Documentación Técnica del Simulador de Ecosistema

Programación Paralela

por

Phanor Castillo

Profesor

Jefferson Amado Penia

May 22, 2025

Universidad Javeriana Cali

Contents

1 Introducción

Contexto: Los modelos de dinámica poblacional son herramientas fundamentales en ecología teórica para entender las complejas interacciones entre depredadores y presas. Este proyecto implementa un simulador que captura estas relaciones mediante un enfoque basado en agentes.

1.1 Objetivos

El simulador desarrollado busca:

- Modelar las interacciones tróficas entre:
  - **Conejos (presa)**: Reproducción rápida, movimiento limitado
  - **Zorros (depredador)**: Dependencia de presas para sobrevivir
  - **Rocas**: Elementos estáticos que afectan la dinámica espacial
- Demostrar los patrones cíclicos característicos de sistemas depredador-presa, donde:
  - El aumento de presas permite el crecimiento de depredadores
  - La sobre-depredación lleva al colapso de ambas poblaciones
  - La recuperación reinicia el ciclo
- Analizar cómo parámetros clave afectan la estabilidad:
  - Tasa de reproducción (`GEN_PROC_RABBITS/FOXES`)
  - Resistencia al hambre (`GEN_FOOD_FOXES`)
  - Distribución espacial de recursos

1.2 Enfoque Técnico

La implementación combina:

- **Programación orientada a objetos**:
  - Clases para cada tipo de entidad (Criatura, Conejo, Zorro, Roca)
  - Sistema de grid para representar el espacio
- **Paralelización con OpenMP**:
  - Procesamiento eficiente de múltiples agentes
  - Optimización para grandes tamaños de grid
- **Visualización interactiva**:
  - Representación gráfica del estado del sistema
  - Controles para pausa/avance manual
  - Ajuste dinámico de parámetros

2 Arquitectura del Sistema

2.1 Diagrama de Clases

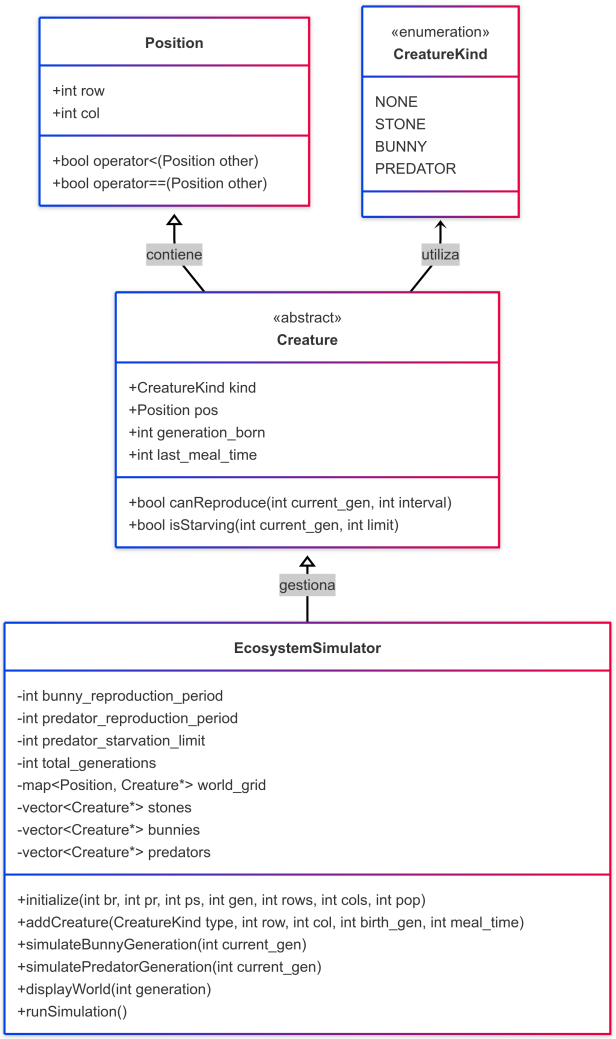

Figure 1: Diagrama de clases del simulador

2.2 Componentes Principales

- **Position**: Estructura para coordenadas (fila, columna)
- **Creature**: Clase base para todas las entidades
- **EcosystemSimulator**: Clase principal con la lógica de simulación

3 Implementación Detallada

3.1 Clase Creature

Representa cualquier entidad en el ecosistema con:

Listing 1: Definición de la clase Creature

```
class Creature {
public:
    CreatureKind kind;          // Tipo de criatura
    Position pos;               // Posicion actual
    int generation_born;        // Generacion de nacimiento
    int last_meal_time;         // Ultima vez que comio

    // Metodos
    bool canReproduce(int current_gen, int interval);
    bool isStarving(int current_gen, int limit);
};
```

3.2 Clase EcosystemSimulator

Contiene toda la lógica de simulación:

