

**TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP**

**KHOA ĐIỆN TỬ**

**Bộ môn: Công nghệ thông tin.**



# **BÀI TẬP KẾT THÚC MÔN HỌC**

**MÔN HỌC**

**LẬP TRÌNH PYTHON**

**SINH VIÊN THỰC HIỆN : ĐẶNG ĐÌNH ĐẠT**

**MSV : K225480106003**

**LỚP : K58KTP**

**GIÁO VIÊN GIẢNG DẠY : NGUYỄN VĂN HUY**

**Link YouTube:** <https://youtu.be/bUkOIOwtrck>



**Link GitHub:** [https://github.com/dat-022004/BTL\\_PyThon.git](https://github.com/dat-022004/BTL_PyThon.git)



**THÁI NGUYÊN - 2025**

## BÀI TẬP KẾT THÚC MÔN

### MÔN HỌC: LẬP TRÌNH PYTHON

### BỘ MÔN: CÔNG NGHỆ THÔNG TIN

**Sinh viên:** Đặng Đình Đạt

Msv: K225480106003

**Lớp:** K58KTP

Ngành : Kỹ thuật phần mềm

**Giáo viên giảng dạy:** Nguyễn Văn Huy

Ngày giao đề tài : 01/06/2025

Ngày hoàn thành: 10/06/2025

Tên đề tài: Xây ứng dụng Critter Caretaker (Chapter 8) cho phép tạo, cho ăn, cho ngủ critter qua giao diện.

#### **Đầu vào – đầu ra:**

- Đầu vào: Tên critter từ Entry, các nút hành động.
- Đầu ra: Trạng thái critter (hunger, boredom) hiển thị trên Label.

#### **Tính năng yêu cầu:**

- Class Critter với attributes và methods.
- GUI: Entry tạo critter, Buttons: “Tạo”, “Cho ăn”, “Chơi”, “Ngủ”.
- Cập nhật trạng thái real-time.
- Bắt lỗi khi chưa tạo critter.

#### **Kiểm tra & kết quả mẫu:**

- Tạo “Bob” → Hunger=0, Boredom=0.
- Nhấn “Cho ăn” → Hunger giảm, Label cập nhật.

**GIÁO VIÊN HƯỚNG DẪN**

*(Ký và ghi rõ họ tên)*

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

Xếp loại: ..... Điểm : .....

Thái Nguyên, ngày....tháng.....năm 20....

**GIÁO VIÊN HƯỚNG DẪN**

*(Ký và ghi rõ họ tên)*

# MỤC LỤC

LỜI CAM ĐOAN .....	6
DANH MỤC HÌNH ẢNH .....	7
LỜI NÓI ĐẦU .....	7
CHƯƠNG 1: GIỚI THIỆU ĐẦU BÀI .....	10
1.1. Đặt vấn đề .....	10
1.2. Mục tiêu của đề tài .....	10
1.3. Phạm vi thực hiện .....	11
1.4. Các tính năng chính của phần mềm .....	12
1.5. Thách thức và khó khăn .....	13
1.6. Kiến thức và công cụ sử dụng .....	13
Tóm tắt Chương 1 .....	14
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT .....	15
2.1. Ngôn ngữ lập trình Python .....	15
2.2. Thư viện Tkinter – Công cụ xây dựng giao diện đồ họa .....	16
2.3. Cập nhật trạng thái theo thời gian với phương thức after() .....	17
2.4. Xử lý trạng thái, cảnh báo và phản hồi người dùng .....	17
2.5. Các kỹ thuật hỗ trợ .....	18
Tóm tắt Chương 2 .....	19
CHƯƠNG 3: THIẾT KẾ VÀ XÂY DỰNG CHƯƠNG TRÌNH .....	20
3.1. Phân tích tổng thể và sơ đồ hệ thống .....	20
3.1.1. Phân tích chức năng hệ thống .....	20
3.2. Sơ đồ khối thuật toán chính .....	21
3.2.1. Tạo thú .....	21

3.2.2. Cho ăn.....	22
3.2.3. Chơi .....	23
3.2.4. Ngủ .....	24
3.2.5. Tự động cập nhật .....	25
3.3. Cấu trúc dữ liệu.....	26
3.4. Các hàm trong chương trình chính .....	29
3.5. Thiết kế giao diện .....	33
Tóm tắt Chương 3 .....	33
<b>CHƯƠNG 4: THỰC NGHIỆM VÀ KẾT LUẬN .....</b>	<b>34</b>
4.1. Thực nghiệm .....	34
4.1.1. Tạo Critter.....	34
4.1.2. Cho ăn.....	35
4.1.3. Chơi .....	38
4.1.4. Ngủ .....	40
4.1.5. Kiểm tra trạng thái chết .....	41
4.1.6. Ghi nhận thực nghiệm .....	42
4.2. Kết luận.....	43
4.2.1. Kết quả đạt được.....	43
4.2.2. Bài học kinh nghiệm.....	43
4.2.3. Hướng phát triển tương lai .....	44
4.3. Lời cảm ơn. ....	45
Tóm tắt Chương 4 .....	45
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>46</b>

## **LỜI CAM ĐOAN**

Em xin cam đoan rằng đề tài “Xây dựng ứng dụng Critter Caretaker” là công trình do em, Đặng Đình Đạt, tự thực hiện dưới sự hướng dẫn của thầy Nguyễn Văn Huy. Toàn bộ nội dung, mã nguồn, và kết quả trình bày trong tài liệu đều được em nghiên cứu, phát triển và kiểm thử dựa trên kiến thức từ môn học Lập trình Python và các tài liệu tham khảo đã được liệt kê. Em chịu hoàn toàn trách nhiệm về tính trung thực và chính xác của các nội dung trong tài liệu. Nếu có bất kỳ sai sót hoặc vi phạm nào, em xin chịu mọi hình thức xử lý theo quy định của nhà trường.

Thái Nguyên, ngày 1 tháng 6 năm 2025

**Sinh viên**

**Đặng Đình Đạt**

## DANH MỤC HÌNH ẢNH

Hình 3.1: Sơ đồ phân cấp chức năng của phần mềm Critter Caretaker

Hình 3.2 : Sơ đồ thuật toán khởi tạo trạng thái Critter

Hình 3.3 : Sơ đồ thuật toán cho Critter ăn

Hình 3.4 : Sơ đồ thuật toán chơi với Critter

Hình 3.5 : Sơ đồ thuật toán cho Critter ngủ

Hình 3.6: Sơ đồ thuật toán tự động cập nhật hunger

Hình 3.7 : Sơ đồ thuật toán tự động cập nhật boredom

Hình 4.1: Thao tác tạo Critter

Hình 4.2: Trường hợp Critter đói ( $\text{hunger} \geq 5$ )

Hình 4.3: Trường hợp Critter không đói ( $\text{hunger} = 0$ )

Hình 4.4: Trường hợp Critter đang ngủ hoặc đã chết

Hình 4.5: Trường hợp chưa tạo Critter

Hình 4.6: Trường hợp Critter buồn chán ( $\text{boredom} \geq 5$ )

Hình 4.7: Trường hợp Critter không buồn chán ( $\text{boredom} = 0$ )

Hình 4.8: Trường hợp Critter đang ngủ hoặc đã chết

Hình 4.9: Trường hợp chưa tạo Critter

Hình 4.10: Cho Critter ngủ

Hình 4.11 Critter đã chết

## LỜI NÓI ĐẦU

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, lập trình máy tính đã trở thành một kỹ năng thiết yếu, không chỉ trong lĩnh vực kỹ thuật mà còn trong nhiều khía cạnh của đời sống như giáo dục, giải trí, và quản lý. Ngôn ngữ lập trình Python, với cú pháp đơn giản, dễ học và tính linh hoạt, đã trở thành công cụ phổ biến để phát triển các ứng dụng từ đơn giản đến phức tạp. Trong khuôn khổ môn học Lập trình Python tại Trường Đại học Kỹ thuật Công nghiệp, đề tài “Xây dựng ứng dụng Critter Caretaker” được thực hiện nhằm áp dụng các kiến thức lý thuyết vào thực tiễn, tạo ra một phần mềm mô phỏng nuôi thú ảo với giao diện đồ họa thân thiện, sử dụng Python và thư viện Tkinter.

Đề tài Critter Caretaker không chỉ mang tính giải trí mà còn có giá trị giáo dục, giúp người học củng cố các kỹ năng lập trình cơ bản như quản lý dữ liệu, xử lý thời gian thực, thiết kế giao diện người dùng, và xử lý sự kiện. Ứng dụng cho phép người dùng tạo một thú ảo (Critter), chăm sóc nó thông qua các hành động như cho ăn, chơi, và cho ngủ, đồng thời theo dõi trạng thái thời gian thực. Thông qua việc thực hiện đề tài, em đã có cơ hội vận dụng các kiến thức về Python, từ cấu trúc điều khiển, hàm, đến các kỹ thuật xây dựng giao diện đồ họa, đồng thời rèn luyện tư duy logic và khả năng giải quyết vấn đề.

Mục tiêu của tài liệu này là trình bày chi tiết quá trình xây dựng phần mềm Critter Caretaker, từ lý do lựa chọn đề tài, cơ sở lý thuyết, thiết kế thuật toán, đến kết quả thực nghiệm và hướng phát triển tương lai. Tài liệu không chỉ nhằm đáp ứng yêu cầu của bài tập kết thúc môn mà còn là cơ hội để em tổng hợp, đánh giá và chia sẻ những kiến thức, kinh nghiệm thu được trong quá trình học tập.

Trong quá trình thực hiện, em đã nhận được sự hướng dẫn tận tình của thầy Nguyễn Văn Huy, cùng với sự hỗ trợ từ các tài liệu tham khảo và bạn bè đồng môn. Em xin chân thành cảm ơn thầy Nguyễn Văn Huy đã cung cấp kiến thức nền tảng và định hướng rõ ràng, giúp em hoàn thành đề tài đúng thời hạn. Đồng thời, em xin cảm ơn nhà trường và Khoa Điện tử đã tạo điều kiện để em tiếp cận với môn học Lập trình Python, mở ra cơ hội khám phá tiềm năng của lập trình trong việc phát triển các ứng dụng thực tiễn.



Mặc dù đã nỗ lực hết sức, tài liệu và chương trình vẫn có thể tồn tại những thiếu sót do hạn chế về thời gian và kinh nghiệm. Em rất mong nhận được những ý kiến đóng góp từ thầy cô và các bạn để hoàn thiện đề tài hơn nữa.

# CHƯƠNG 1: GIỚI THIỆU ĐẦU BÀI

## 1.1. Đặt vấn đề

Trong bối cảnh công nghệ thông tin phát triển mạnh mẽ, phần mềm không chỉ phục vụ các lĩnh vực như sản xuất, kinh doanh hay khoa học kỹ thuật, mà còn đóng vai trò quan trọng trong giải trí và giáo dục. Các ứng dụng mô phỏng, đặc biệt là các chương trình nuôi thú ảo, đã trở thành một xu hướng phổ biến, mang lại giá trị giải trí đồng thời giúp người dùng rèn luyện các kỹ năng như quản lý thời gian, trách nhiệm, và khả năng phản ứng với các tình huống thay đổi.

Phần mềm nuôi thú ảo là một ví dụ điển hình của ứng dụng mô phỏng, nơi người dùng tương tác với một thực thể ảo thông qua các hành động như chăm sóc, nuôi dưỡng, và theo dõi trạng thái. Những chương trình này không chỉ mang tính giải trí mà còn giúp người dùng hiểu được tầm quan trọng của việc duy trì sự chăm sóc liên tục và xử lý các tình huống phát sinh. Từ nhu cầu này, đề tài xây dựng phần mềm "Critter Caretaker" ra đời, nhằm tạo ra một ứng dụng đơn giản nhưng đầy đủ tính năng, sử dụng ngôn ngữ lập trình Python và thư viện giao diện đồ họa Tkinter.

Critter Caretaker là một chương trình mô phỏng nuôi thú ảo, cho phép người dùng tạo một thú ảo (Critter), chăm sóc nó thông qua các hành động như cho ăn, chơi, và cho ngủ, đồng thời theo dõi trạng thái của thú theo thời gian thực. Đề tài này được chọn vì tính thực tiễn, khả năng ứng dụng kiến thức lập trình Python, và giá trị giáo dục trong việc giúp người dùng hiểu về logic chương trình, tương tác người dùng, và xử lý thời gian thực. Mặc dù có quy mô nhỏ, chương trình vẫn mang lại trải nghiệm tương tác thú vị và là nền tảng để phát triển các ứng dụng phức tạp hơn trong tương lai.

