the correlation between consecutive experiences and improves learning stability.
- **Q-Learning Update:** The neural network is trained to minimise the mean squared error (MSE) between the predicted Q-values and the target Q-values. The target Q-values are computed using the Bellman equation:

```
q_values = entity.model(state_tensor)
next_q_values = entity.model(next_state_tensor)

target = reward_tensor + gamma * torch.max(next_q_values).detach()

loss = nn.MSELoss()(q_values[action], target)
```

- **Optimization:** The Adam optimizer is used to update the neural network weights based on the computed loss.

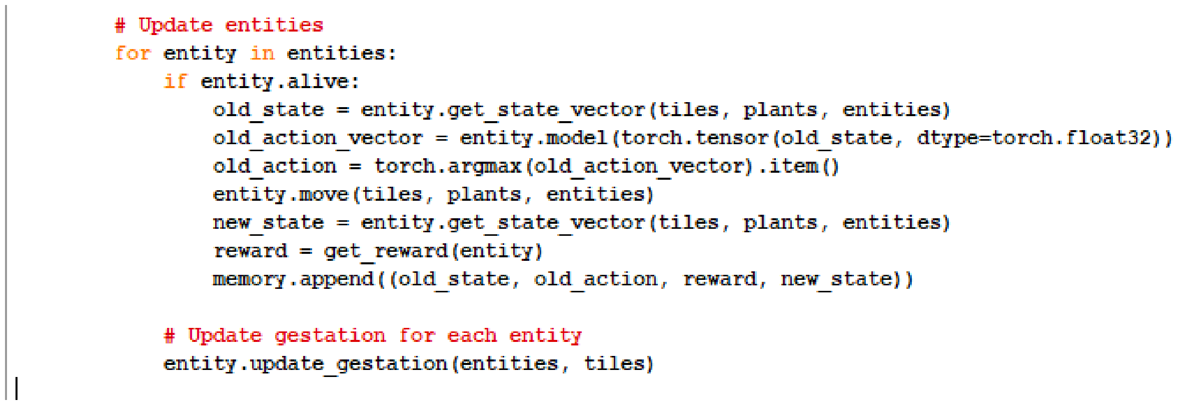
```
# Define optimizer
optimizer = optim.Adam([param for entity in entities for param in entity.model.parameters()], lr=learning_rate)
```

Integration and Performance

The neural networks are integrated into the entity update loop, ensuring that entities make decisions based on their current state and the learned policies. Continuous training and optimization allow the entities to adapt and improve their behaviours over time.

python

Copy code



```
        # Update entities
        for entity in entities:
            if entity.alive:
                old_state = entity.get_state_vector(tiles, plants, entities)
                old_action_vector = entity.model(torch.tensor(old_state, dtype=torch.float32))
                old_action = torch.argmax(old_action_vector).item()
                entity.move(tiles, plants, entities)
                new_state = entity.get_state_vector(tiles, plants, entities)
                reward = get_reward(entity)
                memory.append((old_state, old_action, reward, new_state))

            # Update gestation for each entity
            entity.update_gestation(entities, tiles)
```

Conclusion

The implementation of neural networks in the Ecosystem Simulation Project has been instrumental in achieving realistic and adaptive behaviours in simulated entities. Through careful design, state representation, action selection, reward systems, and training methodologies, the project demonstrates the power and flexibility of neural networks in ecological modelling and simulation. Future enhancements will continue to build on this foundation, further refining the behaviours and interactions within the simulated ecosystem.

7. Testing and Results

The Ecosystem Simulation Project underwent extensive testing to evaluate the behaviour and evolution of entities within the simulated environment. Various conditions were set to observe how the entities adapted, interacted, and evolved over time. Key aspects of the testing process and results are detailed below.

Testing Methodology

1. **Simulation Runs:**
    - Multiple simulation runs were conducted with varying initial conditions, such as different numbers of entities, resource distributions, and genetic variations.
    - Each run lasted for a significant number of time steps to allow for observable evolution and adaptation.
2. **Parameter Variations:**
    - The simulations varied parameters like mutation probability, reproduction threshold, hunger and thirst thresholds, and sensory radius.
    - These variations aimed to test the robustness of the simulation and the adaptability of the entities to changing environments.
3. **Data Collection:**
    - During each simulation run, data on population dynamics, trait distributions, and environmental conditions were collected.
    - Real-time statistics and graphical plots were generated to provide insights into the ongoing simulation.

