**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*



**BÁO CÁO BÀI TẬP LỚN MÔN**

**LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

### **ĐỀ TÀI: Interactive Simulation of Ecosystem Food Chains**

Mã lớp: 151965

Giaó viên hướng dẫn : TS. Nguyễn Thị Thu Trang

### NHÓM 5

|  |  |
| --- | --- |
| **Họ và tên thành viên** | **Mã số sinh viên** |
| Phạm Đức Ngự Bình | 20225696 |
| Nguyễn Công Bình | 20225695 |
| Nguyễn Minh Chiến | 20215000 |
| Trần Bá Công | 20215002 |
| Khổng Minh Cường | 20225603 |

Hà Nội, 12/2024

**PHÂN CÔNG CÔNG VIỆC**

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ và tên thành viên** | **Mã số sinh viên** | **Công việc** | **Tỉ lệ tham gia** |
| Phạm Đức Ngự Bình | 20225696 | -Nhóm trưởng  -Viết báo cáo  -Hoàn thiện class diagram và use cases diagram  -Quản lý repo nhóm  -Code WorldRenderer và test chương trình | 100% |
| Nguyễn Công Bình | 20225695 | Code các lớp Organism, Animal, Plant, Herbivore, Carnivore | 100% |
| Nguyễn Minh Chiến | 20215000 | Phụ GUI | 100% |
| Trần Bá Công | 20215002 | -Code các lớp World, Simulator  - Phụ các lớp Organism, Animal, Plant, Herbivore, Carnivore | 100% |
| Khổng Minh Cường | 20225603 | Code lớp Main | 100% |

**CHƯƠNG I. Phân tích đề tài**

1. **Yêu cầu của đề tài**
   1. **Mục đích**: Mô phỏng hệ sinh thái gồm thực vật, động vật ăn cỏ và động vật ăn thịt, giúp người dùng hiểu cách năng lượng được truyền qua các cấp độ dinh dưỡng và ảnh hưởng của sự thay đổi kích thước quần thể đối với sự ổn định của hệ sinh thái.
   2. **Yêu cầu thiết kế**: Xây dựng hệ thống mô phỏng hệ sinh thái, bao gồm:

* **Giao diện người dùng (GUI)**:
* **Menu** gồm lựa chọn :

+Tạo hệ sinh thái theo kịch bản cân bằng, vượt ngưỡng hoặc tuyệt chủng

+Nút Help để xem hướng dẫn mô phỏng và giải thích các kiến thức cơ bản

+Nút Quit để thoát chương trình với sự xác nhận của người dùng

* **Giao diện** điều chỉnh các thông số của mô phỏng như tỉ lệ sinh, tử và hiệu quả năng lượng
* **Mô phỏng**: Bản đồ ô lưới hình chữ nhật với thực vật, động vật ăn cỏ và động vật ăn thịt được đặt trong từng ô thể hiện vị trí. Các yếu tố được mô phỏng bao gồm:
* **Hiển thị**:

+**Thực vật**: Biểu tượng cây cối, không di chuyển nhưng tự động sinh sản sau một thời gian nhất định.

+**Động vật ăn cỏ**: Biểu tượng động vật ăn cỏ như cừu, dê …, di chuyển để tìm thực vật.

+**Động vật ăn thịt**: Biểu tượng động vật ăn thịt như sư tử, sói …, di chuyển để săn bắt động vật ăn cỏ.

* **Chuyển động**:

+Động vật ăn cỏ và động vật ăn thịt di chuyển ngẫu nhiên hoặc hướng về nguồn thức ăn.

+Động vật ăn cỏ di chuyển đến thực vật gần nhất, động vật ăn thịt di chuyển để săn động vật ăn cỏ.

* **Sinh sản**:

Các sinh vật sinh sản khi đạt đủ ngưỡng năng lượng. Sinh vật mới xuất hiện ở các ô trống liền kề.

* **Tương tác**:

+Thực vật sinh ra năng lượng qua quang hợp và bị động vật ăn cỏ tiêu thụ.

+Động vật ăn cỏ tiêu thụ thực vật để có năng lượng. Nếu không có thực vật để ăn, chúng mất năng lượng theo thời gian và có thể chết.

+Động vật ăn thịt săn động vật ăn cỏ để có năng lượng. Nếu không có con mồi, chúng mất năng lượng theo thời gian và có thể chết.

* **Chuyển giao năng lượng**:

Thực vật chuyển 10% năng lượng cho động vật ăn cỏ khi bị ăn, động vật ăn cỏ chuyển 10% năng lượng cho động vật ăn thịt.

* **Dân số và năng lượng**:

Cập nhật dân số của mỗi loài và năng lượng truyền giữa các cấp sinh vật theo thời gian.