## 1.2. Mục tiêu của đề tài

Mục tiêu chính của đề tài là xây dựng một phần mềm nuôi thú ảo với giao diện đồ họa thân thiện, sử dụng Python và Tkinter, đáp ứng các yêu cầu sau:

- a. **Tạo thú ảo:** Cho phép người dùng nhập tên để tạo một Critter mới với các trạng thái ban đầu được thiết lập (mức độ đói và buồn chán bằng 0, trạng thái sống).

**Tương tác với Critter:** Cung cấp các hành động như:

- **Cho ăn:** Giảm mức độ đói của Critter.

- **Chơi:** Giảm mức độ buồn chán của Critter.
  - **Ngủ:** Tạm dừng các hoạt động của Critter trong một khoảng thời gian để mô phỏng trạng thái nghỉ ngơi.
- Hiển thị trạng thái:** Cập nhật và hiển thị thông tin về tên, mức độ đói, mức độ buồn chán, và trạng thái sống/chết của Critter trên giao diện.
  - Mô phỏng thời gian thực:** Tự động tăng mức độ đói và buồn chán theo thời gian để mô phỏng hành vi tự nhiên, yêu cầu người dùng chăm sóc thường xuyên.
  - Cảnh báo và phản hồi:** Thông báo cho người dùng khi Critter ở trạng thái nguy hiểm (quá đói) hoặc chết, đồng thời cung cấp phản hồi cho các hành động của người dùng.
  - Xử lý lỗi:** Đảm bảo chương trình phản hồi hợp lý khi người dùng thực hiện các thao tác không hợp lệ, như cố gắng tương tác khi chưa tạo Critter.

Mục tiêu tổng quát là tạo ra một ứng dụng đơn giản nhưng đầy đủ tính năng, giúp người dùng trải nghiệm việc chăm sóc thú ảo một cách trực quan, đồng thời củng cố kiến thức lập trình Python thông qua thực hành.

### 1.3. Phạm vi thực hiện

Đề tài Critter Caretaker được giới hạn trong một số phạm vi cụ thể để đảm bảo tính khả thi trong khuôn khổ bài tập kết thúc môn học:

- **Quy mô Critter:** Chương trình chỉ hỗ trợ quản lý một Critter tại một thời điểm. Người dùng phải tạo mới Critter nếu muốn bắt đầu lại.
- **Lưu trữ dữ liệu:** Không sử dụng cơ sở dữ liệu hoặc lưu trữ trạng thái vào file. Dữ liệu về Critter (tên, mức độ đói, buồn chán) sẽ bị xóa khi chương trình tắt.
- **Giao diện đồ họa:** Giao diện được thiết kế đơn giản, sử dụng Tkinter với các thành phần cơ bản như ô nhập liệu (Entry), nút bấm (Button), nhãn (Label), và khung chứa (Frame). Giao diện tập trung vào tính năng cốt lõi và trải nghiệm người dùng trực quan.
- **Tính năng chính:** Chỉ tập trung vào các hành động cơ bản (tạo, cho ăn, chơi, ngủ) và hiển thị trạng thái, không bao gồm các tính năng nâng cao như lưu trữ nhiều Critter, thêm hành vi phức tạp, hoặc tích hợp âm thanh/hình ảnh động.

- **Thời gian thực:** Mô phỏng thời gian thực thông qua các bộ đếm thời gian, với mức độ đói tăng mỗi 10 giây và mức độ buồn chán tăng mỗi 20 giây.

Mặc dù giới hạn về quy mô, chương trình vẫn đáp ứng đầy đủ các yêu cầu về tương tác người dùng, xử lý logic thời gian thực, và cung cấp trải nghiệm mô phỏng thực tế.

#### 1.4. Các tính năng chính của phần mềm

Phần mềm Critter Caretaker được thiết kế với các tính năng lý thuyết sau:

- Tạo thú ảo:** Người dùng nhập tên vào ô nhập liệu để tạo một Critter mới. Critter được khởi tạo với trạng thái ban đầu (đói = 0, buồn chán = 0, còn sống).
- Hiển thị trạng thái:** Giao diện hiển thị liên tục thông tin về tên Critter, mức độ đói, mức độ buồn chán, và trạng thái sống/chết thông qua một vùng nhãn chuyên dụng.

##### Tự động cập nhật trạng thái:

- Mức độ đói tăng 1 đơn vị sau mỗi 10 giây nếu không được chăm sóc.
- Mức độ buồn chán tăng 1 đơn vị sau mỗi 20 giây nếu không được chơi.

##### Tương tác chăm sóc:

- **Cho ăn:** Giảm mức độ đói khi người dùng nhấn nút tương ứng.
- **Chơi:** Giảm mức độ buồn chán khi người dùng thực hiện hành động chơi.
- **Ngủ:** Tạm dừng các hành động trong 3 giây, mô phỏng trạng thái nghỉ ngơi, và khóa các nút tương tác trong thời gian này.

**Cảnh báo nguy hiểm:** Hiển thị thông báo khi Critter đạt các ngưỡng trạng thái nguy hiểm:

- Mức độ đói  $\geq 5$ : Cảnh báo nhẹ để nhắc nhở người dùng.
- Mức độ đói  $\geq 10$ : Cảnh báo nghiêm trọng về nguy cơ chết.
- Mức độ đói  $\geq 15$ : Critter chết, chương trình thông báo và khóa các hành động.

**Giao diện trực quan:** Sử dụng màu sắc, biểu tượng cảm xúc (emoji), và bố cục rõ ràng để tăng tính thân thiện và sinh động. Ví dụ, các thông báo sử dụng emoji như 🍎 (cho ăn), 🎲 (chơi), 😴 (ngủ), và 💀 (chết).

## 1.5. Thách thức và khó khăn

Trong quá trình phát triển Critter Caretaker, một số thách thức lý thuyết cần được xem xét:

- a. **Quản lý thời gian thực:** Việc mô phỏng hành vi tự nhiên của Critter (tăng đói và buồn chán theo thời gian) đòi hỏi cơ chế lập lịch chính xác, đảm bảo các tác vụ được thực thi đúng thời điểm mà không làm gián đoạn giao diện người dùng.
- b. **Đồng bộ giao diện và logic:** Trạng thái của Critter (đói, buồn chán, ngủ, chết) cần được cập nhật liên tục trên giao diện, yêu cầu sự phối hợp chặt chẽ giữa logic xử lý và hiển thị.
- c. **Trải nghiệm người dùng:** Giao diện cần được thiết kế đơn giản, trực quan, và dễ sử dụng, đặc biệt với người dùng không quen thuộc với lập trình hoặc ứng dụng GUI. Điều này bao gồm việc sử dụng màu sắc, thông báo rõ ràng, và xử lý lỗi hợp lý.
- d. **Xử lý lỗi người dùng:** Chương trình cần dự đoán và xử lý các trường hợp như người dùng nhập tên rỗng, cố tương tác khi Critter chưa được tạo, hoặc thực hiện hành động không hợp lệ khi Critter đang ngủ hoặc đã chết.
- e. **Tổ chức mã nguồn:** Mã cần được cấu trúc rõ ràng, chia thành các hàm hoặc khối chức năng riêng biệt để dễ hiểu, dễ bảo trì, và có thể mở rộng trong tương lai.

## 1.6. Kiến thức và công cụ sử dụng

Để xây dựng Critter Caretaker, các kiến thức và công cụ lý thuyết sau được áp dụng:

- a. **Ngôn ngữ lập trình Python:** Sử dụng các tính năng cốt lõi của Python như biến, kiểu dữ liệu (chuỗi, số nguyên, boolean), cấu trúc điều khiển (if, else), và hàm để xử lý logic chương trình. Python cung cấp cú pháp đơn giản, dễ đọc, phù hợp cho việc phát triển ứng dụng mô phỏng.
- b. **Thư viện Tkinter:** Công cụ chính để xây dựng giao diện đồ họa, cung cấp các thành phần như cửa sổ, nhãn, ô nhập liệu, nút bấm, và hệ thống lưới để bố trí giao diện. Tkinter hỗ trợ xử lý sự kiện và tùy chỉnh giao diện để tạo trải nghiệm thân thiện.

- c. **Xử lý thời gian thực:** Sử dụng phương thức `after()` của Tkinter để lập lịch các tác vụ định kỳ, như tăng chỉ số đói và buồn chán, tạo hiệu ứng mô phỏng thời gian thực.
- d. **Xử lý sự kiện:** Kết hợp Python và Tkinter để xử lý các hành động người dùng (như nhấn nút, thay đổi kích thước cửa sổ) thông qua các hàm callback và sự kiện `bind()`.
- e. **Tư duy logic và mô phỏng:** Áp dụng tư duy lập trình để mô phỏng hành vi của thú ảo dựa trên thời gian và tương tác người dùng, bao gồm việc quản lý trạng thái, đưa ra phản hồi, và xử lý các trường hợp lỗi.

### *Tóm tắt Chương 1*

*Chương 1 đã trình bày tổng quan lý thuyết về đề tài Critter Caretaker, nhấn mạnh vai trò của ứng dụng nuôi thú ảo trong giải trí và giáo dục. Mục tiêu là xây dựng một phần mềm đơn giản, sử dụng Python và Tkinter, cho phép người dùng tạo và chăm sóc Critter thông qua các hành động như cho ăn, chơi, ngủ, đồng thời hiển thị trạng thái thời gian thực. Phạm vi đề tài giới hạn ở một Critter, giao diện cơ bản, và không lưu trữ dữ liệu. Các tính năng chính bao gồm tạo Critter, tương tác, cập nhật trạng thái, và cảnh báo. Thách thức nằm ở quản lý thời gian thực, đồng bộ giao diện, và tổ chức mã nguồn. Các công cụ như Python, Tkinter, và phương thức `after()` là nền tảng lý thuyết để phát triển chương trình.*

## CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

Phần mềm Critter Caretaker được phát triển bằng ngôn ngữ lập trình Python, tận dụng các tính năng cốt lõi của Python và thư viện Tkinter để tạo giao diện đồ họa, quản lý thời gian thực thông qua phương thức `after()`, và xử lý tương tác người dùng. Dưới đây là các khái niệm lý thuyết chính liên quan đến lập trình Python được sử dụng trong chương trình.

### 2.1. Ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình cấp cao, dễ đọc, và linh hoạt, được sử dụng rộng rãi trong nhiều lĩnh vực như phát triển ứng dụng, phân tích dữ liệu, và mô phỏng. Python hỗ trợ viết mã rõ ràng, ngắn gọn nhờ cú pháp đơn giản và các thư viện tích hợp mạnh mẽ. Trong Critter Caretaker, các tính năng chính của Python được sử dụng bao gồm:

- a. **Biến và kiểu dữ liệu:** Python sử dụng các kiểu dữ liệu như chuỗi (str) để lưu tên Critter, số nguyên (int) để lưu chỉ số đói (hunger) và buồn chán (boredom), và giá trị logic (bool) để theo dõi trạng thái sống/chết (is\_dead). Các biến này được quản lý để lưu trữ và cập nhật trạng thái của Critter.
- b. **Cấu trúc điều khiển:** Các câu lệnh điều kiện (if, else) được dùng để kiểm tra trạng thái, ví dụ: kiểm tra xem Critter đã được tạo chưa hoặc có đang ngủ không trước khi thực hiện hành động. Vòng lặp (while, for) được sử dụng gián tiếp trong các cơ chế thời gian để lặp lại các tác vụ định kỳ.
- c. **Hàm (Functions):** Python hỗ trợ định nghĩa các hàm để thực hiện các tác vụ cụ thể, như xử lý hành động cho ăn, chơi, hoặc ngủ. Các hàm này giúp tổ chức mã nguồn theo từng chức năng, dễ bảo trì và tái sử dụng.
- d. **Quản lý sự kiện:** Python kết hợp với Tkinter để xử lý các sự kiện người dùng (như nhấn nút) thông qua các hàm callback, đảm bảo chương trình phản hồi nhanh chóng với hành động của người dùng.

Python cung cấp sự linh hoạt trong việc kết hợp logic xử lý và giao diện, giúp Critter Caretaker dễ dàng mô phỏng hành vi của thú ảo một cách tự nhiên.