Key Results

Population Dynamics

One of the primary observations was the fluctuation in population size due to the availability of resources and interactions among entities. Key findings include:

- **Resource-Driven Population Changes:**
    - The population size varied in response to the availability of food and water resources. Periods of resource abundance led to population growth, while scarcity resulted in population decline.
    - Entities' ability to find and consume resources directly impacted their survival and reproductive success.
- **Interaction-Driven Dynamics:**
    - Predatory behaviour among carnivorous and omnivorous entities influenced the population sizes of prey species. Increased predation pressure led to a decline in prey populations, affecting the overall ecosystem balance.
    - Social interactions, such as mating and competition, also played a role in population dynamics.

Trait Evolution

The simulation demonstrated significant changes in traits over time, indicating adaptation to the environment. Notable trends include:

- **Speed and Sensory Radius:**
    - Entities with higher speeds and larger sensory radii had a higher likelihood of finding resources and mates, leading to the propagation of these traits.
    - Over successive generations, average speed and sensory radius values increased, demonstrating positive selection for these traits.
- **Reproduction and Survival Thresholds:**
    - Entities with optimised thresholds for hunger, thirst, and reproduction exhibited better survival and reproductive success. These thresholds evolved to balance the need for resource acquisition and reproduction.
    - The average values for these thresholds converged towards optimal ranges that maximised fitness in the given environment.
- **Diet Preference:**
    - The distribution of diet preferences (herbivore, carnivore, omnivore) shifted based on resource availability and predation pressure.
    - In environments with abundant plant resources, herbivores thrived, whereas environments with more prey supported higher carnivore populations.

Graphical Analysis

Graphical plots provided valuable insights into the dynamics of the simulated ecosystem. Key visualisations include:

- **Population Trends:**
    - Plots of entity population over time showed distinct cycles of growth and decline corresponding to resource availability and environmental conditions.
    - These plots helped identify critical periods of resource scarcity and population bottlenecks.
- **Trait Distributions:**
    - Graphs depicting average values for speed, sensory radius, reproduction threshold, hunger threshold, and other traits illustrated the evolutionary trends within the population.
    - These visualisations highlighted the adaptive changes and the emergence of favourable traits over generations.
- **Diet Preferences:**
    - Plots showing the percentage distribution of diet preferences (herbivore, carnivore, omnivore) provided insights into the ecological balance and the impact of predation on population dynamics.
    - Shifts in diet preference distributions indicated changes in the ecosystem structure and resource utilisation patterns.

Conclusion

The testing phase of the Ecosystem Simulation Project successfully demonstrated the complex interactions and adaptive behaviours of entities within a dynamic environment. The key results highlight the significance of resource availability, interaction dynamics, and trait evolution in shaping the ecosystem. The graphical analysis provided a comprehensive view of the simulation's performance, offering valuable insights into the underlying processes driving the observed patterns. The results validate the effectiveness of the simulation model and its potential for further studies in evolutionary biology and ecology.

8. Challenges and Solutions

Challenges

The development and testing of the Ecosystem Simulation Project presented several challenges, each requiring thoughtful solutions to ensure a functional and efficient simulation. Key challenges included:

1. **Balancing Resource Availability:**
    - Maintaining an appropriate balance of resources (food and water) was crucial to ensure a stable population of entities. Insufficient resources led to rapid population decline, while excessive resources caused unrealistic population growth.
2. **Entity Behaviour Optimization:**
    - Tuning the neural networks that governed entity behaviour was critical to achieving realistic and effective actions. The challenge was to ensure entities could make decisions that promoted survival and reproduction while navigating the complex environment.
3. **Performance Optimization:**
    - As the simulation's complexity increased, maintaining real-time performance became challenging. Efficiently managing large datasets and ensuring smooth rendering of the simulation required significant optimization efforts.

