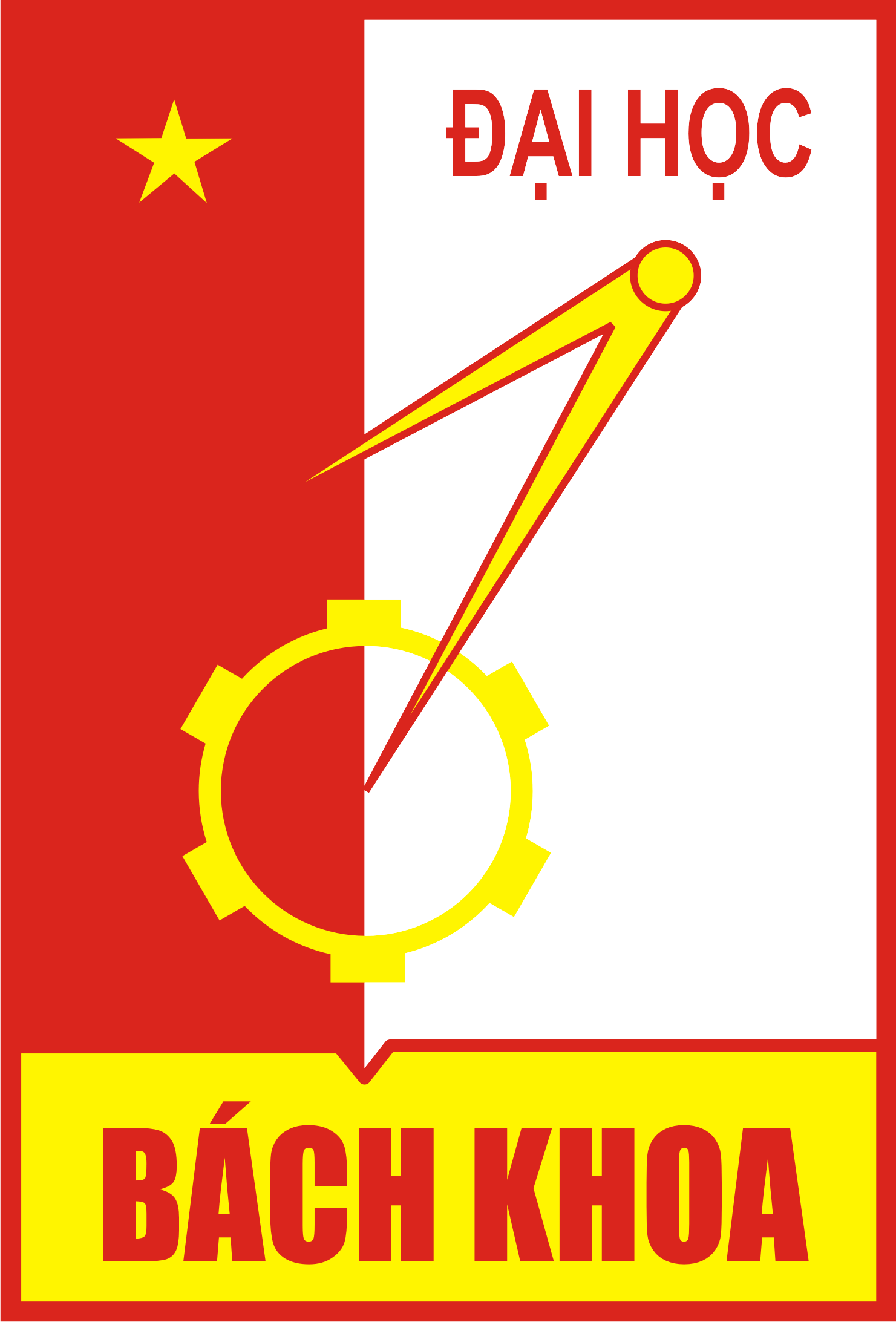
Đại học Bách Khoa Hà Nội – Trường CNTT&TT



**BÁO CÁO MINI-PROJECT LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

**HEDSPI – K67**

|  |  |
| --- | --- |
| Mã học phần: | **IT3103** |
| Mã lớp học: | **151965** |
| Giảng viên: | **Nguyễn Thị Thu Trang** |
| Nhóm số: | **2** |

**Mục lục**

[I. Thành viên và phân công nhiệm vụ 2](#_Toc155957955)

[II. Mô tả đề tài 2](#_Toc155957956)

[1. Yêu cầu ứng dụng 2](#_Toc155957957)

[2. Biểu đồ Use case 3](#_Toc155957958)

[3. Biểu đồ Class Diagram 3](#_Toc155957958)

[III. Thiết kế ứng dụng 4](#_Toc155957959)

[1. Khái quát chung 4](#_Toc155957960)