## 2.2. Thư viện Tkinter – Công cụ xây dựng giao diện đồ họa

Tkinter là thư viện giao diện đồ họa (GUI) tiêu chuẩn của Python, được tích hợp sẵn và cho phép tạo các ứng dụng với giao diện trực quan, thân thiện. Tkinter cung cấp các thành phần giao diện (widget) như nhãn, ô nhập liệu, nút bấm, và khung chứa, cùng với cơ chế xử lý sự kiện để tương tác với người dùng. Các thành phần chính của Tkinter được sử dụng trong Critter Caretaker bao gồm:

- a. **Tk:** Hàm khởi tạo cửa sổ chính của ứng dụng, đóng vai trò là khung chứa cho các widget khác. Cửa sổ này có thể được tùy chỉnh tiêu đề (ví dụ: "Critter Caretaker"), kích thước, và màu nền để tăng tính thẩm mỹ.
- b. **Label:** Widget hiển thị văn bản hoặc hình ảnh tĩnh, dùng để:
  - Hiển thị hướng dẫn như "Nhập tên Critter".
  - Hiển thị trạng thái hiện tại của Critter (tên, mức độ đói, buồn chán).
  - Hiển thị thông báo phản hồi như "Đã cho Critter ăn!".
- c. **Entry:** Widget cho phép người dùng nhập văn bản, được sử dụng để nhập tên Critter. Entry hỗ trợ tùy chỉnh font chữ, màu nền, viền, và kích thước để phù hợp với giao diện.
- d. **Button:** Widget tạo các nút bấm tương tác, như "Tạo Critter", "Cho ăn", "Chơi", và "Ngủ". Button có thể được gán hàm callback để xử lý hành động khi nhấn, đồng thời hỗ trợ tùy chỉnh màu sắc, kích thước font, và trạng thái bật/tắt.
- e. **Frame và LabelFrame:** Widget chứa dùng để nhóm các thành phần giao diện, tạo bố cục rõ ràng. LabelFrame có thêm tiêu đề (như "Trạng thái Critter") để tăng tính trực quan và dễ hiểu.
- f. **Hệ thống lưới (grid):** Một phương pháp bố trí trong Tkinter, sắp xếp các widget theo hàng và cột, tương tự như bảng. Grid đảm bảo các thành phần giao diện được căn chỉnh linh hoạt, thích ứng với kích thước cửa sổ thay đổi.

Tkinter hỗ trợ tùy chỉnh giao diện bằng cách sử dụng màu sắc, font chữ, và biểu tượng cảm xúc (emoji) để làm sinh động trải nghiệm người dùng. Ngoài ra, Tkinter cung cấp cơ chế xử lý sự kiện thông qua phương thức bind(), cho phép chương trình phản ứng với các hành động như click chuột hoặc thay đổi kích thước cửa sổ.



### 2.3. Cập nhật trạng thái theo thời gian với phương thức after()

Để mô phỏng hành vi thời gian thực của Critter, chương trình sử dụng phương thức after() của Tkinter. Đây là một cơ chế lập lịch (scheduling) cho phép thực thi một hàm sau một khoảng thời gian xác định (tính bằng mili giây). after() giúp tạo các tác vụ nền (background tasks) hoặc bộ đếm thời gian (timers) mà không làm gián đoạn giao diện người dùng.

Nguyên lý hoạt động của after():

- a. **Cú pháp:** root.after(time\_ms, callback\_function), trong đó time\_ms là thời gian chờ và callback\_function là hàm được gọi sau khoảng thời gian đó.
- b. **Ứng dụng trong Critter Caretaker:**
  - Tăng chỉ số đói (hunger) mỗi 10 giây, mô phỏng nhu cầu ăn uống tự nhiên.
  - Tăng chỉ số buồn chán (boredom) mỗi 20 giây, mô phỏng trạng thái tâm lý của Critter.
- c. **Tính tuần hoàn:** Để tạo hiệu ứng liên tục, hàm callback có thể gọi lại after() để lặp lại tác vụ, tạo thành một chu kỳ thời gian (ví dụ: cứ mỗi 10 giây tăng chỉ số đói).

Cơ chế này đảm bảo Critter thay đổi trạng thái tự động theo thời gian, tạo cảm giác thực tế và yêu cầu người dùng tương tác thường xuyên để duy trì trạng thái tốt cho Critter.

### 2.4. Xử lý trạng thái, cảnh báo và phản hồi người dùng

Một ứng dụng tương tác như Critter Caretaker cần cung cấp phản hồi rõ ràng và kịp thời để người dùng nắm bắt trạng thái hiện tại và kết quả của hành động. Các khái niệm lý thuyết liên quan bao gồm:

- a. **Quản lý trạng thái:** Python sử dụng các biến để lưu trữ trạng thái của Critter, như mức độ đói, buồn chán, trạng thái sống/chết, hoặc trạng thái ngủ. Các biến này được cập nhật liên tục và đồng bộ với giao diện để phản ánh chính xác trạng thái hiện tại.

- b. **Phản hồi hành động:** Mỗi hành động của người dùng (như nhấn nút "Cho ăn") được phản hồi bằng một thông báo văn bản, thường kèm biểu tượng cảm xúc để tăng tính trực quan. Ví dụ:
- Hành động thành công: "🍎 Đã cho Critter ăn!"
  - Hành động không hợp lệ: "Critter không đói!" hoặc "Critter đang ngủ!"
- c. **Cảnh báo theo ngưỡng:** Chương trình sử dụng các câu lệnh điều kiện để kiểm tra ngưỡng trạng thái và hiển thị cảnh báo:
- Khi  $\text{hunger} \geq 5$ : Thông báo nhẹ để nhắc nhở người dùng chăm sóc.
  - Khi  $\text{hunger} \geq 10$ : Cảnh báo nghiêm trọng về nguy cơ Critter chết.
  - Khi  $\text{hunger} \geq 15$ : Thông báo Critter chết và khóa các chức năng tương tác.
- d. **Giao diện thông báo:** Một vùng hiển thị (thường là Label hoặc Frame) được sử dụng để cập nhật thông báo liên tục, giúp người dùng dễ dàng theo dõi trạng thái và hành động cần thực hiện.

## 2.5. Các kỹ thuật hỗ trợ

Ngoài các khái niệm chính, một số kỹ thuật Python và Tkinter hỗ trợ được sử dụng để nâng cao trải nghiệm người dùng và đảm bảo tính ổn định:

- a. **Tự động điều chỉnh giao diện:** Tkinter hỗ trợ xử lý sự kiện thay đổi kích thước cửa sổ thông qua phương thức `bind()` với sự kiện `<Configure>`. Điều này cho phép chương trình tự động điều chỉnh kích thước font chữ, khoảng cách, và bố cục để đảm bảo giao diện luôn rõ ràng trên các kích thước màn hình khác nhau.
- b. **Kiểm soát trạng thái widget:** Tkinter cho phép thay đổi trạng thái của widget (như Button) thông qua thuộc tính `state` (`normal` hoặc `disabled`). Điều này được dùng để ngăn người dùng thực hiện hành động không hợp lệ, ví dụ: không cho nhấn nút khi Critter đang ngủ hoặc đã chết.
- c. **Tổ chức mã nguồn:** Python khuyến khích tổ chức mã theo các hàm riêng biệt, mỗi hàm đảm nhiệm một chức năng cụ thể (như xử lý cho ăn, chơi, hoặc cập nhật trạng thái). Điều này giúp mã nguồn dễ đọc, dễ bảo trì, và dễ mở rộng trong tương lai.
- d. **Sử dụng biểu tượng cảm xúc và màu sắc:** Biểu tượng cảm xúc (emoji) và màu sắc được tích hợp vào giao diện để tăng tính sinh động. Ví dụ, các nút chức năng

sử dụng màu sắc khác nhau để dễ phân biệt, và thông báo sử dụng emoji để làm rõ nội dung.

- e. **Xử lý lỗi người dùng:** Tkinter cung cấp module messagebox để hiển thị các hộp thoại cảnh báo hoặc lỗi, như khi người dùng cố tạo Critter mà không nhập tên. Điều này đảm bảo chương trình không bị crash và cung cấp phản hồi rõ ràng.

## *Tóm tắt Chương 2*

*Phần mềm Critter Caretaker được xây dựng dựa trên các tính năng cốt lõi của Python, bao gồm quản lý biến, cấu trúc điều khiển, và hàm để xử lý logic chương trình. Thư viện Tkinter cung cấp các công cụ như Tk, Label, Entry, Button, và grid để tạo giao diện đồ họa trực quan. Phương thức after() cho phép mô phỏng thời gian thực, trong khi các kỹ thuật xử lý trạng thái, phản hồi, và cảnh báo đảm bảo tính tương tác. Các kỹ thuật hỗ trợ như điều chỉnh giao diện, kiểm soát widget, và xử lý lỗi giúp chương trình ổn định và thân thiện với người dùng. Những kiến thức này tạo nền tảng cho việc phát triển các ứng dụng Python tương tác trong tương lai.*

## CHƯƠNG 3: THIẾT KẾ VÀ XÂY DỰNG CHƯƠNG TRÌNH

Sau khi đã xác định rõ mục tiêu và các kiến thức lý thuyết cần áp dụng, bước tiếp theo trong quá trình xây dựng phần mềm Critter Caretaker là thiết kế tổng thể hệ thống, lựa chọn cấu trúc dữ liệu phù hợp và hiện thực hóa các chức năng thông qua mã nguồn Python. Trong chương này, em trình bày chi tiết quá trình phân tích thiết kế, mô hình hóa hệ thống, cấu trúc dữ liệu và các chức năng chính của chương trình.

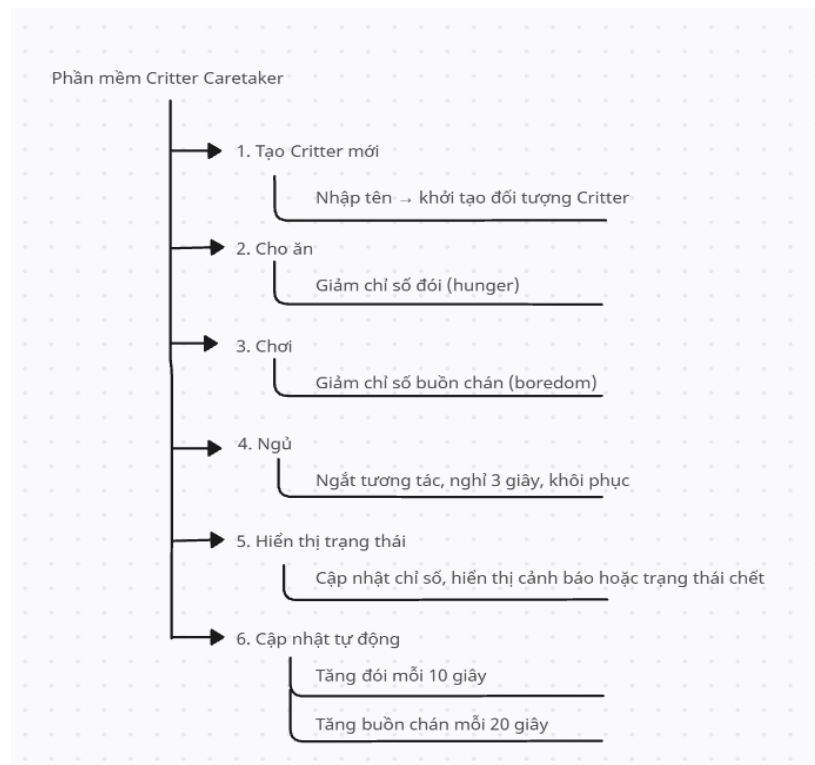
### 3.1. Phân tích tổng thể và sơ đồ hệ thống

#### 3.1.1. Phân tích chức năng hệ thống

Phần mềm Critter Caretaker được chia thành hai thành phần chính:

- Giao diện người dùng (GUI): Tạo và xử lý các tương tác với người dùng như nhập tên Critter, bấm nút cho ăn, chơi, ngủ và hiển thị thông tin.
- Logic xử lý Critter: Theo dõi, cập nhật và quản lý các trạng thái của Critter theo thời gian thực, đồng thời xử lý các tình huống như đói, chán, ngủ và chết.