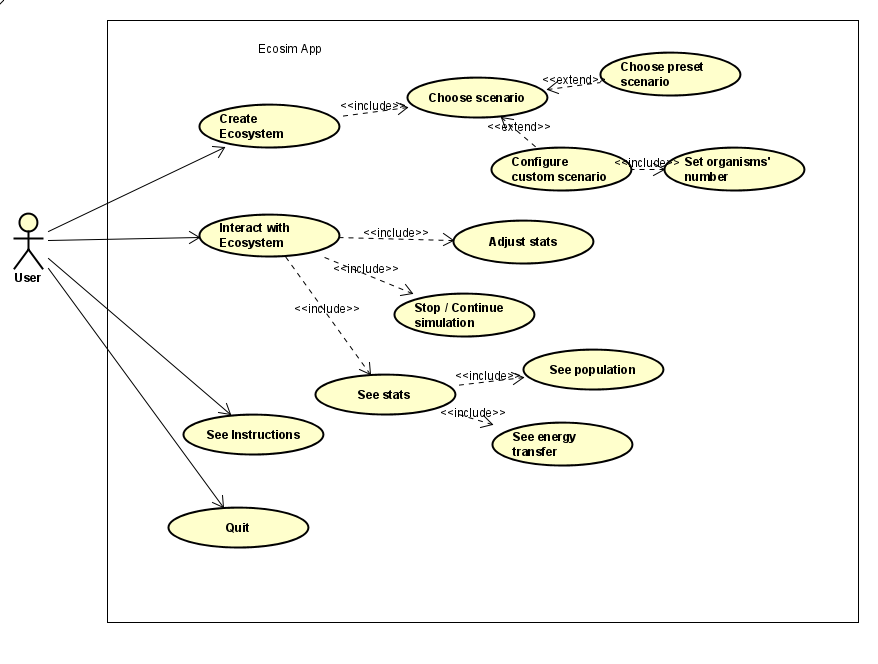
* **Tùy chỉnh**:

Điều chỉnh các thông số quần thể như tỉ lệ sinh, tỉ lệ tử, hiệu quả năng lượng.

1. **Hướng thực hiện đề tài**
   1. **Xây dựng thiét kế đề tài**
   2. **Cài đặt**
   3. **Kiểm thử**
   4. **Hoàn thiện, sửa lỗi**

**Chương II: Phân tích thiết kế đề tài**

1. **Biểu đồ use cases**

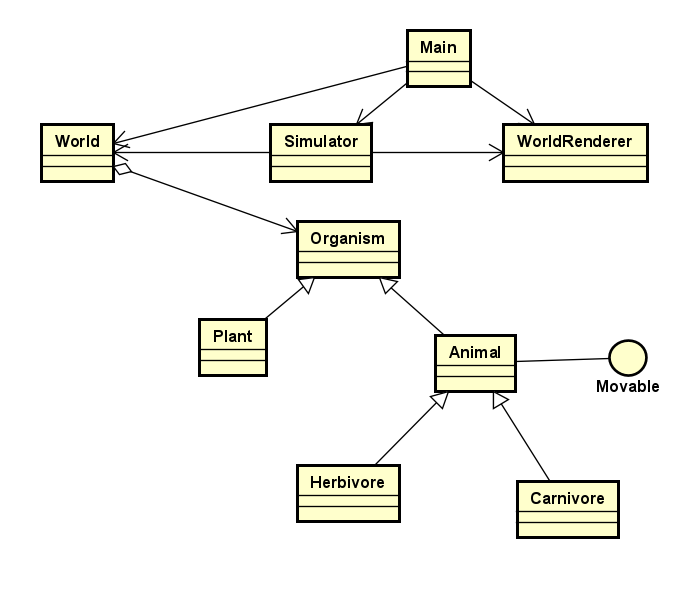


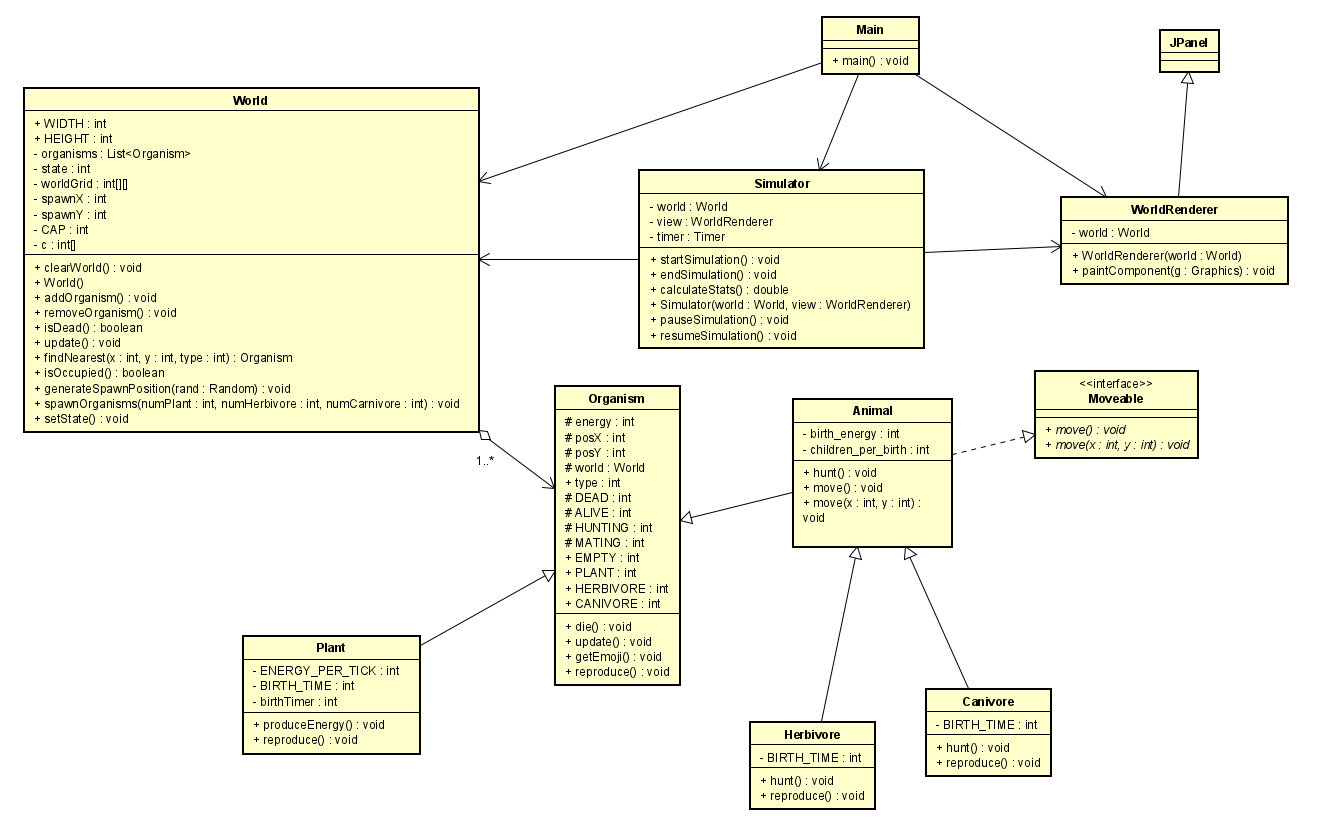
Theo yêu cầu đề tài, nhóm xây dựng **biểu đồ use case** mô tả sự tương tác giữa **Người dùng (User)** với phần mềm mô phỏng như sau :

