TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP

KHOA ĐIỆN TỬ

BÀI TẬP LỚN

LẬP TRÌNH PYTHON

NGÀNH : KỸ THUẬT MÁY TÍNH

HỆ : ĐẠI HỌC CHÍNH QUY

THÁI NGUYÊN - 2025

TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP

KHOA ĐIỆN TỬ

BÀI TẬP LỚN

LẬP TRÌNH PYTHON

BỘ MÔN: CÔNG NGHỆ THÔNG TIN

GIÁO VIÊN GIẢNG DẠY : TS. NGUYỄN VĂN HUY

LỚP : K58KTP.K01

SINH VIÊN THỰC HIỆN : LƯỜNG VĂN HẠNH

LINK GITHUB :

https://github.com/LuongHanh/BTL_Python

THÁI NGUYÊN – 2025

TRƯỜNG ĐHKTCN
KHOA ĐIỆN TỬ

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM
Độc lập - Tự do - Hạnh phúc

BÀI TẬP KẾT THÚC MÔN HỌC

MÔN HỌC: LẬP TRÌNH PYTHON

BỘ MÔN : CÔNG NGHỆ THÔNG TIN

Sinh viên: LƯỜNG VĂN HẠNH

Lớp : K58KTP.K01 *Ngành: Kỹ thuật máy tính*

Giáo viên hướng dẫn: Ts. Nguyễn Văn Huy

Ngày giao đề *20/05/2025* *Ngày hoàn thành*

Tên đề tài : Critter Caretaker với GUI

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.*

GIÁO VIÊN HƯỚNG DẪN

(Ký và ghi rõ họ tên)

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Thái Nguyên, ngày….tháng…..năm 20....

GIÁO VIÊN HƯỚNG DẪN

(Ký ghi rõ họ tên)

LỜI MỞ ĐẦU

Trong thời đại công nghệ phát triển nhanh chóng, việc xây dựng các ứng dụng tương tác với người dùng thông qua giao diện đồ họa (GUI) ngày càng trở nên phổ biến và cần thiết. Bài tập lớn môn Lập trình Python với đề tài "Critter Caretaker với GUI" là một cơ hội tốt để sinh viên vận dụng các kiến thức đã học về lập trình hướng đối tượng, xử lý sự kiện, và thiết kế giao diện với thư viện Tkinter để phát triển một ứng dụng hoàn chỉnh.

Đề tài yêu cầu xây dựng một hệ thống tương tác với "critter" – một đối tượng mô phỏng thú cưng ảo – cho phép người dùng tạo mới, cho ăn, cho chơi hoặc cho ngủ. Thông qua quá trình phát triển chương trình, sinh viên không chỉ rèn luyện kỹ năng lập trình mà còn nâng cao khả năng phân tích, thiết kế hệ thống và xử lý ngoại lệ trong các tình huống thực tế.

Báo cáo này trình bày quá trình thiết kế, xây dựng và kiểm thử ứng dụng Critter Caretaker, từ cơ sở lý thuyết đến hiện thực hóa chương trình và đánh giá kết quả. Mục tiêu là tạo ra một sản phẩm vừa đáp ứng yêu cầu đề bài, vừa thể hiện được tư duy lập trình và tính sáng tạo cá nhân.

MỤC LỤC

CHƯƠNG I : GIỚI THIỆU ĐẦU BÀI

Trong bài tập lớn cuối môn này, em được giao thực hiện đề tài số 4: Xây dựng ứng dụng "Critter Caretaker" với giao diện đồ họa (GUI). Đây là một bài toán mô phỏng việc chăm sóc một sinh vật ảo (critter) thông qua các hành động như cho ăn, chơi, ngủ… thông qua giao diện người dùng do người lập trình thiết kế.

Chương trình sẽ cho phép người dùng:

- Nhập tên để tạo một Critter mới.
- Thực hiện các hành động tương tác với Critter như:
  - Cho ăn (giảm mức độ đói – hunger),
  - Chơi (giảm sự chán – boredom),
  - Ngủ (đặt lại trạng thái Critter).
- Hiển thị trạng thái hiện tại của Critter gồm mức độ đói và chán.
- Cập nhật trạng thái mỗi khi người dùng thực hiện hành động.
- Báo lỗi nếu người dùng cố gắng tương tác khi chưa tạo Critter.

Các tính năng chính:

- Giao diện trực quan với Tkinter (nhập tên, các nút điều khiển, trạng thái).
- Mô hình hóa Critter bằng lập trình hướng đối tượng (class, thuộc tính, phương thức).
- Xử lý ngoại lệ khi chưa tạo Critter hoặc nhập tên rỗng.
- Tự động thay đổi trạng thái hunger/boredom theo thời gian hoặc sau mỗi hành động (tùy chọn nâng cao).

Thách thức của bài toán:

Mặc dù đề tài không yêu cầu thuật toán phức tạp, nhưng việc triển khai một ứng dụng tương tác giữa người dùng và sinh vật ảo vẫn đặt ra một số thách thức nhất định trong quá trình lập trình. Để hoàn thành bài toán một cách hiệu quả, người thực hiện cần giải quyết tốt các khía cạnh sau:

- Tổ chức class Critter hợp lý, đảm bảo khả năng mở rộng và kiểm soát trạng thái.