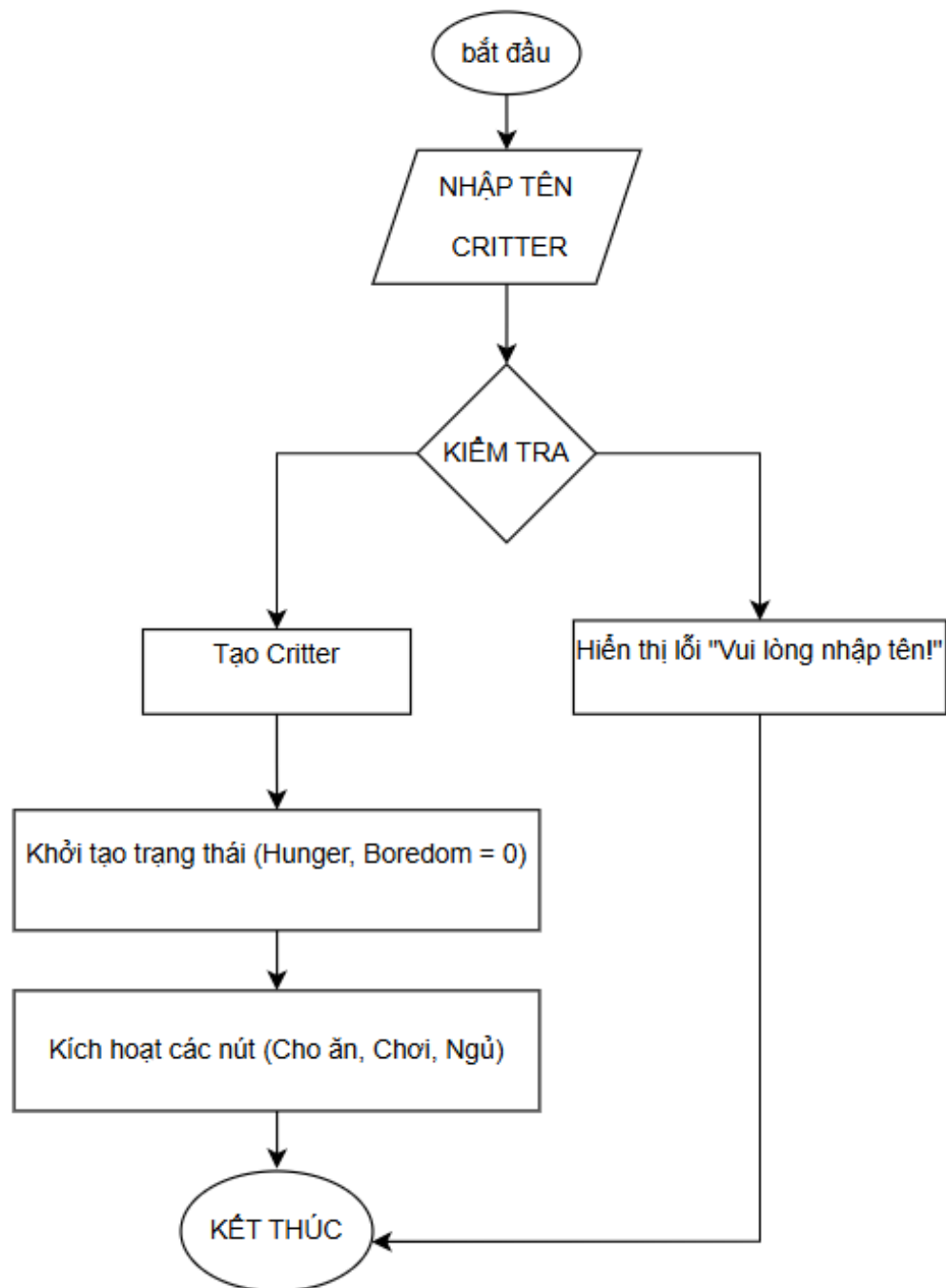
Từ hai thành phần này, ta có thể mô hình hóa hệ thống bằng sơ đồ phân cấp chức năng như sau:



Hình 3.1: Sơ đồ phân cấp chức năng của phần mềm Critter Caretaker

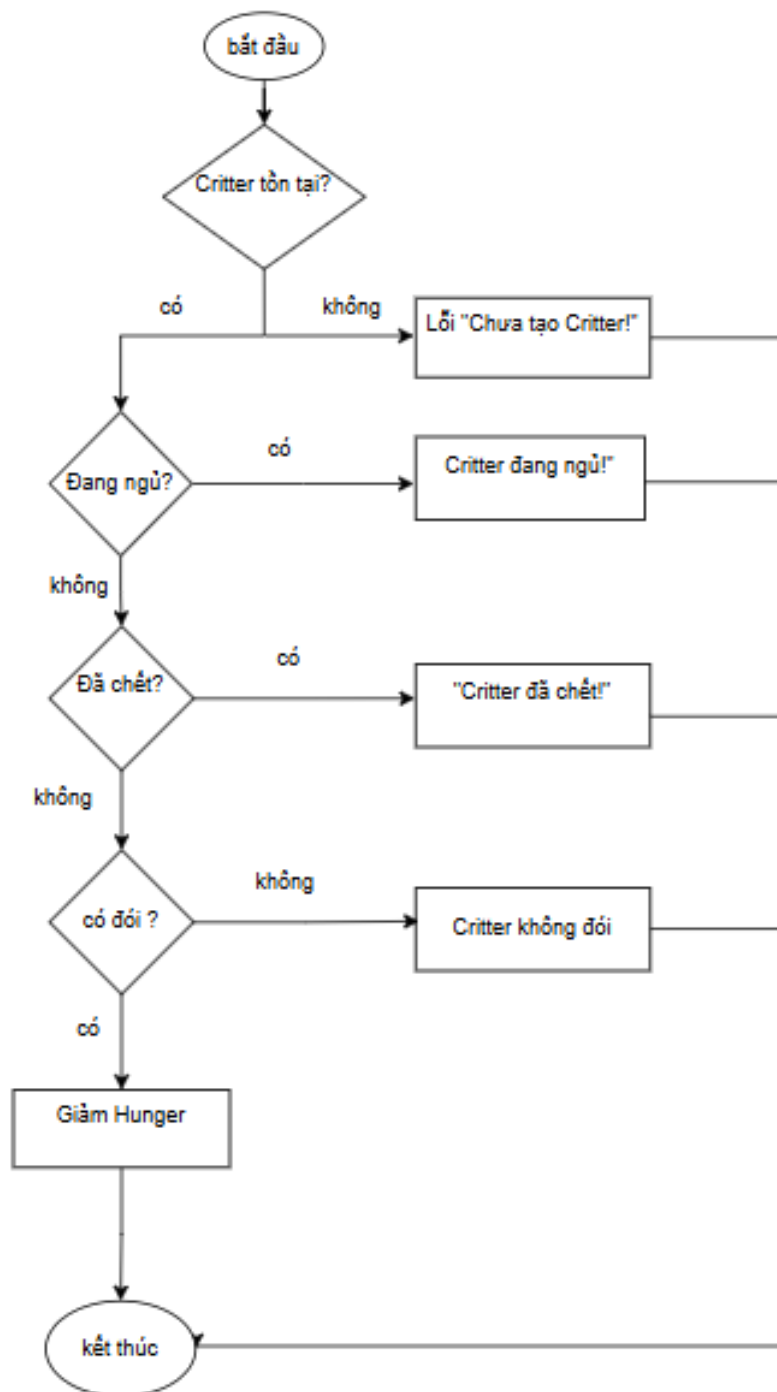
## 3.2. Sơ đồ khối thuật toán chính

### 3.2.1. Tạo thú



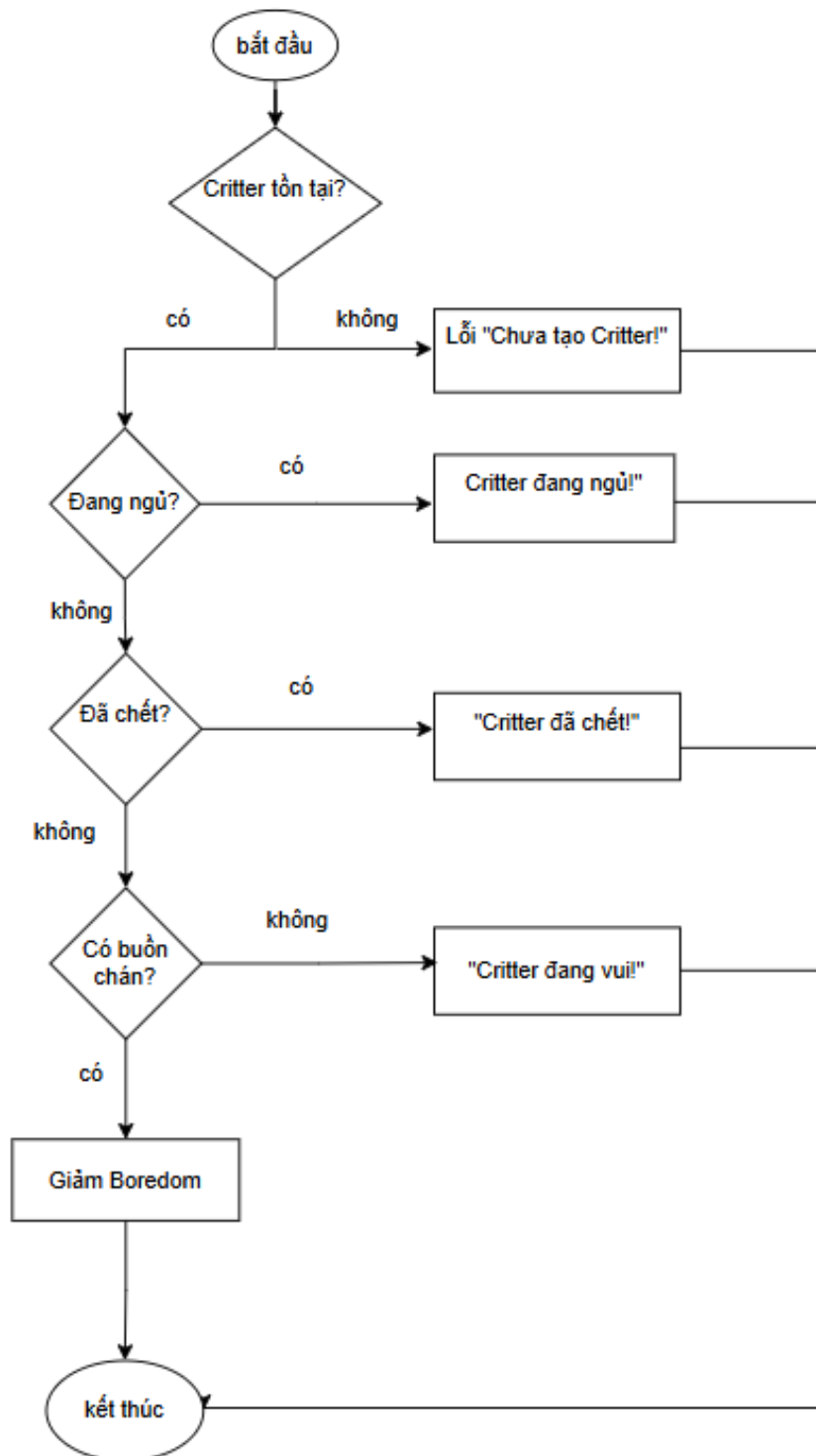
Hình 3.2 : Sơ đồ thuật toán khởi tạo trạng thái Critter

### 3.2.2. Cho ăn



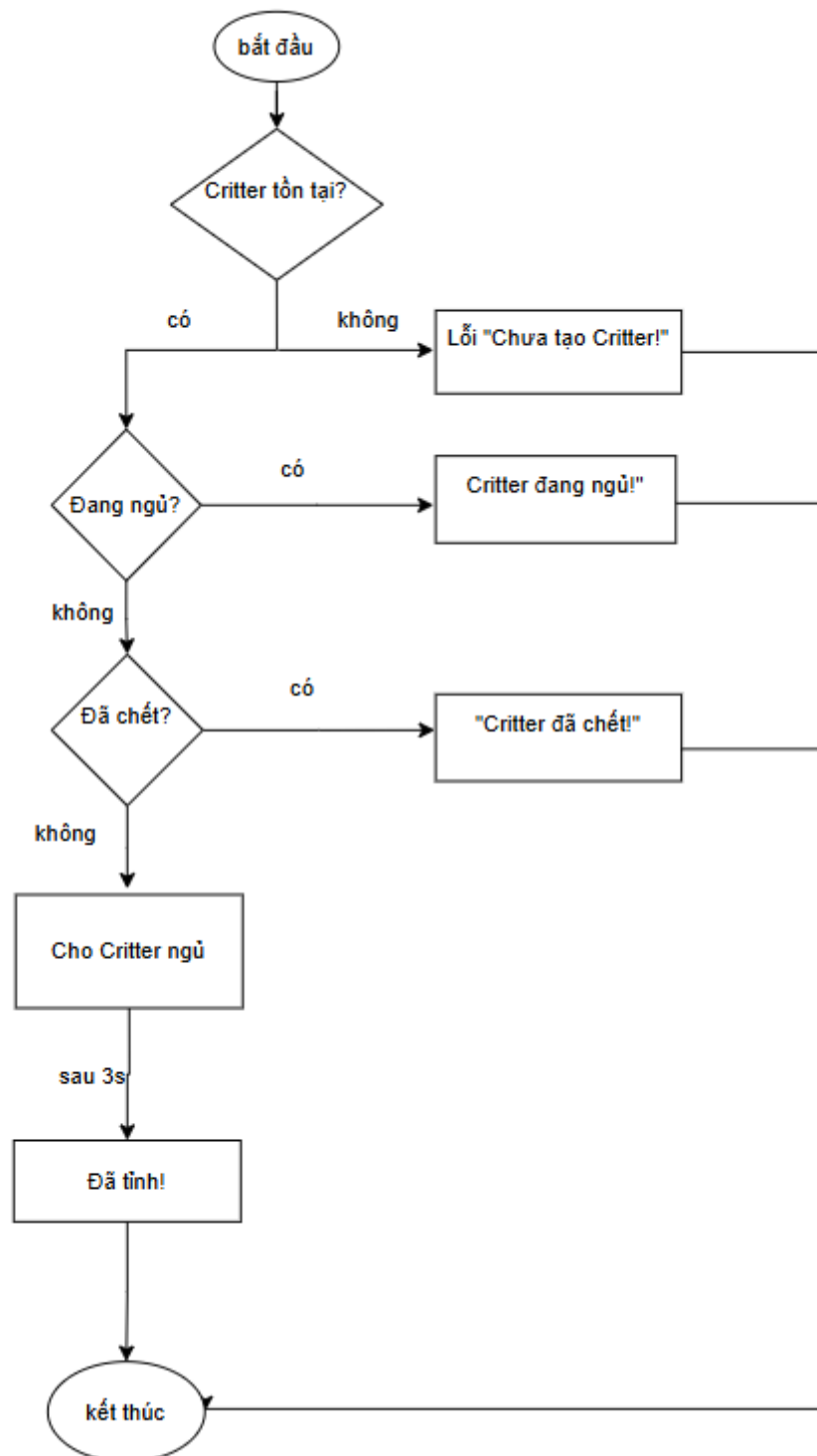
Hình 3.3 : Sơ đồ thuật toán cho Critter ăn