[2. Nội dung chi tiết cho từng package 5](#_Toc155957961)

[ Model 5](#_Toc155957962)

[ View 6](#_Toc155957963)

[ Controller 7](#_Toc155957964)

[3. Giải thích thiết kế 7](#_Toc155957965)

[ Các tính chất OOP được áp dụng 11](#_Toc155957968)

[ Một số phương thức quan trọng của các lớp 11](#_Toc155957969)

# Thành viên và phân công nhiệm vụ

* + - * Các thành viên trong nhóm đảm nhận một số công việc khác nhau của project.
      * Quá trình thiết kế, phát triển đề tài được cả nhóm bàn bạc và thực hiện qua group chat và các buổi họp trên Teams.
      * Bảng phân công

|  |  |  |
| --- | --- | --- |
| Thành viên | MSSV | Nhiệm vụ đảm nhận |
| Lê Tuấn Anh | 20205051 | - Thiết kế kiến trúc ứng dụng  - Code phần backend (Model) của ứng dụng  - Kiểm thử, sửa lỗi  - Làm bản cáo |
| Lê Việt Anh | 20225689 | - Thiết kế và cập nhật sơ đồ usecase, class diagram  - Thiết kế giao diện Emulator, Menu, Help  - Tạo và xử lý môi trường git |
| Lê Hải Anh | 20225594 | - Code phần giao diện Emulator, Menu  - Code phần Controller (MenuController)  - Kiểm thử, sửa lỗi |
| Hà Hoàng Kỳ Anh | 20210065 | - Code phần giao diện Help  - Code phần Controller (HelpController)  - Làm báo cáo |
| Lê Năng Quang Anh | 20226101 | - Code phần giao diện Menu  - Code phần Controller (MenuController) |

# Mô tả đề tài

## Yêu cầu ứng dụng

Dự án này mô phỏng một hệ sinh thái bao gồm các nhà sản xuất, 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 cách động lực dân số ảnh hưởng đến sự ổn định của hệ sinh thái.

* + - * GUI: bạn được tự do thiết kế giao diện người dùng tuy nhiên project này mục đích là thiết kế một chương trình sử dụng OOP nên không cần quá tập trung vào phần giao diện.
      * Thiết kế: Ứng dụng cần có các tính năng sau:
* Trên màn hình chính:
* Bắt đầu: Các tùy chọn cho các kịch bản mô phỏng khác nhau (Hệ sinh thái cân bằng, Quá tải dân số, Tuyệt chủng).
* Trợ giúp : giải thích các quy tắc mô phỏng và các khái niệm sinh học.
* Thoát: Nút thoát với xác nhận của người dùng.
* Trong màn hình mô phỏng:
* Đại diện trực quan:
  + **Nhà sản xuất**: Biểu tượng màu xanh lá cây nhỏ, sinh sản trong các ô trống.
  + **Động vật ăn cỏ**: Biểu tượng động vật cỡ trung bình, di chuyển để tiêu thụ nhà sản xuất.
  + **Động vật ăn thịt**: Biểu tượng động vật ăn thịt lớn, di chuyển để săn động vật ăn cỏ.
* Chuyển động :
  + Động vật ăn cỏ và động vật ăn thịt di chuyển ngẫu nhiên hoặc dựa trên sự gần gũi với thức ăn.
  + Tốc độ di chuyển có thể điều chỉnh qua cài đặt.
* Sinh sản :
  + Nhà sản xuất tự động sinh sản theo khoảng thời gian cố định.
  + Động vật ăn cỏ và động vật ăn thịt sinh sản dựa trên mức năng lượng.
* Tương tác :
  + Nhà sản xuất tạo ra năng lượng thông qua quang hợp.
  + Động vật ăn cỏ tiêu thụ nhà sản xuất để lấy năng lượng.
  + Động vật ăn thịt săn động vật ăn cỏ để lấy năng lượng.
* Chuyển giao năng lượng :
  + Nhà sản xuất chuyển đổi ánh sáng mặt trời thành năng lượng.
  + Động vật ăn cỏ nhận 10% năng lượng của nhà sản xuất.
  + Động vật ăn thịt nhận 10% năng lượng của động vật ăn cỏ.
* Hình ảnh hóa quần thể và năng lượng : Hiển thị thời gian thực kích thước quần thể và chuyển giao năng lượng.
* Điều khiển để điều chỉnh các thông số quần thể (tỷ lệ sinh, tỷ lệ tử vong, hiệu quả năng lượng).

- Trong menu trợ giúp :

* Giải thích động lực chuỗi thức ăn, chuyển giao năng lượng và tương tác quần thể.

- Trong tủy chọn thoát :

* Thoát sau khi xác nhận từ người dùng.