* **Tạo hệ sinh thái (Create Ecosystem)**:
  + Người dùng có thể tạo một hệ sinh thái, bao gồm việc **Chọn kịch bản (Choose scenario)** bằng 1 trong 2 cách sau:
    - **Chọn kịch bản có sẵn (Choose preset scenario)**
    - **Tùy chỉnh kịch bản (Configure custom scenario** bằng cách **thiết lập số lượng từng loại sinh vật (Set organisms' number)**.
* **Tương tác với hệ sinh thái (Interact with Ecosystem)**:
  + Người dùng có thể tương tác với hệ sinh thái, bao gồm các chức năng:
    - **Điều chỉnh chỉ số (Adjust stats)**.
    - **Dừng / Tiếp tục mô phỏng (Stop / Continue simulation)**.
    - **Xem thống kê (See stats)**:
      * **Xem số lượng sinh vật (See population**
      * **Xem luồng năng lượng (See energy transfer)**
* **Xem hướng dẫn (See Instructions)**:
  + Người dùng có thể truy cập phần hướng dẫn sử dụng ứng dụng.
* **Thoát (Quit)**:
  + Người dùng có thể thoát khỏi ứng dụng

.

1. **Biểu đồ lớp**





### **2.1. Lớp Main**

* **Mối quan hệ**: Liên hệ với cả lớp Simulator và lớp WorldRenderer.
* **Chức năng**: Là lớp chạy chương trình , hiển thị giao diện và xử lý yêu cầu từ người dùng.

### **2.2. Lớp World**

* **Mối quan hệ**:
  + Liên hệ với lớp Simulator và Organism.
  + Có quan hệ kết tập với Organism.
* **Chức năng**: Đại diện cho thế giới có hình dạng lưới chữ nhật, gồm các thực thể sống.

### **2.3. Lớp Simulator**

* **Mối quan hệ**:
  + Liên hệ với World và WorldRenderer.
  + Tương tác với lớp Main.
* **Chức năng**: Xử lý logic mô phỏng, cập nhật trạng thái của hệ sinh thái.

### **2.4. Lớp WorldRenderer**

* **Mối quan hệ**:
  + Liên hệ với lớp Main.
* **Chức năng**: Hiển thị và vẽ thế giới mô phỏng lên màn hình.

### **2.5. Lớp Organism**

* **Mối quan hệ**: Là lớp **cha** của Plant và Animal.
* **Chức năng**: Đại diện cho tất cả thực thể sống trong hệ sinh thái.

### **2.6. Lớp Plant**

* **Mối quan hệ**: Kế thừa từ lớp Organism.
* **Chức năng**: Đại diện cho thực vật trong hệ sinh thái.

### **2.7. Lớp Animal**

* **Mối quan hệ**:
  + Kế thừa từ lớp Organism.
  + Có quan hệ realization với Movable.
  + Là lớp **cha** của Herbivore và Carnivore.
* **Chức năng**: Đại diện cho động vật, các thực thể có khả năng di chuyển, săn bắt và sinh sản.

### **2.8. Interface Movable**

* **Chức năng**: Đại diện cho hành động di chuyển, mà các lớp Animal sẽ triển khai (implement).

### **2.9. Lớp Herbivore và Carnivore**

* **Mối quan hệ**: Cả hai đều kế thừa từ lớp Animal.
* **Chức năng**:
  + **Herbivore**: Đại diện cho động vật ăn thực vật.
  + **Carnivore**: Đại diện cho động vật ăn thịt.

1. **Một số phương thức quan trọng**

* Lớp trừu tượng Organism:

package sim;

public abstract class Organism {

public static final int ***EMPTY*** = 0,

***PLANT*** = 1,

***HERBIVORE*** = 2,

***CARNIVORE*** = 3;

protected final int DEAD = 1, ALIVE = 0, HUNTING = 3, MATING = 4;

protected int energy;

protected int posX;

protected int posY;

protected World world;

protected int state;

protected int birthCooldown;

protected int type;

// Constructor

public Organism(int energy, int posX, int posY, World world, int type) {

this.energy = energy;

this.posX = posX;

this.posY = posY;

this.world = world;

this.state = ALIVE;

}

public void die() {

this.state = DEAD;

world.unoccupy(posX, posY);

}

public abstract void update();

public boolean isDead() {

return this.state == DEAD || this.energy <= 0;

}

public abstract void reproduce();

public abstract String getEmoji();

public int getPosX() {

return this.posX;

}

public int getPosY() {

return this.posY;