### 3.2.3. Chơi



Hình 3.4 : Sơ đồ thuật toán chơi với Critter

### 3.2.4. Ngủ

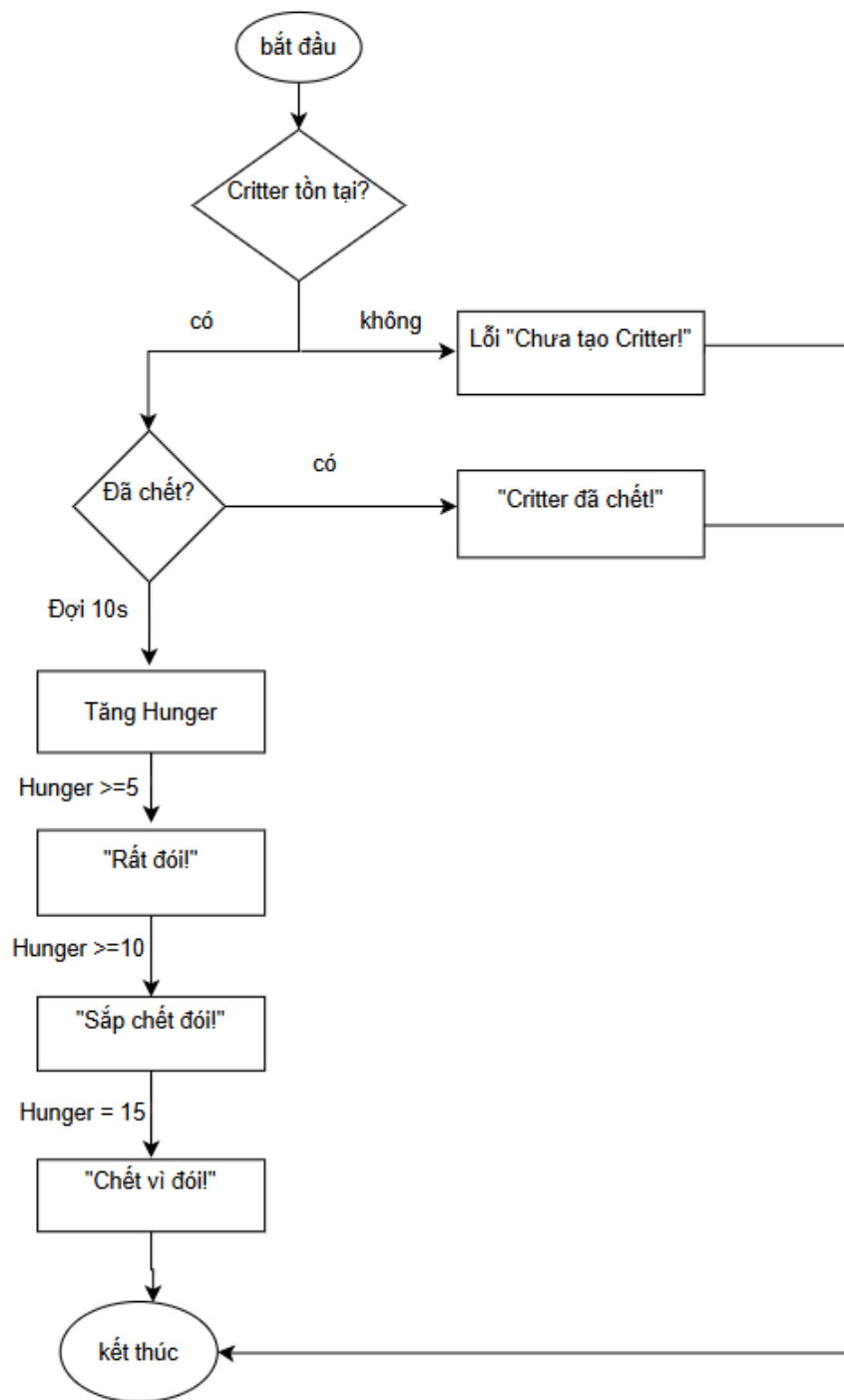


Hình 3.5 : Sơ đồ thuật toán cho Critter ngủ



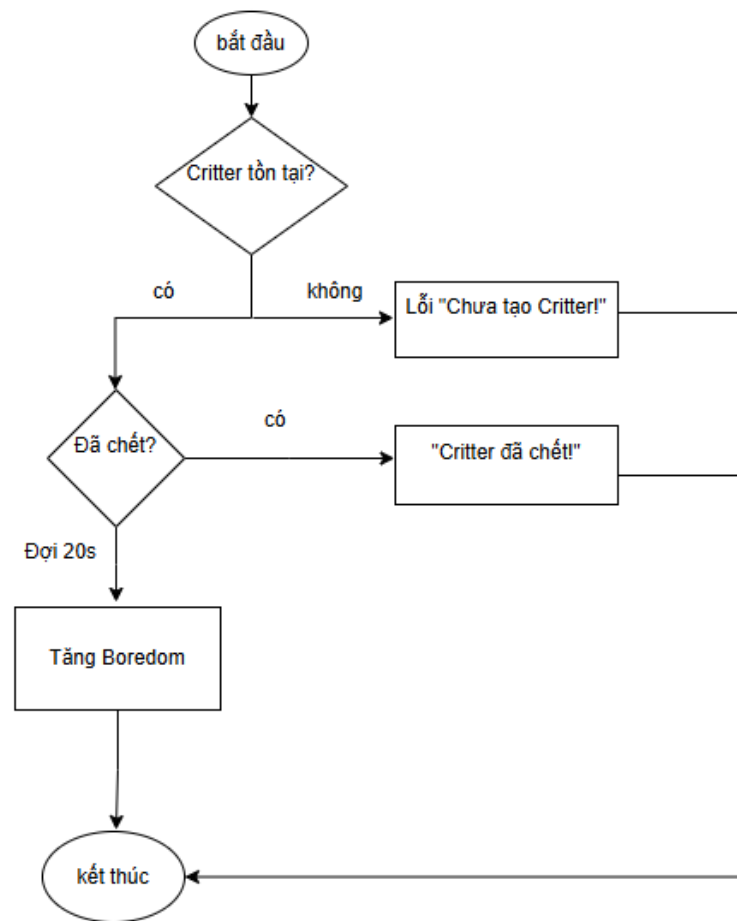
### 3.2.5. Tự động cập nhật

#### a. Tăng độ đói (auto\_increase\_hunger):



Hình 3.6: Sơ đồ thuật toán tự động cập nhật hunger

**b. Tăng độ buồn chán (auto\_increase\_boredom):**



*Hình 3.7 : Sơ đồ thuật toán tự động cập nhật boredom*

### **3.3. Cấu trúc dữ liệu**

Cấu trúc dữ liệu của Critter Caretaker được thiết kế đơn giản, sử dụng các biến cơ bản của Python để lưu trữ và quản lý trạng thái của thú ảo (Critter). Dữ liệu được quản lý hoàn toàn trong bộ nhớ chương trình, không sử dụng cơ sở dữ liệu bên ngoài hoặc lưu trữ vào file. Điều này có nghĩa là trạng thái của Critter sẽ bị mất khi chương trình kết thúc. Lý thuyết về cấu trúc dữ liệu được xây dựng để đảm bảo tính hiệu quả trong việc lưu trữ, truy cập, và cập nhật trạng thái, đồng thời hỗ trợ các chức năng mô phỏng thời gian thực và tương tác người dùng. Các biến chính được sử dụng bao gồm:

**a. Tên Critter (name):**

- **Kiểu dữ liệu:** Chuỗi (str).
- **Mô tả:** Lưu trữ tên của Critter do người dùng nhập thông qua ô nhập liệu (Entry) trong giao diện Tkinter. Tên này được sử dụng để cá nhân hóa các thông báo và hiển thị trạng thái, ví dụ: “Bông đang rất đói!”.
- **Vai trò:** Định danh duy nhất cho Critter, giúp người dùng nhận diện thú ảo của mình. Tên được hiển thị trong vùng trạng thái và các thông báo phản hồi.
- **Khởi tạo:** Được gán giá trị từ chuỗi người dùng nhập khi nhấn nút “Tạo Critter”. Nếu chuỗi rỗng, chương trình sẽ yêu cầu nhập lại thông qua thông báo lỗi.

**b. Mức độ đói (hunger):**

- **Kiểu dữ liệu:** Số nguyên (int).
- **Mô tả:** Biểu thị mức độ đói của Critter, bắt đầu từ 0 (không đói). Giá trị này tăng 1 đơn vị mỗi 10 giây nếu không được chăm sóc (thông qua hàm tự động cập nhật) và giảm 1 đơn vị khi người dùng nhấn nút “Cho ăn”.
- **Vai trò:** Là chỉ số chính để mô phỏng nhu cầu ăn uống tự nhiên của Critter. Giá trị này được kiểm tra để đưa ra các cảnh báo hoặc xác định trạng thái chết của Critter.
- **Ngưỡng quan trọng:**
  - $\text{hunger} \geq 5$ : Kích hoạt thông báo nhẹ để nhắc nhở người dùng chăm sóc.
  - $\text{hunger} \geq 10$ : Kích hoạt thông báo nghiêm trọng về nguy cơ chết.
  - $\text{hunger} \geq 15$ : Critter chuyển sang trạng thái chết ( $\text{is\_dead}=\text{True}$ ).
- **Hiển thị:** Giá trị được cập nhật liên tục trên vùng trạng thái của giao diện.

**c. Mức độ buồn chán (boredom):**

- **Kiểu dữ liệu:** Số nguyên (int).
- **Mô tả:** Biểu thị mức độ buồn chán của Critter, bắt đầu từ 0 (không buồn chán). Giá trị tăng 1 đơn vị mỗi 20 giây nếu không được chơi (thông qua hàm tự động cập nhật) và giảm 1 đơn vị khi người dùng nhấn nút “Chơi”.
- **Vai trò:** Mô phỏng trạng thái tâm lý của Critter, khuyến khích người dùng tương tác để giữ Critter vui vẻ. Mặc dù không dẫn đến chết như hunger, chỉ số này góp phần tăng tính thực tế của mô phỏng.

- **Hiển thị:** Giá trị được hiển thị trên vùng trạng thái, giúp người dùng theo dõi trạng thái tâm lý của Critter.

**d. Trạng thái sống/chết (is\_dead):**

- **Kiểu dữ liệu:** Boolean (bool).
- **Mô tả:** Biểu thị trạng thái sống (False) hoặc chết (True) của Critter. Mặc định là False khi tạo Critter, chuyển thành True khi hunger đạt hoặc vượt ngưỡng 15.
- **Vai trò:** Quyết định khả năng tương tác của Critter. Khi is\_dead=True, tất cả các hành động (cho ăn, chơi, ngủ) bị vô hiệu hóa, và giao diện hiển thị trạng thái chết kèm biểu tượng (☹️).
- **Hiển thị:** Được tích hợp vào vùng trạng thái, ví dụ: “Tên: Bông (☹️ Đã chết)” khi Critter chết.