## Biểu đồ Use case



## Biểu đồ Class Diagram

A screenshot of a computer program

Description automatically generated

# Thiết kế ứng dụng

# 1. Khái quát chung

Ứng dụng được phát triển sử dụng kiến trúc MVC (Model – View – Controller) bao gồm 3 package:

* + Model: chứa các lớp lưu trữ dữ liệu và logic. Nó bao gồm các lớp và phương thức thực hiện các tác vụ như tính toán logic, thêm hoặc sử dữ liệu,…
  + View: chịu trách nhiệm hiển thị thông tin cho người dùng. Các dữ liệu được cung cấp bởi Model sẽ được thể hiện lên người dùng bởi thành phần View.
  + Controller: đóng vai trò trung tâm, nắm bắt và điều phối giữa Model và View. Khi người dùng tương tác với giao diện, các yêu cầu sẽ được gửi đến Controller. Controller sau đó xử lý sơ bộ dữ liệu này và chuyển tiếp cho Model. Model, chứa logic nghiệp vụ, sẽ tiếp tục xử lý và trả dữ liệu đã qua xử lý trở lại cho Controller. Cuối cùng, Controller sẽ sử dụng View để hiển thị dữ liệu đã qua xử lý cho người dùng, hoàn thành quá trình tương tác.
  + Ví dụ trong ứng dụng: khi người dùng thực hiện di chuyển một quân cờ trên giao diện (View), yêu cầu đó sẽ được Controller nắm bắt (thông qua các event handler). Controller nhận di chuyển đó, chuyển nó thành dữ liệu nước đi ứng với thao tác của người dùng và gửi cho Model để xử lý nước đi với các logic. Sau khi Model xử lí xong nước đi, nó sẽ gửi lại một kết quả nước đi đến cho Controller, Controlelr nhận kết quả đó và hiển thị lên giao diện.

A diagram of a computer system

Description automatically generated

→ Sử dụng kiến trúc MVC mang lại lợi ích lớn trong việc phân tách trách nhiệm giữa các thành phần. Trong MVC, Model, View và Controller đều có trách nhiệm riêng biệt từ đó giúp mã nguồn được tổ chức tốt, dễ dàng bảo trì và mở rộng.

**2. Nội dung chi tiết cho từng package**

* **Model :** 
  + **HeSinhThai.java :** Nơi định nghĩa các thuộc tính và phương thức cho hệ sinh thái
  + **O.java :**  Nơi định nghĩa các thuộc tính và phương thức cho một ô (1 đơn vị trên thuộc tính luoi của hệ sinh thai)
  + **SinhVat.java (trừu tượng) :** Nơi định nghĩa các thuộc tính căn bản và các phương thức trừu tượng cơ bản cho các sinh vật
  + **SinhVatSanXuat.java :** là lớp con kế thừa từ lớp SinhVat, định nghĩa cụ thể và bổ sung thêm các thuộc tính và phương thức để làm rõ cho Sinh vật sản xuất.
  + **SinhVatAnCo.java :** là lớp con kế thừa từ lớp SinhVat, định nghĩa cụ thể và bổ sung thêm các thuộc tính và phương thức để làm rõ cho Sinh vật ăn cỏ.
  + **SinhVatAnThit.java :** là lớp con kế thừa từ lớp SinhVat, định nghĩa cụ thể và bổ sung thêm các thuộc tính và phương thức để làm rõ cho Sinh vật ăn thịt.
* **View :** 
  + **Menu-view.fxml** : Giao diện UI cho màn hình trang chủ chính bao gồm (background, button, tiêu đề , .v.v)
  + **Emulator-view.fxml :** Giao diện UI cho màn hình mô phỏng hệ sinh thái bao gồm (lưới, image, text,..)
  + **Help.fxml :** Giao diện UI cho màn hình hướng dẫn bao gồm (image, button,..) được trình chiếu dưới dạng slide
* **Controller :** 
  + **MenuController.java** : Là nơi giao tiếp xử lý các handle được gửi từ màn hình UI Menu-view.fxml và liên kết với nội dung logic trong model
  + **EmulatorController.java :** Là nơi giao tiếp xử lý các handle được gửi từ màn hình UI EmulatorController.java và liên kết với nội dung logic trong model sau đó thể hiện lại trên màn hình UI
  + **HelpController.java :** Là nơi giao tiếp xử lý các handle được gửi từ màn hình UI Menu-view.fxml.