}

}

* Lớp Plant kế thừa lớp Organism, override các phương thức update(), reproduce(), getEmoji() :
* package sim;
* import java.util.ArrayList;
* import java.util.List;
* import java.util.Random;
* // Lớp Plant đại diện cho thực vật trong hệ sinh thái, kế thừa từ Organism
* public class Plant extends Organism {
* private final int ENERGY\_PER\_TICK = 100; // Lượng năng lượng sản xuất được mỗi tick (chu kỳ)
* private final int BIRTH\_TIME = 10;       // Thời gian hồi chiêu giữa các lần sinh sản
* /\*\*
* \* Constructor: Khởi tạo một đối tượng Plant với năng lượng ban đầu, vị trí và tham chiếu tới thế giới.
* \*
* \* @param energy Năng lượng ban đầu của cây
* \* @param posX   Tọa độ x của cây trong thế giới
* \* @param posY   Tọa độ y của cây trong thế giới
* \* @param world  Tham chiếu tới đối tượng World
* \*/
* public Plant(int energy, int posX, int posY, World world) {
* super(energy, posX, posY, world, Organism.PLANT); // Gọi constructor của lớp cha Organism
* birthCooldown = BIRTH\_TIME; // Khởi tạo thời gian hồi chiêu sinh sản
* }
* /\*\*
* \* Phương thức sản xuất năng lượng thông qua quá trình quang hợp.
* \* Năng lượng của cây tăng lên mỗi chu kỳ và cập nhật vào chỉ số tổng của thế giới.
* \*/
* public void produceEnergy() {
* this.energy += ENERGY\_PER\_TICK; // Tăng năng lượng của cây
* world.c[0] += ENERGY\_PER\_TICK;  // Cập nhật tổng năng lượng nhận được của thực vật trong thế giới
* }
* /\*\*
* \* Phương thức sinh sản của thực vật.
* \* Cây sẽ tạo ra cây con ở một ô kề bên nếu đủ điều kiện và có vị trí trống.
* \*/
* @Override
* public void reproduce() {
* // Giảm thời gian hồi chiêu sinh sản
* if (birthCooldown > 0) {
* birthCooldown--;
* }
* // Nếu hết thời gian hồi chiêu và số lượng sinh vật trong thế giới chưa đạt giới hạn
* else if (world.getOrganisms().size() < world.CAP) {
* // Các hướng có thể sinh sản: Trái, Phải, Lên, Xuống
* int[] dx = { -1, 1, 0, 0 }; // Hướng di chuyển theo trục x
* int[] dy = { 0, 0, -1, 1 }; // Hướng di chuyển theo trục y
* Random rand = new Random();
* List<Integer> choices = new ArrayList<Integer>();
* // Thêm các hướng vào danh sách lựa chọn
* choices.add(0); // Trái
* choices.add(1); // Phải
* choices.add(2); // Lên
* choices.add(3); // Xuống
* int choice, reproduceX, reproduceY;
* // Lựa chọn ngẫu nhiên một ô kề bên còn trống để sinh cây con
* do {
* choice = rand.nextInt(choices.size());
* Integer c = choices.get(choice);
* reproduceX = this.posX + dx[c];
* reproduceY = this.posY + dy[c];
* choices.remove(c); // Loại bỏ hướng đã chọn khỏi danh sách
* } while (!choices.isEmpty() && world.isOccupied(reproduceX, reproduceY));
* // Nếu tìm được vị trí trống, tạo một cây con ở vị trí đó
* if (!world.isOccupied(reproduceX, reproduceY)) {
* world.addOrganism(new Plant(100, reproduceX, reproduceY, world)); // Tạo cây mới
* world.occupy(reproduceX, reproduceY, Organism.PLANT); // Đánh dấu ô đã bị chiếm
* birthCooldown = BIRTH\_TIME; // Reset thời gian hồi chiêu sinh sản
* }
* }
* }
* /\*\*
* \* Phương thức cập nhật trạng thái của cây trong mỗi chu kỳ.
* \* Gồm hai hoạt động: quang hợp để sản xuất năng lượng và sinh sản nếu đủ điều kiện.
* \*/
* @Override
* public synchronized void update() {
* // Nếu cây đã chết thì không làm gì cả
* if (isDead()) {
* return;
* }
* // Sản xuất năng lượng và thử sinh sản
* this.produceEnergy();
* this.reproduce();
* }
* /\*\*
* \* Phương thức trả về emoji biểu diễn cây trong thế giới.
* \*
* \* @return Chuỗi emoji biểu diễn cho cây.
* \*/
* @Override
* public String getEmoji() {
* return "🌱"; // Biểu tượng cây non
* }
* }

* Lớp Animal cũng là lớp trừu tượng kế thừa lớp Organism, override phương thức update(), đồng thời riêng lớp này implement interface Movable:

package sim;

public interface Movable {

void move(int x, int y);

void move();

}

package sim;

import java.util.Random;

public abstract class Animal extends Organism implements Movable {

// Constructor

public Animal(int x, int y, int initialEnergy, World world, int type) {

super(initialEnergy, x, y, world, type);

}

// Di chuyển đến một ô mới trong thế giới

public void move(int destX, int destY) {

int x = posX, y = posY;

if (x == destX && y == destY)

return;

int dx = destX - x, dy = destY - y;

if (dx > 0)

x++;

else if (dx < 0)

x--;

if (dy > 0)

y++;

else if (dy < 0)

y--;

// Kiểm tra ô trống

if (world.isOccupied(x, y)) {

return;

}

world.unoccupy(this.posX, this.posY); //Cập nhật ô

this.posX = x;

this.posY = y;

world.occupy(x, y, type);

}

// Giảm năng lượng do mất năng lượng qua thời gian

public void decayEnergy() {

this.energy -= 3; // Giảm năng lượng mỗi tick (có thể thay đổi tùy theo mô phỏng)