**e. Trạng thái ngủ (is\_sleeping):**

- **Kiểu dữ liệu:** Boolean (bool).
- **Mô tả:** Biểu thị trạng thái ngủ (True) hoặc không ngủ (False) của Critter. Mặc định là False, chuyển thành True trong 3 giây khi người dùng nhấn nút “Ngủ”.
- **Vai trò:** Ngăn người dùng thực hiện các hành động khác (cho ăn, chơi) trong thời gian ngủ, mô phỏng trạng thái nghỉ ngơi. Khi is\_sleeping=True, các nút điều khiển bị vô hiệu hóa.
- **Hiển thị:** Trạng thái ngủ được phản ánh qua thông báo trên giao diện, ví dụ: “😴 Bông đang ngủ...”.

**f. Thời gian ngủ (sleep\_seconds):**

- **Kiểu dữ liệu:** Số nguyên (int).
- **Mô tả:** Lưu trữ số giây còn lại trong trạng thái ngủ, bắt đầu từ 3 và giảm dần về 0 trong quá trình đếm ngược khi Critter ngủ.
- **Vai trò:** Quản lý thời gian ngủ, đảm bảo Critter chỉ ngủ trong 3 giây. Biến này được sử dụng để cập nhật thông báo đếm ngược trên giao diện, ví dụ: “😴 Bông đang ngủ... [X] giây còn lại”.
- **Hiển thị:** Giá trị được tích hợp vào thông báo trạng thái ngủ, giúp người dùng theo dõi thời gian còn lại.

#### g. Quản lý dữ liệu:

- Tất cả các biến được lưu trữ trong bộ nhớ chương trình (RAM) và không được ghi vào file. Điều này có nghĩa là khi ứng dụng tắt, toàn bộ trạng thái của Critter (tên, hunger, boredom, v.v.) sẽ bị mất.
- Dữ liệu được truy cập và cập nhật thông qua các hàm xử lý, đảm bảo tính nhất quán giữa logic chương trình và giao diện hiển thị.
- Việc sử dụng các kiểu dữ liệu cơ bản của Python (str, int, bool) giúp đơn giản hóa việc quản lý trạng thái, phù hợp với quy mô nhỏ của đề tài.

#### h. Ưu điểm và hạn chế:

- **Ưu điểm:** Cấu trúc dữ liệu đơn giản, dễ triển khai, và hiệu quả cho một ứng dụng mô phỏng quy mô nhỏ. Các biến cơ bản của Python đảm bảo tốc độ xử lý nhanh và dễ tích hợp với Tkinter.
- **Hạn chế:** Do không lưu trữ vào file, dữ liệu bị mất khi tắt chương trình. Điều này giới hạn khả năng duy trì trạng thái lâu dài và không hỗ trợ tính năng như tải lại Critter cũ.

### 3.4. Các hàm trong chương trình chính

Chương trình Critter Caretaker được tổ chức thành các hàm riêng biệt, mỗi hàm xử lý một chức năng cụ thể, phù hợp với các sơ đồ khối thuật. Các hàm được thiết kế để tách biệt logic xử lý và giao diện, đảm bảo mã nguồn rõ ràng, dễ bảo trì, và dễ mở rộng. Dưới đây là lý thuyết về vai trò và chức năng của từng hàm chính:

#### a. `create_critter()`:

- **Mục đích:** Xử lý việc tạo Critter mới khi người dùng nhấn nút “Tạo Critter”.
- **Quy trình :**
  - Đọc chuỗi tên từ ô nhập liệu (Entry).
  - Kiểm tra xem tên có rỗng không; nếu rỗng, hiển thị thông báo lỗi yêu cầu nhập lại.
  - Nếu tên hợp lệ, khởi tạo các biến trạng thái: name (tên nhập), hunger=0, boredom=0, is\_dead=False.
  - Cập nhật giao diện: Hiển thị trạng thái Critter trên vùng trạng thái và thông báo “🐾 Đã tạo Critter tên [tên]!” trên vùng thông báo.

- Kích hoạt các nút điều khiển (“Cho ăn”, “Chơi”, “Ngủ”) bằng cách đặt trạng thái normal.
- Lập lịch gọi các hàm tự động cập nhật: tăng hunger sau 10 giây và tăng boredom sau 20 giây, sử dụng cơ chế after của Tkinter.
- **Vai trò:** Đảm bảo Critter được khởi tạo đúng và sẵn sàng cho các tương tác tiếp theo, đồng thời kích hoạt cơ chế thời gian thực.

#### b. `feed_critter()`:


- **Mục đích:** Xử lý hành động cho ăn khi người dùng nhấn nút “Cho ăn”.
- **Quy trình :**
  - Kiểm tra điều kiện: Nếu Critter chưa được tạo, đã chết, hoặc đang ngủ, hiển thị thông báo lỗi hoặc trạng thái tương ứng (ví dụ: “Chưa tạo Critter!”, “💀 Critter đã chết”, “😴 Critter đang ngủ”).
  - Nếu Critter hợp lệ, kiểm tra hunger:
    - Nếu  $\text{hunger}=0$ , hiển thị “Critter không đói!”.
    - Nếu  $\text{hunger}>0$ , giảm hunger đi 1 đơn vị và hiển thị “🍎 Đã cho [tên] ăn!”.
  - Cập nhật vùng trạng thái để hiển thị giá trị hunger mới.
- **Vai trò:** Mô phỏng hành động chăm sóc, giảm chỉ số đói và cung cấp phản hồi trực quan.

#### c. `play_critter()`:



- **Mục đích:** Xử lý hành động chơi khi người dùng nhấn nút “Chơi”.
- **Quy trình :**
  - Kiểm tra điều kiện: Nếu Critter chưa được tạo, đã chết, hoặc đang ngủ, hiển thị thông báo phù hợp.
  - Nếu Critter hợp lệ, kiểm tra boredom:
    - Nếu  $\text{boredom}=0$ , hiển thị “Critter đang vui vẻ rồi!”.
    - Nếu  $\text{boredom}>0$ , giảm boredom đi 1 đơn vị và hiển thị “🎲 Đã chơi với [tên]!”.
  - Cập nhật vùng trạng thái để hiển thị giá trị boredom mới.

- **Vai trò:** Mô phỏng hành động giải trí, giảm chỉ số buồn chán và tăng tính tương tác.


#### d. `sleep_critter()`:

- **Mục đích:** Xử lý hành động ngủ khi người dùng nhấn nút “Ngủ”.
- **Quy trình :**
  - Kiểm tra điều kiện: Nếu Critter chưa được tạo, đã chết, hoặc đang ngủ, hiển thị thông báo phù hợp.
  - Nếu hợp lệ, thiết lập trạng thái ngủ: Đặt `is_sleeping=True`, `sleep_seconds=3`, và vô hiệu hóa các nút điều khiển (`state="disabled"`).
  - Kích hoạt cơ chế đếm ngược thông qua hàm `countdown_sleep`.
  - Hiển thị thông báo ban đầu: “ [tên] đang ngủ... 3 giây còn lại”.
- **Vai trò:** Mô phỏng trạng thái nghỉ ngơi, tạm dừng các tương tác khác trong 3 giây.

#### e. `countdown_sleep()`:

- **Mục đích:** Quản lý quá trình đếm ngược thời gian ngủ.
- **Quy trình :**
  - Mỗi giây, giảm `sleep_seconds` đi 1 đơn vị.
  - Cập nhật thông báo: “ [tên] đang ngủ... [X] giây còn lại”.
  - Khi `sleep_seconds=0`, đặt `is_sleeping=False`, bật lại các nút điều khiển (`state="normal"`), và hiển thị “ [tên] đã tỉnh dậy!”.
  - Cập nhật vùng trạng thái để phản ánh trạng thái hiện tại.
- **Vai trò:** Đảm bảo thời gian ngủ được kiểm soát chính xác, với phản hồi trực quan mỗi giây.

#### f. `update_status()`:

- **Mục đích:** Cập nhật giao diện để hiển thị trạng thái hiện tại của Critter.
- **Quy trình:** Lấy giá trị hiện tại của `name`, `hunger`, `boredom`, và `is_dead`, sau đó hiển thị trên vùng trạng thái (Label) trong khung trạng thái (LabelFrame). Ví dụ: “Tên: Bông, Đói: 5, Buồn chán: 3” hoặc “Tên: Bông ( Đã chết)”.
- **Vai trò:** Đảm bảo giao diện luôn phản ánh chính xác trạng thái của Critter, đồng bộ với logic xử lý.

**g. auto\_increase\_hunger():**

- **Mục đích:** Tự động tăng chỉ số đói theo thời gian.
- **Quy trình :**
  - Lập lịch gọi sau mỗi 10 giây sử dụng after.
  - Nếu Critter còn sống và không ngủ, tăng hunger lên 1 đơn vị.
  - Kiểm tra ngưỡng: Hiển thị cảnh báo tại  $\text{hunger} \geq 5$  (“😞 Đang rất đói!”),  $\text{hunger} \geq 10$  (“⚠️ Sắp chết đói!”), và đặt `is_dead=True` tại  $\text{hunger} \geq 15$  với thông báo “💀 [tên] đã chết!” và vô hiệu hóa nút.
  - Cập nhật vùng trạng thái và lập lịch gọi lại hàm.
- **Vai trò:** Mô phỏng nhu cầu ăn uống tự nhiên, yêu cầu người dùng chăm sóc định kỳ.

**h. auto\_increase\_boredom():**

- **Mục đích:** Tự động tăng chỉ số buồn chán theo thời gian.
- **Quy trình :**
  - Lập lịch gọi sau mỗi 20 giây.
  - Nếu Critter còn sống và không ngủ, tăng boredom lên 1 đơn vị.
  - Cập nhật vùng trạng thái và lập lịch gọi lại hàm.
- **Vai trò:** Mô phỏng trạng thái tâm lý, khuyến khích người dùng tương tác để giữ Critter vui vẻ.

**i. disable\_buttons() / enable\_buttons():**

- **Mục đích:** Quản lý trạng thái của các nút điều khiển dựa trên trạng thái Critter.
- **Quy trình:**
  - `disable_buttons()`: Đặt trạng thái các nút thành disabled khi Critter ngủ hoặc chết, ngăn tương tác không hợp lệ.
  - `enable_buttons()`: Đặt trạng thái các nút thành normal khi Critter sẵn sàng tương tác (sống và không ngủ).
- **Vai trò:** Đảm bảo người dùng chỉ thực hiện các hành động hợp lệ, tăng tính ổn định của chương trình.

**j. on\_resize():**

- **Mục đích:** Điều chỉnh giao diện khi cửa sổ thay đổi kích thước.



- **Quy trình:** Phát hiện sự kiện thay đổi kích thước cửa sổ (sử dụng bind với <Configure>), sau đó điều chỉnh kích thước font chữ, khoảng cách, và bố cục của các widget (Label, Entry, Button) để đảm bảo giao diện rõ ràng trên mọi kích thước màn hình.
- **Vai trò:** Tăng tính linh hoạt và thân thiện của giao diện, đảm bảo trải nghiệm người dùng nhất quán.

### 3.5. Thiết kế giao diện

Giao diện được thiết kế sử dụng Tkinter với các thành phần sau, phù hợp với logic xử lý trong các sơ đồ:

- **Ô nhập liệu (Entry):** Nhận tên Critter từ người dùng, đặt ở vị trí trung tâm để dễ thao tác.
- **Nút điều khiển (Button):** Bốn nút: "Tạo Critter", "Cho ăn", "Chơi", và "Ngủ", được bố trí trong một Frame để căn chỉnh gọn gàng. Mỗi nút sử dụng màu sắc và biểu tượng cảm xúc riêng để phân biệt.
- **Vùng trạng thái (LabelFrame):** Hiển thị thông tin Critter (tên, hunger, boredom, trạng thái sống/chết) trong một khung có tiêu đề "Trạng thái Critter".
- **Vùng thông báo (Frame):** Hiển thị các thông báo phản hồi và cảnh báo, sử dụng emoji để tăng tính trực quan.
- **Hệ thống lưới (grid):** Sắp xếp các thành phần theo hàng và cột, đảm bảo giao diện linh hoạt và thích ứng với kích thước cửa sổ.