Solutions

To address these challenges, several solutions were implemented, enhancing the simulation's robustness and performance:

1. **Dynamic Resource Management:**
    - **Regrowth Mechanics:** Implemented a system where plants could regrow over time, ensuring a steady supply of food resources. Tiles representing land had a probabilistic chance to generate new plants after a certain period, maintaining resource availability.
    - **Probabilistic Resource Placement:** Initial placement of resources was determined using a probabilistic approach, distributing plants and water sources in a balanced manner across the map. This helped in creating a realistic and varied environment for the entities.
2. **Neural Network Tuning:**
    - **Parameter Adjustment:** Fine-tuned the parameters of the neural networks, such as learning rates, reward functions, and action selection mechanisms. This ensured that entities could learn and adapt their behaviours more effectively.
    - **Training Process Improvement:** Enhanced the training processes by incorporating techniques like experience replay and mini-batch updates. This allowed the neural networks to learn from a diverse set of experiences, improving decision-making accuracy.
3. **Efficient Algorithms:**

    - **Code Optimization:** Refactored the code to improve efficiency, including optimising loops, reducing redundant calculations, and enhancing memory management. This resulted in smoother performance and reduced computational overhead.
    - **Handling Large Datasets:** Implemented efficient data structures and algorithms to handle the large datasets generated during the simulation. Techniques such as batching updates and using optimised libraries (e.g., NumPy, PyTorch) ensured efficient data processing.

Implementation Details

1. **Dynamic Resource Management:**
    - **Regrowth Time:** Each land tile had a regrowth timer that counted down, and once it reached zero, the tile had a chance to regrow a plant based on a predefined probability. This mechanism ensured a dynamic and self-sustaining ecosystem.
    - **Initial Resource Distribution:** The initial distribution of resources was generated using Perlin noise, creating a realistic terrain with varying elevations and resource concentrations. This randomness mimicked natural landscapes, providing a diverse environment for the entities.
2. **Neural Network Tuning:**
    - **State Representation:** Entities' state vectors were carefully designed to include relevant features such as hunger, thirst, reproductive urge, position, and proximity to resources and mates. This comprehensive state representation enabled better decision-making.
    - **Reward Function:** The reward function was crafted to incentivize survival behaviours (e.g., finding food and water) and penalise negative states (e.g., starvation, dehydration). This guided the neural networks to prioritise actions that improved entity fitness.
3. **Efficient Algorithms:**
    - **Experience Replay:** Used a memory buffer to store past experiences, allowing the neural networks to learn from previous actions and their outcomes. This helped in stabilising the learning process and improving policy updates.
    - **Batch Processing:** Implemented mini-batch processing during training to optimise the computational load. By updating the neural networks using small batches of experiences, the simulation maintained high performance without compromising learning quality.

Conclusion

The challenges faced during the development of the Ecosystem Simulation Project were met with innovative and effective solutions. Dynamic resource management ensured a balanced and sustainable environment, while neural network tuning and efficient algorithms enhanced the realism and performance of entity behaviours. These efforts collectively contributed to a robust and dynamic simulation, providing valuable insights into ecosystem dynamics and evolutionary processes. The project demonstrates the potential for simulating complex biological systems and offers a foundation for future research and development in this field.

9. Future Enhancements

Potential future enhancements include:

1. **Enhanced AI:**
   - **More Sophisticated AI Models:** Implementing more advanced artificial intelligence models could significantly improve the decision-making capabilities of entities. Leveraging deep reinforcement learning techniques and incorporating recurrent neural networks (RNNs) or attention mechanisms could help entities better adapt to complex, dynamic environments and exhibit more realistic behaviours.
   - **Behavioural Diversity:** Introducing mechanisms to support a broader range of behaviours and learning strategies among entities, enabling them to develop unique survival tactics based on their experiences and environmental conditions.
2. **Complex Interactions:**
   - **Cooperative Behaviours:** Allowing entities to engage in cooperative behaviours, such as forming packs for hunting or sharing resources, could add a new layer of complexity to the simulation. This would also provide insights into social dynamics and group strategies within ecosystems.
   - **Competition and Predation:** Enhancing the predation and competition mechanics to reflect more realistic scenarios. This could involve developing detailed models for predator-prey dynamics, territory establishment, and resource competition.
   - **Ecosystem Dynamics:** Introducing additional species with different ecological roles (e.g., decomposers, pollinators) and modelling their interactions with existing entities to create a more holistic ecosystem simulation.
3. **Extended Visualisation:**
   - **Detailed Visualisation Tools:** Adding more comprehensive visualisation tools to analyse the simulation data. This could include heat maps for resource distribution, movement patterns of entities, and real-time graphs showing population dynamics and genetic variations.
   - **Interactive Dashboards:** Developing interactive dashboards that allow users to explore various metrics and visualise the relationships between different parameters and outcomes in the simulation.
   - **3D Visualisation:** Transitioning to a 3D visualisation framework to provide a more immersive experience and better spatial understanding of the ecosystem.
4. **User Customization:**
   - **Customizable Simulation Parameters:** Allowing users to customise various simulation parameters, such as environmental conditions, initial population characteristics, and mutation rates. This flexibility would enable users to explore different scenarios and study their impact on the ecosystem.
   - **Scenario-Based Simulations:** Introducing predefined scenarios or challenges that users can run to observe specific phenomena, such as