**3. Giải thích thiết kế**

### Các tính chất OOP được áp dụng

* **Tính đóng gói:**
  + Hầu hết các thuộc tính của các lớp đều được đặt thành private hoặc protected và có các getter và setter để truy cập vào chúng từ bên ngoài lớp. Không phải thuộc tính nào cũng được tạo cả getter và setter, chỉ những thuộc tính cần được truy cập mới được tạo.
* **Tính trừu tượng:**
  + Các lớp trừu tượng (abstract class) được sử dụng để thiết lập các phương thức trừu tượng (abstract method), các lớp con của lớp đó sẽ thực hiện triển khai các phương thức đó.
* **Tính kế thừa:**
  + Lớp **SinhVatSanXuat, SinhVatAnCo, SinhVatAnThit** kế thừa lớp **SinhVat**
* **Tính đa hình:**
  + Compile-time Polymorphism (static polymorphism): được thể hiện thông qua các phương thức overload. Nhiều phương thức có cùng tên nhưng có chữ ký khác nhau sẽ được thực hiện khác nhau dựa trên chữ ký của chúng.
  + Run-time Polymorphism (dynamic polymorphism): các lớp con mở rộng một lớp abstract sẽ override các phương thức abstract của lớp đó để thực hiện hoạt động cụ thể cho từng lớp con.

### Một số thuộc tính và phương thức quan trọng của các lớp

* **HeSinhThai.java**:
  + **Thuộc tính:**

private O[][] luoi;

private int chieuNgang;  
private int chieuRong;

private int deathrate;

private int birthrate;  
private int birthrate;

private int buocThoiGian;

private int slsvSanXuat;

private int slsvAnCo;

private int slsvAnThit;

private boolean isStopped;

public void initializeSinhVat(){  
}

“Lấp” các sinh vật 1 cách ngẫu nhiên vào lưới dựa vào slsvSanXuat, slsvAnCo, slsvAnThit

public void Capnhat(){  
}

Duyệt qua từng ô trong lưới kiểm tra xem sinhvat trong ô là gì ? từ đó sẽ thực hiện việc gọi các hành vi tương ứng của chúng ở trong class tương ứng

* **O.java**:
  + **Thuộc tính:**

private int x;  
private int y;

private SinhVat sinhvat;

public ObjectProperty<SinhVat> sinhvatProperty(){

}

Theo dõi các thay đổi của thuộc tính và cho phép liên kết (bind) nó với các thành phần khác trong giao diện. Khi giá trị của thuộc tính thay đổi, các phần tử giao diện hoặc các đối tượng ràng buộc với thuộc tính này cũng được tự động cập nhật mà không cần gọi thủ công.

* **SinhVat.java**:
  + **Thuộc tính:**

protected int energy; // Năng lượng ban đầu của sinh vật  
protected int x;

protected int y;

protected static HeSinhThai heSinhThai;

* + **public abtract void sinhsan()**: Phương thức trừu tượng mô tả hành vi sinh sản của sinh vật.
  + **public void chet()**: Phương thức mô tả hành vi chết của sinh vật (khi năng lượng <= 0)
* **SinhVatSanXuat.java**
  + **public void sinhsan()**: Phương thức mô tả hành vi sinh sản của sinh vật sản xuất (khi trải qua 1 số thời gian nhất định thì sẽ có 1 tỉ lệ nhất định là sẽ sinh sản trên 1 ô trống ngẫu nhiên trên lưới hệ sinh thái).
  + **public void quanghop()**:Phương thức mô tả việc tăng năng lượng qua thời gian của sinh vật sản xuất (sẽ tăng năng lượng ngẫu nhiên trong 1 khoảng)
* **SinhVatAnCo.java**
  + **public void dichuyen()**: Phương thức mô tả hành vi di chuyển của sinh vật ăn cỏ (sinh vật ăn cỏ sẽ xác định mục tiêu gần nhất qua phương thức **timSinhVatSanXuatGanNhat()** sau đó sẽ quyết định lựa chọn 1 trong các xung quanh nó để đi)
  + **public SinhVatSanXuat timSinhVatSanXuatGanNhat()** : phương thức trả về 1 sinhvatsanxuat gần nhất
  + **public void sinhsan()**: Phương thức mô tả hành vi sinh sản của sinh vật ăn cỏ (khi năng lượng vượt quá ngưỡng ban đầu thì sinh vật sẽ sinh ra 1 sinh vật ở xung quanh nó).
* **SinhVatAnThit.java**
  + **public void dichuyen()**: Phương thức mô tả hành vi di chuyển của sinh vật ăn thịt (sinh vật ăn thịt sẽ xác định mục tiêu gần nhất qua phương thức **timSinhVatAnCoGanNhat()** sau đó sẽ quyết định lựa chọn 1 trong các xung quanh nó để đi)
  + **public SinhVatAnCo timSinhVatAnCoGanNhat()** : phương thức trả về 1 SinhVatAnCo gần nhất
  + **public void sinhsan()**: Phương thức mô tả hành vi sinh sản của sinh vật ăn thịt (khi năng lượng vượt quá ngưỡng ban đầu thì sinh vật sẽ sinh ra 1 sinh vật ở xung quanh nó).