}

public abstract void hunt();

*@Override*

public synchronized void update() {

this.decayEnergy();

if (this.isDead()) {

this.die(); // Kiểm tra xem đã chêt hay chưa

return;

}

if(state == ALIVE) {

Random rand = new Random();

if(energy > 120 )

state = rand.nextInt((MATING - HUNTING) + 1) + HUNTING;

else state = HUNTING;

}

hunt(); // Săn

reproduce(); // Đẻ

}

*@Override*

public void move() {

// Tạo random mới

Random rand = new Random();

int[] dx = { -1, 1, 0, 0 }; // hướng ngang

int[] dy = { 0, 0, -1, 1 }; // hướng dọc

// Pick a random direction

int direction = rand.nextInt(4); // chọn 1 trong 4 lựa chọn

// Tính vị trí mới

int newX = this.posX + dx[direction];

int newY = this.posY + dy[direction];

// Kiểm tra biên

if (newX >= 0 && newX < world.WIDTH && newY >= 0 && newY < world.HEIGHT) {

move(newX, newY); // Di chuyển

}

}

public boolean isHunting() {

return this.state == HUNTING ;

}

public boolean isMating() {

return this.state == MATING ;

}

}

-Hai lớp Carnivore và Herbivore kế thừa lớp Animal, override các phương thức hunt() và reproduce():

package sim;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

// Lớp Carnivore đại diện cho động vật ăn thịt, kế thừa từ lớp Animal

public class Carnivore extends Animal {

    // Thời gian hồi chiêu giữa các lần sinh sản

    protected final int BIRTH\_TIME = 50;

    /\*

     Constructor: Khởi tạo một Carnivore với vị trí, năng lượng ban đầu và tham chiếu tới thế giới.

     \*/

    public Carnivore(int x, int y, int initialEnergy, World world) {

        super(x, y, initialEnergy, world, Organism.CARNIVORE);

    }

    /\*\*

     \* Phương thức săn mồi của Carnivore.

     \* Carnivore sẽ tìm động vật ăn cỏ gần nhất và di chuyển đến để ăn.

     \* Nếu động vật ăn cỏ ở ô kề bên, nó sẽ tiêu thụ và tăng năng lượng.

     \*/

    @Override

    public void hunt() {

        // Nếu đang trong trạng thái giao phối thì không săn mồi

        if (isMating()) return;

        // Tìm động vật ăn cỏ gần nhất

        Organism herbivore = world.findNearest(this.posX, this.posY, Organism.HERBIVORE);

        if (herbivore != null && !herbivore.isDead()) {

            // Tính khoảng cách giữa Carnivore và Herbivore

            int dx = herbivore.posX - this.posX, dy = herbivore.posY - this.posY;

            // Nếu Herbivore ở ô kề bên (xung quanh), ăn Herbivore

            if (Math.abs(dx) <= 1 && Math.abs(dy) <= 1) {

                world.c[2] += herbivore.energy / 10; // Cập nhật chỉ số của thế giới

                this.energy += herbivore.energy / 10; // Tăng năng lượng cho Carnivore

                herbivore.die(); // Động vật ăn cỏ bị tiêu thụ

                state = ALIVE; // Cập nhật trạng thái của Carnivore là còn sống

            }

            // Di chuyển tới vị trí của Herbivore

            move(herbivore.posX, herbivore.posY);

        } else {

            // Nếu không tìm thấy Herbivore, di chuyển ngẫu nhiên

            move();

        }

    }

    /\*\*

     \* Phương thức sinh sản của Carnivore.

     \* Carnivore sẽ tạo ra một con non nếu đủ năng lượng và đã hết thời gian hồi chiêu sinh sản.

     \* Nếu không thể sinh sản, nó sẽ thực hiện di chuyển.

     \*/

package sim;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;

// Lớp Herbivore đại diện cho động vật ăn cỏ, kế thừa từ lớp Animal

public class Herbivore extends Animal {

    // Thời gian chờ giữa các lần sinh sản

    private static final int BIRTH\_TIME = 30;

    // Constructor: Khởi tạo một Herbivore với vị trí, năng lượng ban đầu và tham chiếu tới thế giới

    public Herbivore(int x, int y, int initialEnergy, World world) {

        super(x, y, initialEnergy, world, Organism.HERBIVORE);

    }

    /\*\*

     \* Phương thức sinh sản của Herbivore.

     \* Herbivore sẽ tạo ra một con non nếu đủ năng lượng và đã hết thời gian hồi chiêu sinh sản.

     \* Nếu không thể sinh sản, nó sẽ thực hiện di chuyển.

     \*/

    @Override

    public void reproduce() {

        // Nếu đang trong trạng thái săn mồi thì không thể sinh sản

        if (isHunting()) return;

        // Giảm thời gian hồi chiêu sinh sản

        if (birthCooldown > 0) {

            birthCooldown--;

        }

        // Nếu đủ điều kiện sinh sản

        else if (this.energy > 120 && world.getOrganisms().size() < world.CAP) {

            // Mảng biểu diễn 4 hướng di chuyển: Trái, Phải, Lên, Xuống

            int[] dirX = { -1, 1, 0, 0 };

            int[] dirY = { 0, 0, -1, 1 };

            Random rand = new Random();

            List<Integer> choices = new ArrayList<Integer>();

            // Thêm các lựa chọn hướng vào danh sách

            choices.add(0); // Trái

            choices.add(1); // Phải

            choices.add(2); // Lên

            choices.add(3); // Xuống

            int choice, reproduceX, reproduceY;

            // Tìm vị trí trống để sinh sản

            do {

                choice = rand.nextInt(choices.size());