    climate change effects, introduction of invasive species, or resource depletion scenarios.
  - **Modding Support:** Providing support for user-created modifications (mods) to the simulation, allowing the community to introduce new features, species, or behaviours.
5. **Improved Performance and Scalability:**
   - **Parallel Computing:** Leveraging parallel computing techniques and hardware acceleration (e.g., GPU) to improve the simulation's performance and handle larger, more complex ecosystems in real-time.
   - **Optimization Algorithms:** Continually refining the optimization algorithms to reduce computational overhead and enhance the efficiency of the simulation.
   - **Distributed Simulation:** Exploring the possibility of distributed simulation frameworks that allow for large-scale simulations spanning multiple computational nodes.

Conclusion

The Ecosystem Simulation Project has demonstrated a robust and dynamic simulation environment, providing valuable insights into ecosystem dynamics and evolutionary processes. Future enhancements aimed at advancing AI capabilities, introducing complex interactions, extending visualisation tools, enabling user customization, and improving performance will further enhance the simulation's utility and realism. These improvements will not only contribute to scientific research but also offer educational and recreational opportunities for users interested in exploring the fascinating world of ecosystems.

10. Conclusion

The Ecosystem Simulation Project has successfully achieved its objectives, resulting in a dynamic and interactive simulation that effectively models ecosystem interactions and evolutionary processes. This project marks a significant advancement in the field of ecological modelling and artificial intelligence. The following points highlight the key achievements and future prospects of the project:

1. **Realistic and Adaptive Entity Behaviours:**
    - The integration of neural networks into the simulation allowed entities to exhibit realistic and adaptive behaviours. By utilising reinforcement learning, entities could make decisions based on their current state and environmental conditions, leading to more lifelike interactions and survival strategies.
    - Entities displayed a variety of behaviours, including seeking food and water, hunting prey, avoiding predators, and reproducing. These behaviours evolved over generations, providing valuable insights into how different traits and strategies can affect an entity's survival and reproductive success.
2. **Detailed and Interactive Environment:**
    - The simulation environment was meticulously designed, with a diverse range of tiles representing different terrain types, such as land, water, sand, and trees. This diversity allowed for complex interactions between entities and their surroundings.
    - The introduction of plants and their regrowth dynamics added another layer of realism, influencing the availability of resources and the distribution of entities within the environment.
3. **Comprehensive Visualization and Analysis Tools:**
    - The project included extensive visualisation tools that allowed users to monitor and analyse various aspects of the simulation. Real-time statistics and graphs provided insights into population dynamics, genetic variations, and behavioural patterns.
    - Interactive features, such as the ability to select and observe individual entities, further enhanced the user experience and facilitated a deeper understanding of the simulation's inner workings.
4. **Continuous Testing and Optimization:**
    - Rigorous testing and optimization were conducted throughout the project to ensure the stability and performance of the simulation. Various parameters, including learning rates, mutation probabilities, and environmental conditions, were fine-tuned to achieve optimal results.
    - The simulation was designed to be scalable, allowing for the inclusion of more complex behaviours and interactions in future iterations without compromising performance.
5. **Future Enhancements:**
    - While the current simulation provides a robust foundation, there are numerous opportunities for future enhancements. Potential improvements include the implementation of more sophisticated AI models, the introduction of cooperative behaviours and competition, the addition of detailed visualisation tools, and the provision of user customization options.

    ○ These enhancements will not only increase the realism and analytical capabilities of the simulation but also expand its applicability in various fields, such as ecological research, educational tools, and gaming.

Summary

In conclusion, the Ecosystem Simulation Project has successfully created a detailed and dynamic simulation environment that models the complexities of ecosystem interactions and evolutionary processes. The integration of neural networks has enabled entities to exhibit realistic and adaptive behaviours, providing valuable insights into ecosystem dynamics. Continuous testing and optimization have ensured a stable and performant simulation. Future enhancements, as outlined, hold the promise of further improving the simulation's realism and analytical capabilities, making it an invaluable tool for both research and education.