### Tóm tắt Chương 3

*Chương 3 đã trình bày lý thuyết thiết kế và xây dựng Critter Caretaker, tập trung vào phân tích chức năng, các sơ đồ khối thuật toán (tạo Critter, cho ăn, chơi, ngủ, tự động cập nhật), và cấu trúc dữ liệu sử dụng các biến Python cơ bản. Các hàm chính được thiết kế để xử lý logic và giao diện, đồng bộ với các sơ đồ. Giao diện sử dụng Tkinter với bố cục trực quan, hỗ trợ tương tác và phản hồi người dùng. Thiết kế này đảm bảo chương trình mô phỏng hành vi thú ảo một cách thực tế, phù hợp với yêu cầu đề tài.*

## CHƯƠNG 4: THỰC NGHIỆM VÀ KẾT LUẬN

Chương này trình bày lý thuyết về quá trình thực nghiệm để kiểm tra các chức năng của phần mềm Critter Caretaker, bao gồm việc tạo Critter, thực hiện các hành động (cho ăn, chơi, ngủ), và kiểm tra trạng thái chết. Nội dung cũng bao gồm các kết quả đạt được, bài học kinh nghiệm từ quá trình thực nghiệm, và các hướng phát triển tiềm năng trong tương lai, phù hợp với các yêu cầu và sơ đồ khối thuật toán được mô tả trong các chương trước.

### 4.1. Thực nghiệm

Thực nghiệm được tiến hành để kiểm tra tính đúng đắn và hiệu quả của các chức năng chính của phần mềm Critter Caretaker, bao gồm tạo Critter, cho ăn, chơi, ngủ, và xử lý trạng thái chết. Các bước thực nghiệm được thiết kế dựa trên các sơ đồ khối thuật toán để đảm bảo rằng mọi chức năng hoạt động như mong đợi và giao diện phản hồi chính xác các hành động của người dùng.

#### 4.1.1. Tạo Critter

**a. Mục đích:** Kiểm tra khả năng tạo một Critter mới với tên do người dùng nhập và khởi tạo trạng thái ban đầu.

**b. Quy trình :**

- Nhập tên “Bông” vào ô nhập liệu (Entry) và nhấn nút “Tạo Critter”.
- Kiểm tra xem chương trình có khởi tạo Critter với các giá trị ban đầu:
  - Tên: “Bông”.
  - Mức độ đói (hunger): 0.
  - Mức độ buồn chán (boredom): 0.
  - Trạng thái sống/chết (is\_dead): False.
- Kiểm tra giao diện:
  - Vùng trạng thái (LabelFrame) hiển thị thông tin: “Tên: Bông, Đói: 0, Buồn chán: 0”.
  - Vùng thông báo hiển thị: “🐾 Đã tạo Critter tên Bông!”.
  - Các nút điều khiển (“Cho ăn”, “Chơi”, “Ngủ”) được mở khóa (state="normal").

- Kiểm tra bộ đếm thời gian:
  - Bộ đếm tăng hunger bắt đầu sau 10 giây.
  - Bộ đếm tăng boredom bắt đầu sau 20 giây.
- c. **Kết quả mong đợi:** Critter được tạo thành công, giao diện cập nhật chính xác, và các bộ đếm thời gian được kích hoạt để mô phỏng trạng thái tự nhiên.



Hình 4.1: Thao tác tạo Critter

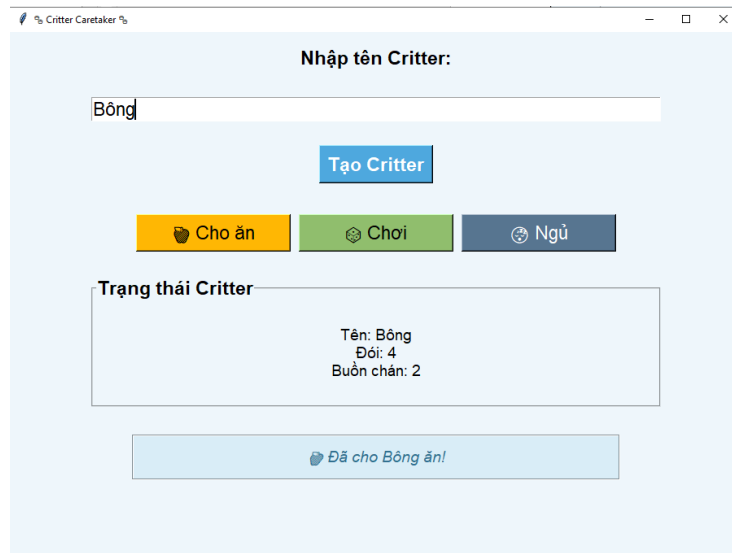
#### 4.1.2. Cho ăn

- a. **Mục đích:** Kiểm tra chức năng giảm mức độ đói của Critter khi nhấn nút “Cho ăn” và xử lý các trường hợp không hợp lệ.

b. **Quy trình :**

**Trường hợp 1: Critter đói ( $\text{hunger} \geq 5$ ):**

- Đợi đến khi  $\text{hunger} \geq 5$  (sau khoảng 50 giây nếu không tương tác).
- Nhấn nút “Cho ăn”.
- Kiểm tra: hunger giảm 1 đơn vị, ví dụ từ 5 xuống 4.
- Kiểm tra giao diện: Vùng trạng thái cập nhật giá trị hunger mới, vùng thông báo hiển thị “🍎 Đã cho Bông ăn!”.



Hình 4.2: Trường hợp Critter đói ( $\text{hunger} \geq 5$ )

### Trường hợp 2: Critter không đói ( $\text{hunger} = 0$ ):

- Nhấn nút “Cho ăn” ngay sau khi tạo Critter.
- Kiểm tra: Vùng thông báo hiển thị “Bông không đói!”.



Hình 4.3: Trường hợp Critter không đói ( $\text{hunger} = 0$ )

### Trường hợp 3: Critter đang ngủ hoặc đã chết:

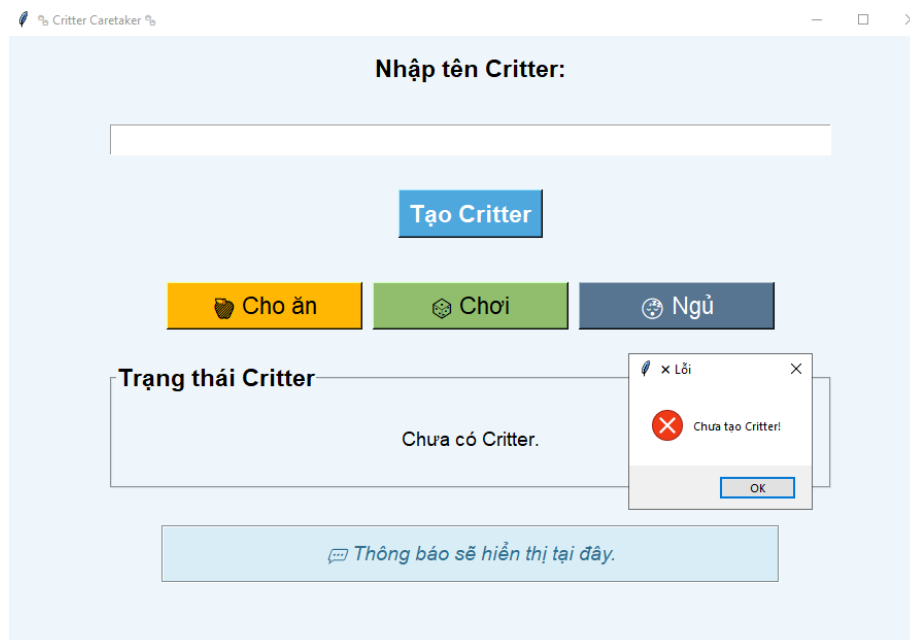
- Khi Critter ngủ ( $\text{is\_sleeping}=\text{True}$ ) hoặc chết ( $\text{is\_dead}=\text{True}$ ), nhấn nút “Cho ăn”.
- Kiểm tra: Vùng thông báo hiển thị “😴 Critter đang ngủ, hoặc “💀 Critter đã chết, không thể cho ăn.”.



Hình 4.4: Trường hợp Critter đang ngủ hoặc đã chết

#### Trường hợp 4: Chưa tạo Critter:

- Nhấn nút “Cho ăn” trước khi tạo Critter.
- Kiểm tra: Hộp thoại lỗi (messagebox.showerror) hiển thị “Chưa tạo Critter!”.



Hình 4.5: Trường hợp chưa tạo Critter

- c. **Kết quả mong đợi:** Chức năng cho ăn hoạt động đúng, giảm hunger khi hợp lệ, và cung cấp phản hồi rõ ràng cho các trường hợp không hợp lệ.

### 4.1.3. Chơi

**a. Mục đích:** Kiểm tra chức năng giảm mức độ buồn chán của Critter khi nhấn nút “Chơi” và xử lý các trường hợp không hợp lệ.

**b. Quy trình :**

- **Trường hợp 1: Critter buồn chán ( $boredom \geq 5$ ):**
  - Đợi đến khi  $boredom \geq 1$  (sau khoảng 20 giây nếu không tương tác).
  - Nhấn nút “Chơi”.
  - Kiểm tra: boredom giảm 1 đơn vị, ví dụ từ 1 xuống 0.
  - Kiểm tra giao diện: Vùng trạng thái cập nhật giá trị boredom mới, vùng thông báo hiển thị “🎲 Đã chơi với Bông!”.



Hình 4.6: Trường hợp Critter buồn chán ( $boredom \geq 5$ )

## Trường hợp 2: Critter không buồn chán (boredom = 0):

- Nhấn nút “Chơi” ngay sau khi tạo Critter.
- Kiểm tra: Vùng thông báo hiển thị “Bông đang vui vẻ rồi!”.



The screenshot shows the 'Critter Caretaker' application window. At the top, there's a title bar with the application name and standard window controls. Below the title bar, the main area has a light blue background. It starts with a label 'Nhập tên Critter:' followed by a text input field containing the name 'Bông'. Below the input field is a blue button labeled 'Tạo Critter'. Underneath this are three buttons: 'Cho ăn' (yellow), 'Chơi' (green), and 'Ngủ' (dark blue). Below these buttons is a section titled 'Trạng thái Critter' which contains a box displaying the following information: 'Tên: Bông', 'Đói: 0', and 'Buồn chán: 0'. At the bottom of the interface is a light blue status bar with the text 'Bông đang vui vẻ rồi!'.

Hình 4.7: Trường hợp Critter không buồn chán (boredom = 0)

## Trường hợp 3: Critter đang ngủ hoặc đã chết:

- Khi Critter ngủ hoặc chết, nhấn nút “Chơi”.
- Kiểm tra: Vùng thông báo hiển thị “😴 Critter đang ngủ” hoặc “💀 Critter đã chết, không thể chơi.”.

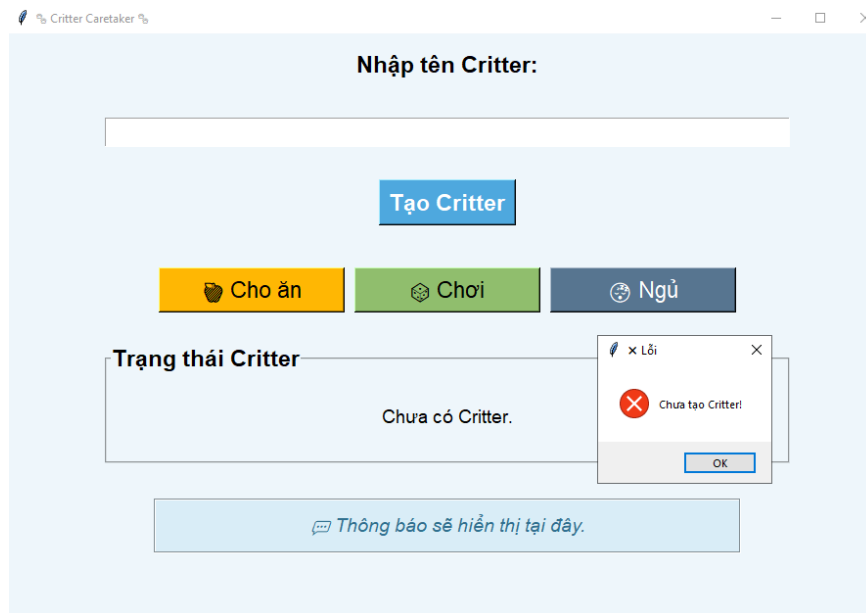


This screenshot shows the same 'Critter Caretaker' application window as in Figure 4.7, but with different state values. The 'Trạng thái Critter' box now displays: 'Tên: Bông', 'Đói: 4', and 'Buồn chán: 2'. The status bar at the bottom now shows '😴 Bông đang ngủ... 2 giây còn lại', indicating the critter is asleep.

Hình 4.8: Trường hợp Critter đang ngủ hoặc đã chết

#### Trường hợp 4: Chưa tạo Critter:

- Nhấn nút “Chơi” trước khi tạo Critter.
- Kiểm tra: Hộp thoại lỗi hiển thị “Chưa tạo Critter!”.