                Integer c = choices.get(choice);

                reproduceX = this.posX + dirX[c];

                reproduceY = this.posY + dirY[c];

                choices.remove(c); // Loại bỏ hướng đã chọn

            } while (!choices.isEmpty() && world.isOccupied(reproduceX, reproduceY));

            // Nếu tìm thấy vị trí trống, tạo một Herbivore mới

            if (!world.isOccupied(reproduceX, reproduceY)) {

                // Thêm Herbivore con vào thế giới

                world.addOrganism(new Herbivore(reproduceX, reproduceY, 100, world));

                this.energy -= 40; // Giảm năng lượng sau khi sinh sản

                world.occupy(reproduceX, reproduceY, Organism.HERBIVORE);

                birthCooldown = BIRTH\_TIME; // Reset thời gian hồi chiêu sinh sản

            }

        } else {

            // Nếu không thể sinh sản, thực hiện di chuyển

            move();

        }

        state = ALIVE; // Cập nhật trạng thái là còn sống

    }

    /\*\*

     \* Phương thức săn mồi của Herbivore.

     \* Herbivore sẽ tìm kiếm thực vật gần nhất để ăn, nếu không sẽ di chuyển.

     \*/

    @Override

    public void hunt() {

        // Nếu đang trong trạng thái giao phối thì không săn mồi

        if (isMating()) return;

        // Tìm thực vật gần nhất từ vị trí hiện tại

        Organism plant = world.findNearest(this.posX, this.posY, Organism.PLANT);

        if (plant != null && !plant.isDead()) {

            // Tính khoảng cách giữa Herbivore và thực vật

            int dx = plant.posX - this.posX, dy = plant.posY - this.posY;

            // Nếu thực vật ở ô kề bên (xung quanh), ăn thực vật

            if (Math.abs(dx) <= 1 && Math.abs(dy) <= 1) {

                // Tăng năng lượng cho Herbivore và cập nhật chỉ số của thế giới

                world.c[1] += plant.energy / 10;

                this.energy += plant.energy / 10;

                plant.die(); // Thực vật bị ăn và chết

                state = ALIVE; // Cập nhật trạng thái là còn sống

            }

            // Di chuyển tới vị trí của thực vật

            move(plant.posX, plant.posY);

        } else {

            // Nếu không tìm thấy thực vật, di chuyển ngẫu nhiên

            move();

        }

    }

    /\*\*

     \* Phương thức biểu diễn Herbivore dưới dạng emoji.

     \*

     \* @return Chuỗi emoji đại diện cho Herbivore.

     \*/

    @Override

    public String getEmoji() {

        return "🦌"; // Goat emoji

    }

}

* Lớp World:

package sim;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.Random;

public class World {

    // Các hằng số biểu thị trạng thái của thế giới

    private final int DEAD = 0, HEALTHY = 1, CLEARED = 2;

    // Kích thước của thế giới (chiều rộng và chiều cao)

    public final int WIDTH = 30;

    public final int HEIGHT = 30;

    // Giới hạn số lượng thực thể trong thế giới

    public final int CAP = 500;

    // Danh sách các thực thể (Organisms) trong thế giới

    private List<Organism> organisms;

    // Lưới (grid) mô phỏng không gian của thế giới

    private int[][] worldGrid;

    // Trạng thái hiện tại của thế giới

    private int state;

    // Mảng c lưu trữ các giá trị tạm thời (chưa được sử dụng cụ thể)

    protected int[] c = new int[3];

    // Tọa độ để sinh ra thực thể

    private int spawnX, spawnY;

    // Constructor: Khởi tạo lưới và danh sách thực thể

    public World() {

        this.worldGrid = new int[WIDTH][HEIGHT];

        this.organisms = new ArrayList<>();

        this.state = HEALTHY; // Trạng thái ban đầu là khỏe mạnh

    }

    // Phương thức trả về số lượng cây (Plants) trong thế giới

    public long getPlantsCount() {

        return organisms.stream().filter(o -> o instanceof Plant).count();

    }

    // Phương thức trả về số lượng động vật ăn cỏ (Herbivores)

    public long getHerbivoresCount() {

        return organisms.stream().filter(o -> o instanceof Herbivore).count();

    }

    // Phương thức trả về số lượng động vật ăn thịt (Carnivores)

    public long getCarnivoresCount() {

        return organisms.stream().filter(o -> o instanceof Carnivore).count();

    }

    // Getter: Lấy danh sách thực thể

    public List<Organism> getOrganisms() {

        return organisms;

    }

    // Kiểm tra thế giới đã chết hay chưa

    public boolean isDead() {

        return state == DEAD;

    }

    // Thiết lập trạng thái cho thế giới

    public void setState(int state) {

        this.state = state;

    }

    // Khởi tạo các thực thể ban đầu trong thế giới

    public void initializeOrganisms(List<Organism> initialOrganisms) {

        this.organisms.clear();

        this.organisms.addAll(initialOrganisms);

    }

    // Cập nhật trạng thái của thế giới và các thực thể

    public void update() {

        // Nếu không còn thực thể nào, trạng thái chuyển sang DEAD

        if (organisms.isEmpty()) {

            this.state = DEAD;

            return;

        }

        System.out.println("World IS ALIVE with " + organisms.size() + " organisms");

        // Dùng iterator để loại bỏ các thực thể đã chết một cách an toàn

        Iterator<Organism> iterator = organisms.iterator();

        while (iterator.hasNext()) {

            Organism o = iterator.next();

            if (o.isDead()) {

                System.out.println(o + " is dead");

                iterator.remove();