Appendices

Appendix A: Requirements.txt

makefile
Copy code

```
pygame==2.1.0
numpy==1.21.0
matplotlib==3.4.2
torch==1.9.0
pynoise==1.2.3
```

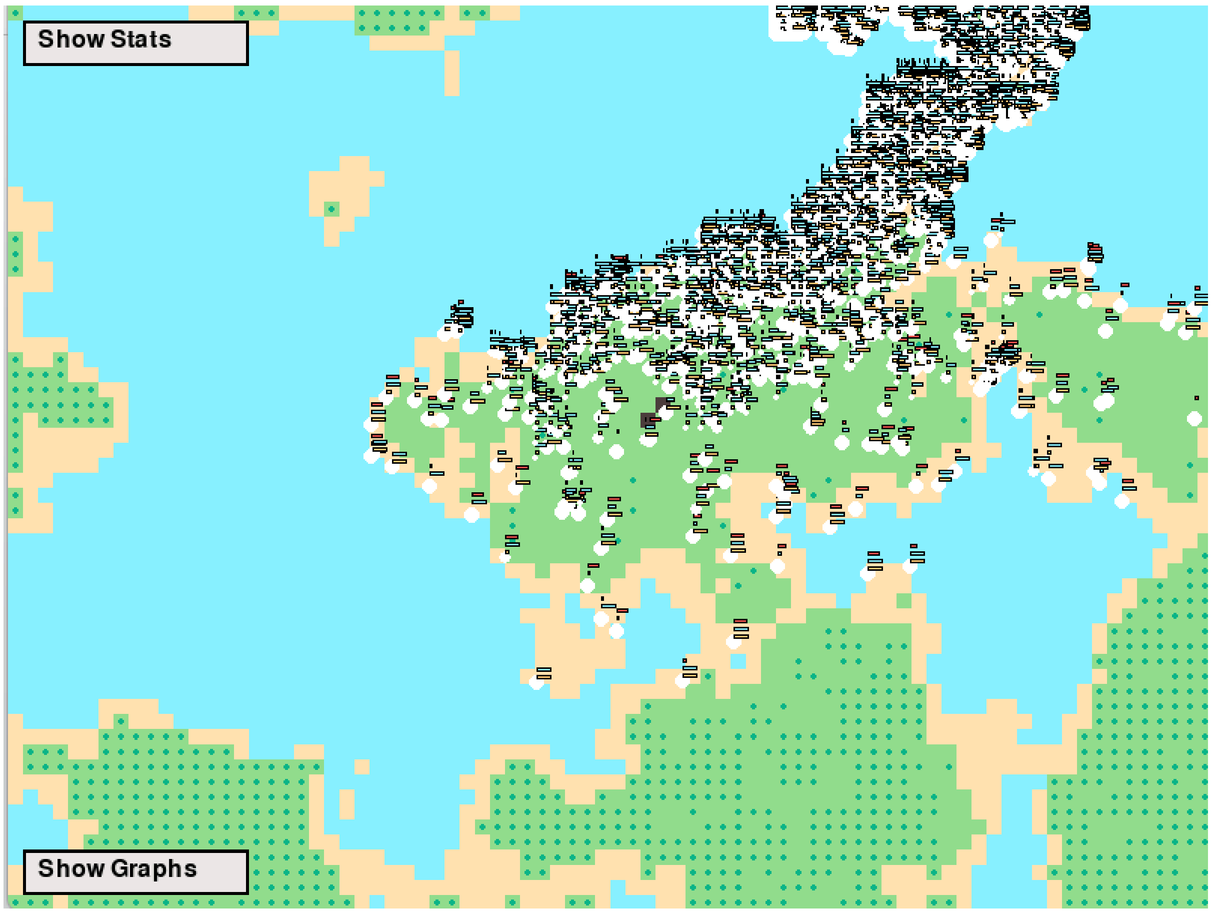
Appendix B: Key Functions and Classes

- `generate_map`: Generates the environment map using Perlin noise.
- `update_environment`: Updates the state of the environment, including plant regrowth.
- `draw_environment`: Renders the environment on the screen.
- `calculate_avg_genes`: Computes average traits of the entity population.
- `draw_stats`: Displays real-time statistics of the simulation.
- `plot_graphs`: Plots simulation data for analysis.