Hình 4.9: Trường hợp chưa tạo Critter

- c. **Kết quả mong đợi:** Chức năng chơi hoạt động đúng, giảm boredom khi hợp lệ, và cung cấp phản hồi phù hợp cho các trường hợp không hợp lệ.

#### 4.1.4. Ngủ

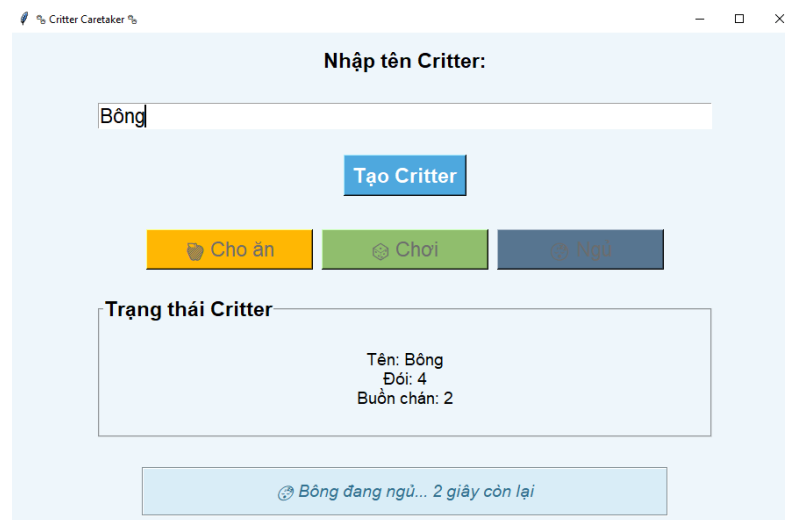
- a. **Mục đích:** Kiểm tra chức năng ngủ, đảm bảo Critter nghỉ ngơi trong 3 giây và khóa các hành động khác trong thời gian này.

b. **Quy trình:**

- Nhấn nút “Ngủ” khi Critter còn sống và không ngủ.
- Kiểm tra:
  - Biến `is_sleeping` được đặt thành `True`.
  - Biến `sleep_seconds` khởi tạo bằng 3.
  - Các nút điều khiển (“Cho ăn”, “Chơi”, “Ngủ”) bị vô hiệu hóa (`state="disabled"`).
- Kiểm tra đếm ngược:
  - Mỗi giây, vùng thông báo cập nhật: “😴 Bông đang ngủ... [X] giây còn lại” (X từ 3 xuống 0).



- Sau 3 giây, `is_sleeping` trở thành `False`, các nút được bật lại (`state="normal"`), và thông báo hiển thị “😴 Bông đã tỉnh dậy!”.
- Kiểm tra các trường hợp không hợp lệ:
  - Nhấn “Ngủ” khi Critter đang ngủ: Hiển thị “😴 Critter đang ngủ rồi!”.
  - Nhấn “Ngủ” khi Critter đã chết: Hiển thị “💀 Critter đã chết, không thể ngủ.”.
  - Nhấn “Ngủ” trước khi tạo Critter: Hiển thị hộp thoại lỗi “Chưa tạo Critter!”.



Hình 4.10: Cho Critter ngủ

- c. Kết quả mong đợi:** Chức năng ngủ hoạt động đúng, khóa các nút trong 3 giây, cập nhật thông báo đếm ngược, và khôi phục trạng thái sau khi ngủ xong.

#### 4.1.5. Kiểm tra trạng thái chết

- a. Mục đích:** Kiểm tra cơ chế tự động tăng hunger và xử lý trạng thái chết của Critter.

**b. Quy trình :**

- Không tương tác với Critter sau khi tạo (không nhấn “Cho ăn”).
- Kiểm tra bộ đếm thời gian tăng hunger (mỗi 10 giây):
  - Sau 50 giây ( $\text{hunger} \geq 5$ ): Vùng thông báo hiển thị “😞 Bông đang rất đói, hãy cho ăn đi!”.

- Sau 100 giây ( $\text{hunger} \geq 10$ ): Vùng thông báo hiển thị “⚠️ Bông sắp chết đói rồi!”.
- Sau 150 giây ( $\text{hunger} \geq 15$ ):
  - Biến `is_dead` được đặt thành True.
  - Vùng thông báo hiển thị “💀 Bông đã chết vì đói!”.
  - Các nút điều khiển bị vô hiệu hóa.
- Kiểm tra giao diện: Vùng trạng thái hiển thị “Tên: Bông (💀 Đã chết)”.



Hình 4.11 Critter đã chết

- c. Kết quả mong đợi:** Critter chết sau 150 giây nếu không được chăm sóc, giao diện cập nhật đúng trạng thái, và các chức năng bị khóa.

#### 4.1.6. Ghi nhận thực nghiệm

- a. Chụp ảnh giao diện:** Các bước thực nghiệm (tạo Critter, cho ăn, chơi, ngủ, và trạng thái chết) cần được ghi lại bằng ảnh chụp màn hình để minh họa:
- Giao diện ban đầu với ô nhập tên và nút “Tạo Critter”.
  - Giao diện sau khi tạo Critter, hiển thị trạng thái ban đầu.
  - Giao diện khi nhấn “Cho ăn” hoặc “Chơi” với thông báo tương ứng.
  - Giao diện khi Critter ngủ, hiển thị đếm ngược.
  - Giao diện khi Critter chết, với các nút bị khóa.

- b. Mục đích ghi nhận:** Đảm bảo các chức năng hoạt động đúng như thiết kế và cung cấp bằng chứng trực quan cho kết quả thực nghiệm.

## 4.2. Kết luận

### 4.2.1. Kết quả đạt được

Quá trình thực nghiệm cho thấy phần mềm Critter Caretaker đáp ứng đầy đủ các yêu cầu lý thuyết:

- a. Hoàn thiện chương trình:** Xây dựng thành công một ứng dụng nuôi thú ảo với giao diện đồ họa sử dụng Tkinter, tích hợp các chức năng tạo Critter, cho ăn, chơi, ngủ, và cập nhật trạng thái thời gian thực.

**b. Chức năng hoạt động đúng:**

- Tạo Critter với tên do người dùng nhập và trạng thái ban đầu (hunger=0, boredom=0, is\_dead=False).
- Hành động “Cho ăn” và “Chơi” giảm đúng chỉ số hunger và boredom.
- Chức năng “Ngủ” khóa các nút trong 3 giây và hiển thị đếm ngược.
- Cơ chế tự động tăng hunger (10 giây) và boredom (20 giây) hoạt động ổn định.
- Cảnh báo hiển thị đúng tại các ngưỡng  $\text{hunger} \geq 5$ ,  $\geq 10$ , và Critter chết khi  $\text{hunger} \geq 15$ .

- c. Giao diện đồng bộ:** Vùng trạng thái và thông báo cập nhật liên tục, sử dụng màu sắc và biểu tượng cảm xúc để tăng tính trực quan.

- d. Xử lý lỗi:** Chương trình xử lý các trường hợp không hợp lệ (chưa tạo Critter, Critter ngủ, Critter chết) bằng thông báo và hộp thoại lỗi, đảm bảo không xảy ra crash.

### 4.2.2. Bài học kinh nghiệm

Quá trình thực nghiệm mang lại các bài học lý thuyết quan trọng trong lập trình Python:

- a. Quản lý thời gian thực:** Sử dụng phương thức `after()` của Tkinter để lập lịch các tác vụ định kỳ (tăng hunger, boredom, đếm ngược khi ngủ) giúp hiểu rõ cách xử lý các tác vụ nền trong ứng dụng GUI.

- b. Đồng bộ logic và giao diện:** Việc cập nhật trạng thái Critter và hiển thị trên giao diện đòi hỏi sự phối hợp chặt chẽ giữa logic xử lý (biến, hàm) và các widget Tkinter (Label, Frame), nhấn mạnh tầm quan trọng của đồng bộ dữ liệu.
- c. Tổ chức mã nguồn:** Phân chia chương trình thành các hàm riêng biệt (tạo, cho ăn, chơi, ngủ, cập nhật trạng thái) giúp mã dễ đọc, dễ bảo trì, và dễ mở rộng.
- d. Xử lý tương tác người dùng:** Sử dụng messagebox và thông báo trạng thái để phản hồi hành động người dùng, giúp cải thiện trải nghiệm và xử lý lỗi hiệu quả.
- e. Thiết kế giao diện trực quan:** Việc sử dụng màu sắc, biểu tượng cảm xúc, và bố cục lưới trong Tkinter làm tăng tính thân thiện, nhấn mạnh vai trò của thiết kế giao diện trong ứng dụng.

#### 4.2.3. Hướng phát triển tương lai

Dựa trên kết quả thực nghiệm, phần mềm Critter Caretaker có thể được mở rộng với các tính năng lý thuyết sau:

- a. Lưu trữ trạng thái:** Thêm khả năng lưu trạng thái Critter (tên, hunger, boredom) vào file (ví dụ: sử dụng JSON hoặc CSV) để giữ dữ liệu khi tắt chương trình, và cho phép tải lại trạng thái khi khởi động lại.
- b. Hỗ trợ nhiều Critter:** Cho phép người dùng quản lý đồng thời nhiều Critter, với giao diện hiển thị trạng thái riêng cho từng Critter, đòi hỏi thiết kế lại cấu trúc dữ liệu và giao diện.
- c. Tính năng nâng cao:** Bổ sung các hành động như tắm, khám bệnh, hoặc đi dạo, với các chỉ số mới (sức khỏe, năng lượng) để tăng tính mô phỏng.
- d. Tích hợp đa phương tiện:** Thêm hình ảnh động (GIF) hoặc âm thanh (âm thanh ăn, ngủ) để làm giao diện sinh động hơn, sử dụng các thư viện hỗ trợ như PIL (cho hình ảnh) hoặc pygame (cho âm thanh).
- e. Tương tác nhiều người dùng:** Phát triển phiên bản mạng, cho phép nhiều người dùng chăm sóc Critter chung hoặc chia sẻ trạng thái Critter qua mạng, đòi hỏi tích hợp công nghệ client-server.

- f. Giao diện hiện đại:** Cải tiến giao diện với các hiệu ứng chuyển động, màu sắc gradient, hoặc bố cục phức tạp hơn, có thể sử dụng các thư viện GUI nâng cao như ttk trong Tkinter.

#### 4.3. Lời cảm ơn.

Em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Văn Huy, giảng viên hướng dẫn môn Lập trình Python, người đã tận tình chỉ bảo, cung cấp tài liệu và đưa ra những góp ý quý giá để em hoàn thiện đề tài "Xây dựng ứng dụng Critter Caretaker". Sự hướng dẫn tận tâm của thầy đã giúp em nắm vững kiến thức về Python, Tkinter và áp dụng hiệu quả vào việc phát triển phần mềm, đồng thời khơi dậy niềm đam mê học tập và sáng tạo trong lĩnh vực lập trình.

#### *Tóm tắt Chương 4*

*Chương 4 đã trình bày lý thuyết về quá trình thực nghiệm Critter Caretaker, kiểm tra các chức năng chính (tạo Critter, cho ăn, chơi, ngủ, trạng thái chết) dựa trên các sơ đồ khối thuật toán. Kết quả thực nghiệm cho thấy chương trình hoạt động đúng, với giao diện đồng bộ, phản hồi rõ ràng, và xử lý lỗi hiệu quả. Các bài học kinh nghiệm bao gồm quản lý thời gian thực, đồng bộ logic-giao diện, và tổ chức mã nguồn. Hướng phát triển tương lai đề xuất lưu trữ trạng thái, hỗ trợ nhiều Critter, và tích hợp đa phương tiện để nâng cao tính năng và trải nghiệm người dùng.*

## TÀI LIỆU THAM KHẢO

1. Michael Dawson, *Python Programming for the Absolute Beginner*, 3rd Edition, 2010. Hướng dẫn lập trình Python và game Pizza Panic với Pygame.
2. Pygame Documentation, <https://www.pygame.org/docs/>. Tài liệu chính thức về Pygame, mô tả module, sprite, sự kiện.
3. Al Sweigart, *Making Games with Python & Pygame*, 2012. Hướng dẫn xây dựng game 2D với Pygame.
4. Nguyễn Văn Huy, Tài liệu môn Lập trình Python, Trường ĐH Kỹ thuật Công nghiệp, 2025. Cung cấp kiến thức Python và Pygame.
5. Real Python, <https://realpython.com>. Hướng dẫn Python và Pygame chi tiết.
6. Stack Overflow, <https://stackoverflow.com>. Giải pháp cho quản lý sprite, va chạm trong Pygame.