            }

        }

        // Cập nhật trạng thái cho các thực thể còn lại

        for (Organism o : organisms.toArray(new Organism[0])) {

            o.update();

        }

    }

    // Thêm một thực thể mới vào thế giới

    public void addOrganism(Organism organism) {

        this.organisms.add(organism);

        System.out.println(organism + " added");

    }

    // Xóa một thực thể khỏi thế giới

    public void removeOrganism(Organism organism) {

        System.out.println(organism + " removed");

        this.organisms.remove(organism);

    }

    // Tìm thực thể gần nhất từ tọa độ (x, y) với loại được chỉ định (Plant, Herbivore, Carnivore)

    public Organism findNearest(int x, int y, int type) {

        Organism nearest = null;

        double minDistance = Double.MAX\_VALUE;

        for (Organism organism : organisms) {

            double distance;

            if (organism instanceof Plant && type == Organism.PLANT) {

                distance = Math.sqrt(Math.pow(x - organism.getPosX(), 2) + Math.pow(y - organism.getPosY(), 2));

                if (distance < minDistance) {

                    minDistance = distance;

                    nearest = organism;

                }

            } else if (organism instanceof Herbivore && type == Organism.HERBIVORE) {

                distance = Math.sqrt(Math.pow(x - organism.getPosX(), 2) + Math.pow(y - organism.getPosY(), 2));

                if (distance < minDistance) {

                    minDistance = distance;

                    nearest = organism;

                }

            } else if (organism instanceof Carnivore && type == Organism.CARNIVORE) {

                distance = Math.sqrt(Math.pow(x - organism.getPosX(), 2) + Math.pow(y - organism.getPosY(), 2));

                if (distance < minDistance) {

                    minDistance = distance;

                    nearest = organism;

                }

            }

        }

        return nearest;

    }

    // Sinh ra thực thể ngẫu nhiên trong thế giới

    public void spawnOrganisms(int numPlant, int numHerbivore, int numCarnivore) {

        Random rand = new Random();

        Arrays.fill(c, 0);

        // Sinh cây (Plants)

        for (int i = 0; i < numPlant; i++) {

            generateSpawnPosition(rand);

            Plant newPlant = new Plant(100, spawnX, spawnY, this);

            addOrganism(newPlant);

            occupy(spawnX, spawnY, Organism.PLANT);

        }

        // Sinh động vật ăn cỏ (Herbivores)

        for (int i = 0; i < numHerbivore; i++) {

            generateSpawnPosition(rand);

            Herbivore newHerbivore = new Herbivore(spawnX, spawnY, 100, this);

            addOrganism(newHerbivore);

            occupy(spawnX, spawnY, Organism.HERBIVORE);

        }

        // Sinh động vật ăn thịt (Carnivores)

        for (int i = 0; i < numCarnivore; i++) {

            generateSpawnPosition(rand);

            Carnivore newCarnivore = new Carnivore(spawnX, spawnY, 100, this);

            addOrganism(newCarnivore);

            occupy(spawnX, spawnY, Organism.CARNIVORE);

        }

    }

    // Tạo vị trí ngẫu nhiên để sinh ra thực thể

    private void generateSpawnPosition(Random rand) {

        do {

            spawnX = rand.nextInt(WIDTH);

            spawnY = rand.nextInt(HEIGHT);

        } while (isOccupied(spawnX, spawnY));

    }

    // Kiểm tra vị trí (x, y) có bị chiếm đóng không

    public boolean isOccupied(int x, int y) {

        return x < 0 || x >= WIDTH || y < 0 || y >= HEIGHT || worldGrid[x][y] != Organism.EMPTY;

    }

    // Đánh dấu vị trí (x, y) là đã bị chiếm đóng

    public void occupy(int x, int y, int type) {

        worldGrid[x][y] = type;

    }

    // Giải phóng vị trí (x, y)

    public void unoccupy(int x, int y) {

        worldGrid[x][y] = Organism.EMPTY;

    }

    // Xóa toàn bộ dữ liệu trong thế giới

    public void clearWorld() {

        organisms.clear();

        state = CLEARED;

    }

}

-Lớp World:

package sim;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

import java.util.Random;

public class World {

    private final int DEAD = 0, HEALTHY = 1, CLEARED = 2; // Trạng thái của thế giới

    public final int WIDTH = 30; // Chiều rộng của thế giới

    public final int HEIGHT = 30; // Chiều cao của thế giới

    public final int CAP = 500; // Giới hạn số lượng sinh vật trong thế giới

    private List<Organism> organisms; // Danh sách các sinh vật hiện có trong thế giới

    private int[][] worldGrid; // Ma trận lưu trạng thái của các ô trên lưới

    private int state; // Trạng thái hiện tại của thế giới (DEAD, HEALTHY, CLEARED)

    protected int[] c = new int[3]; // Mảng theo dõi năng lượng tiêu thụ hoặc sinh ra của các sinh vật

    private int spawnX, spawnY; // Tọa độ tạm thời để spawn sinh vật

    public World() {

        this.worldGrid = new int[WIDTH][HEIGHT]; // Khởi tạo lưới

        this.organisms = new ArrayList<>(); // Khởi tạo danh sách sinh vật

        this.state = HEALTHY; // Đặt trạng thái mặc định là "HEALTHY"

    }

    // Phương thức trả về số lượng cây (Plants)

    public long getPlantsCount() {

        return organisms.stream().filter(o -> o instanceof Plant).count();

    }

    // Phương thức trả về số lượng động vật ăn cỏ (Herbivores)

    public long getHerbivoresCount() {

        return organisms.stream().filter(o -> o instanceof Herbivore).count();