Appendix C: Simulation Parameters

- **Screen Dimensions:** `SCREEN_WIDTH = 800`, `SCREEN_HEIGHT = 600`
- **Map Dimensions:** `MAP_WIDTH = 1600`, `MAP_HEIGHT = 1200`
- **Tile Size:** `TILE_SIZE = 20`

- **Simulation Speed:** `FPS = 60`
- **Entity Attributes:** `REPRODUCTION_THRESHOLD = 100`, `HUNGER_THRESHOLD = 600`, `THIRST_THRESHOLD = 600`
- **Resource Parameters:** `PLANT_PROBABILITY = 0.001`, `REGROWTH_TIME = 500`
- **Neural Network Parameters:** `BATCH_SIZE = 64`, `ADULT_AGE = 100`


Show Stats
Show Graphs

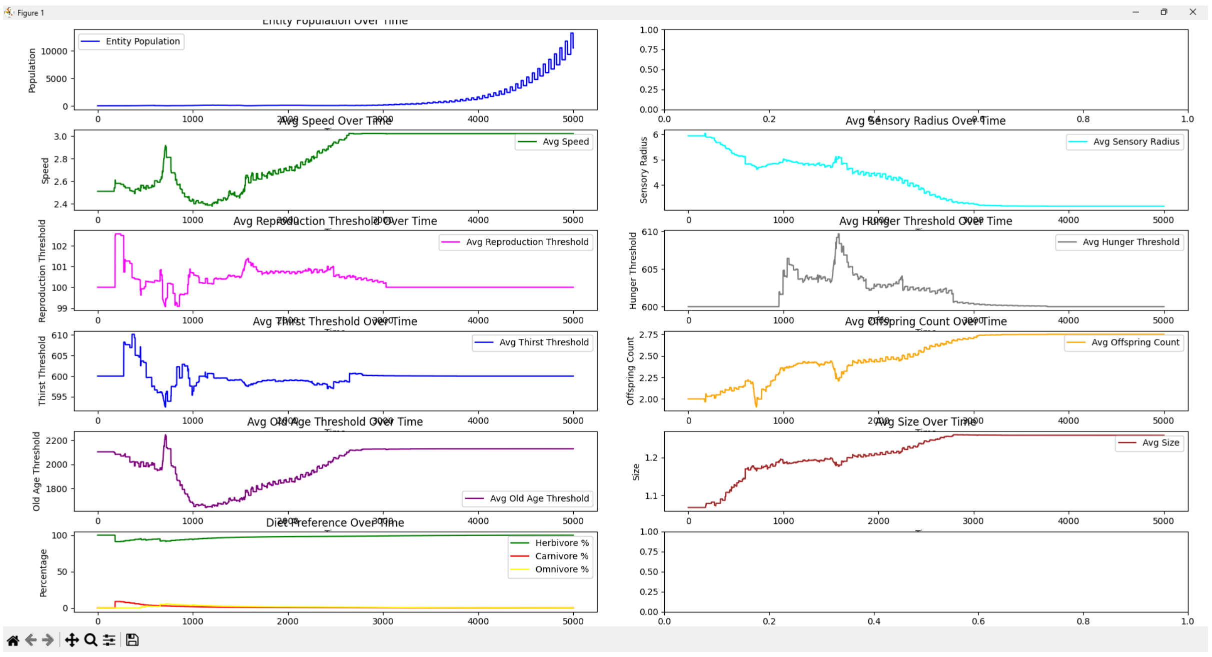


Figure 1
Entity Population Over Time
Entity Population
Population
Avg Speed Over Time
Avg Speed
Speed
Avg Sensory Radius Over Time
Avg Sensory Radius
Sensory Radius
Avg Reproduction Threshold Over Time
Avg Reproduction Threshold
Reproduction Threshold
Avg Hunger Threshold Over Time
Avg Hunger Threshold
Hunger Threshold
Avg Thirst Threshold Over Time
Avg Thirst Threshold
Thirst Threshold
Avg Offspring Count Over Time
Avg Offspring Count
Offspring Count
Avg Old Age Threshold Over Time
Avg Old Age Threshold
Old Age Threshold
Avg Size Over Time
Avg Size
Size
Diet Preference Over Time
Herbivore %
Carnivore %
Omnivore %
Percentage