Listing 2: Estructura principal

```
class EcosystemSimulator {
private:
    // Parametros de simulacion
    int bunny_reproduction_period;
    int predator_reproduction_period;
    int predator_starvation_limit;
    int total_generations;

    // Estructuras de datos
    map<Position, shared_ptr<Creature>> world_grid;
    vector<shared_ptr<Creature>> stones, bunnies, predators;

    // Metodos principales
    void simulateBunnyGeneration(int current_gen);
    void simulatePredatorGeneration(int current_gen);
    void displayWorld(int generation);
};
```

4 Lógica de Simulación

4.1 Flujo Principal

1. Inicialización del ecosistema
2. Para cada generación:
   - Movimiento y reproducción de conejos
   - Movimiento, alimentación y reproducción de zorros
   - Resolución de conflictos
   - Actualización de estados
3. Visualización del estado actual

4.2 Algoritmos Clave

4.2.1 Selección de Movimiento

```
int move_choice = (current_generation + creature->pos.row
                + creature->pos.col) % possible_moves.size();
```

4.2.2 Resolución de Conflictos

Para conejos:

```
auto survivor = *max_element(..., [](..., ...) {
    return (current_gen - a->generation_born)
         < (current_gen - b->generation_born);
});
```

Para zorros:

```
sort(..., [](..., ...) {
    if (a->generation_born != b->generation_born)
        return a->generation_born < b->generation_born;
    return a->last_meal_time > b->last_meal_time;
});
```

5 Visualización y Controles

5.1 Interfaz de Usuario

5.2 Controles Interactivos

- **Pausa/Continuar**: Tecla 'p'

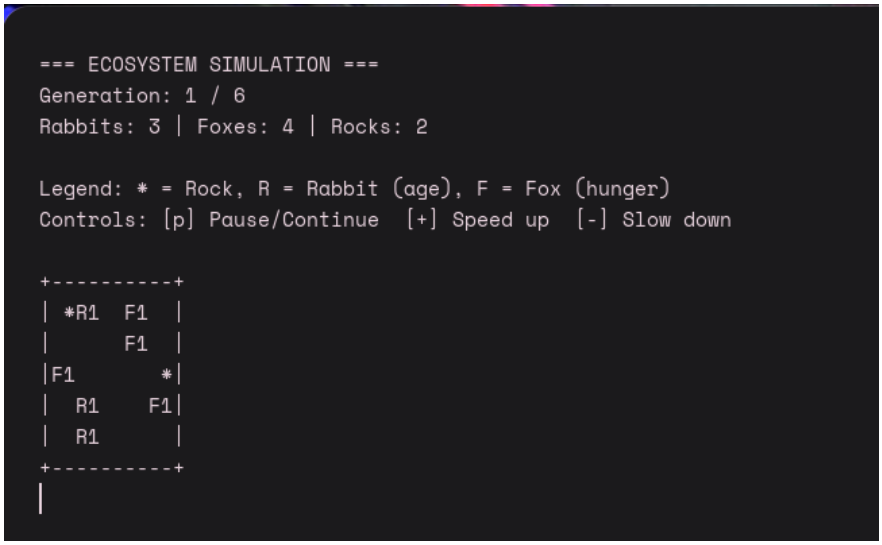

Figure 2: Interfaz del simulador durante la ejecución

- **Aumentar velocidad**: Tecla '+'
- **Disminuir velocidad**: Tecla '-'
- **Tiempo entre generaciones**: Ajustable entre 0.1s y 5s (4s default)

6 Compilación y Ejecución

6.1 Requisitos

- Compilador C++ compatible con C++11
- Biblioteca OpenMP para paralelización
- Sistema Unix/Linux para controles de terminal

6.2 Comandos

```
# Compilacion
g++ -fopenmp project.cpp -o ecosystem

# Ejecucion
./ecosystem
```

7 Conclusiones

- **Patrones poblacionales**: La simulación demuestra claramente los ciclos característicos de depredador-presa, donde:

  - El aumento de conejos precede al crecimiento de zorros
  - La depredación intensa reduce los conejos, llevando al posterior declive zorros
  - La recuperación de conejos reinicia el ciclo ecológico

- **Parámetros clave**: El modelo permite analizar cómo:

  - Los períodos de reproducción (`GEN_PROC_RABBITS/FOXES`)
  - El límite de hambre (`GEN_FOOD_FOXES`)

  afectan la estabilidad del sistema, desde oscilaciones regulares hasta colapsos poblacionales.

- **Elementos estáticos**: Las rocas influyen significativamente en:

  - Patrones de movimiento espacial
  - Creación de microhábitats y refugios
  - Distribución de encuentros depredador-presa

- **Implementación técnica**: La solución paralela con OpenMP permite:

  - Simulaciones a gran escala con eficiencia
  - Visualización interactiva del estado del sistema
  - Análisis cuantitativo de parámetros

Los resultados validan que modelos relativamente simples pueden capturar la esencia de interacciones ecológicas complejas, ofreciendo insights valiosos sobre los mecanismos que mantienen el equilibrio en sistemas naturales.