    }

    // Phương thức trả về số lượng động vật ăn thịt (Carnivores)

    public long getCarnivoresCount() {

        return organisms.stream().filter(o -> o instanceof Carnivore).count();

    }

    // Lấy danh sách các sinh vật trong thế giới

    public List<Organism> getOrganisms() {

        return organisms;

    }

    // Kiểm tra xem thế giới đã chết hay chưa

    public boolean isDead() {

        return state == DEAD;

    }

    // Đặt trạng thái mới cho thế giới

    public void setState(int state) {

        this.state = state;

    }

    // Khởi tạo danh sách sinh vật từ một danh sách đã có

    public void initializeOrganisms(List<Organism> initialOrganisms) {

        this.organisms.clear(); // Xóa danh sách hiện tại

        this.organisms.addAll(initialOrganisms); // Thêm các sinh vật mới

    }

    // Cập nhật trạng thái của thế giới và các sinh vật bên trong

    public void update() {

        if (organisms.isEmpty()) { // Nếu không còn sinh vật nào, đặt trạng thái là DEAD

            this.state = DEAD;

            return;

        }

        System.out.println("World IS ALIVE with " + organisms.size() + " organisms");

        // Sử dụng Iterator để xóa an toàn các sinh vật đã chết trong khi duyệt danh sách

        Iterator<Organism> iterator = organisms.iterator();

        while (iterator.hasNext()) {

            Organism o = iterator.next();

            if (o.isDead()) {

                System.out.println(o + " is dead");

                iterator.remove(); // Xóa sinh vật đã chết khỏi danh sách

            }

        }

        // Cập nhật trạng thái cho từng sinh vật

        for (Organism o : organisms.toArray(new Organism[0])) { // Tạo bản sao để tránh lỗi chỉnh sửa đồng thời

            o.update();

        }

    }

    // Thêm một sinh vật mới vào thế giới

    public void addOrganism(Organism organism) {

        this.organisms.add(organism);

        System.out.println(organism + " added");

    }

    // Xóa một sinh vật khỏi thế giới

    public void removeOrganism(Organism organism) {

        System.out.println(organism + " removed");

        this.organisms.remove(organism);

    }

    // Tìm sinh vật gần nhất theo loại (Plant, Herbivore, Carnivore)

    public Organism findNearest(int x, int y, int type) {

        Organism nearest = null;

        double minDistance = Double.MAX\_VALUE;

        for (Organism organism : organisms) {

            double distance;

            if (organism instanceof Plant && type == Organism.PLANT) {

                distance = Math.sqrt(Math.pow(x - organism.getPosX(), 2) + Math.pow(y - organism.getPosY(), 2));

                if (distance < minDistance) {

                    minDistance = distance;

                    nearest = organism;

                }

            } else if (organism instanceof Herbivore && type == Organism.HERBIVORE) {

                distance = Math.sqrt(Math.pow(x - organism.getPosX(), 2) + Math.pow(y - organism.getPosY(), 2));

                if (distance < minDistance) {

                    minDistance = distance;

                    nearest = organism;

                }

            } else if (organism instanceof Carnivore && type == Organism.CARNIVORE) {

                distance = Math.sqrt(Math.pow(x - organism.getPosX(), 2) + Math.pow(y - organism.getPosY(), 2));

                if (distance < minDistance) {

                    minDistance = distance;

                    nearest = organism;

                }

            }

        }

        return nearest; // Trả về sinh vật gần nhất tìm được

    }

    // Spawn các sinh vật ban đầu vào thế giới

    public void spawnOrganisms(int numPlant, int numHerbivore, int numCarnivore) {

        Random rand = new Random();

        Arrays.fill(c, 0); // Reset mảng năng lượng

        // Spawn các cây

        for (int i = 0; i < numPlant; i++) {

            generateSpawnPosition(rand);

            Plant newPlant = new Plant(100, spawnX, spawnY, this);

            addOrganism(newPlant);

            occupy(spawnX, spawnY, Organism.PLANT);

        }

        // Spawn động vật ăn cỏ

        for (int i = 0; i < numHerbivore; i++) {

            generateSpawnPosition(rand);

            Herbivore newHerbivore = new Herbivore(spawnX, spawnY, 100, this);

            addOrganism(newHerbivore);

            occupy(spawnX, spawnY, Organism.HERBIVORE);

        }

        // Spawn động vật ăn thịt

        for (int i = 0; i < numCarnivore; i++) {

            generateSpawnPosition(rand);

            Carnivore newCarnivore = new Carnivore(spawnX, spawnY, 100, this);

            addOrganism(newCarnivore);

            occupy(spawnX, spawnY, Organism.CARNIVORE);

        }

    }

    // Tạo tọa độ ngẫu nhiên để spawn sinh vật

    private void generateSpawnPosition(Random rand) {

        do {

            spawnX = rand.nextInt(WIDTH);

            spawnY = rand.nextInt(HEIGHT);

        } while (isOccupied(spawnX, spawnY));

    }

    // Kiểm tra xem một ô trong lưới có bị chiếm hay không

    public boolean isOccupied(int x, int y) {

        return x < 0 || x >= WIDTH || y < 0 || y >= HEIGHT || worldGrid[x][y] != Organism.EMPTY;

    }

    // Đánh dấu một ô trên lưới là bị chiếm

    public void occupy(int x, int y, int type) {

        worldGrid[x][y] = type;

    }

    // Giải phóng một ô trên lưới

    public void unoccupy(int x, int y) {

        worldGrid[x][y] = Organism.EMPTY;

    }

    // Xóa toàn bộ sinh vật trong thế giới và đặt trạng thái là CLEARED

    public void clearWorld() {

        organisms.clear();

        state = CLEARED;

    }

}