- Xây dựng GUI trực quan và dễ sử dụng bằng Tkinter, kết nối logic với giao diện.
- Quản lý trạng thái nội tại của Critter sao cho phản ánh đúng quá trình tương tác của người dùng.
- Xử lý ngoại lệ và đảm bảo tính ổn định của chương trình khi người dùng nhập sai hoặc thao tác bất hợp lý.

Để hoàn thành đề tài "Critter Caretaker" với giao diện đồ họa, em đã vận dụng nhiều kiến thức đã học trong học phần, đặc biệt là các khái niệm về lập trình hướng đối tượng trong Python. Việc xây dựng một lớp Critter với các thuộc tính như mức độ đói (hunger), mức độ chán (boredom), và các phương thức xử lý hành vi như ăn, chơi, ngủ đã giúp em hiểu rõ hơn về cách tổ chức dữ liệu và chức năng trong một đối tượng. Mỗi hành động được thực hiện trên Critter đều tác động đến trạng thái của nó, thể hiện rõ đặc điểm đóng gói và quản lý trạng thái nội tại – một nguyên tắc cơ bản của lập trình hướng đối tượng.

Bên cạnh đó, em đã áp dụng thư viện Tkinter để xây dựng giao diện đồ họa (GUI) cho chương trình. Các kiến thức về cách tạo cửa sổ, bố trí các thành phần giao diện như nút bấm (Button), ô nhập liệu (Entry), nhãn hiển thị (Label), cũng như cách gắn sự kiện (event binding) cho các nút chức năng, đã được em vận dụng để hoàn thiện phần giao diện của ứng dụng. Tkinter không chỉ giúp chương trình dễ sử dụng hơn mà còn là cầu nối giữa người dùng và logic xử lý của hệ thống.

Ngoài ra, việc xử lý lỗi và kiểm tra đầu vào cũng là một phần quan trọng trong quá trình phát triển ứng dụng. Em đã sử dụng các cấu trúc điều kiện và hộp thoại thông báo (messagebox) để kiểm soát tình huống khi người dùng chưa tạo Critter mà đã thao tác, hoặc khi nhập tên Critter không hợp lệ. Điều này giúp chương trình trở nên thân thiện và an toàn hơn khi sử dụng.

Cuối cùng, em cũng được rèn luyện kỹ năng quản lý trạng thái của đối tượng trong suốt vòng đời hoạt động. Mỗi hành động của người dùng đều làm thay đổi trạng thái của Critter, và các trạng thái này được cập nhật và hiển thị liên tục trên giao diện. Việc xây dựng một cơ chế cập nhật trạng thái đơn giản nhưng hiệu quả giúp em hiểu rõ tầm quan trọng của việc duy trì tính nhất quán trong thiết kế phần mềm.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1. Lập trình hướng đối tượng với Python

Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một phương pháp thiết kế phần mềm xoay quanh khái niệm "đối tượng" – những thực thể chứa dữ liệu và hành vi. Trong Python, OOP là một đặc điểm cốt lõi và được hỗ trợ một cách linh hoạt, cho phép lập trình viên xây dựng các chương trình rõ ràng, dễ mở rộng và dễ bảo trì.

Một chương trình hướng đối tượng được tổ chức xoay quanh hai yếu tố: lớp (class) và đối tượng (object). Lớp có thể được hiểu như một bản thiết kế, trong đó mô tả các thuộc tính (biến) và hành vi (phương thức) của một kiểu đối tượng cụ thể. Đối tượng là thể hiện (instance) cụ thể của lớp – mỗi đối tượng mang các giá trị dữ liệu riêng biệt và có thể tương tác thông qua các phương thức được định nghĩa trong lớp.

Python cho phép định nghĩa lớp bằng từ khóa class, trong đó phương thức __init__() đóng vai trò là constructor – tự động được gọi khi đối tượng được khởi tạo. Các thuộc tính có thể được phân biệt thành thuộc tính lớp (class attributes) – chia sẻ giữa tất cả các đối tượng, và thuộc tính đối tượng (instance attributes) – chỉ thuộc về từng thể hiện cụ thể. Việc truy cập và thao tác với dữ liệu bên trong được thực hiện thông qua từ khóa self, đóng vai trò là tham chiếu đến chính đối tượng đó trong phạm vi lớp.

Một trong những đặc điểm nổi bật của OOP là khả năng đóng gói (encapsulation). Điều này có nghĩa là dữ liệu và các phương thức thao tác với dữ liệu đó được gói gọn trong một đơn vị duy nhất (đối tượng), giúp kiểm soát việc truy cập và thay đổi dữ liệu. Trong Python, quy ước sử dụng dấu gạch dưới (_ hoặc __) để biểu thị mức độ truy cập của các thuộc tính (ví dụ: private, protected) là một công cụ đơn giản nhưng hiệu quả trong việc thực thi đóng gói.

Kế thừa (inheritance) là một đặc trưng quan trọng khác của lập trình hướng đối tượng, cho phép lớp con kế thừa thuộc tính và phương thức từ lớp cha mà không cần viết lại. Cơ chế này hỗ trợ tái sử dụng mã, đồng thời cho phép mở rộng hệ thống mà không phá vỡ cấu trúc sẵn có. Python cung cấp công cụ super() để gọi đến phương thức của lớp cha từ trong lớp con – thường được dùng trong quá trình khởi tạo hoặc ghi đè (override) hành vi của phương thức.

Một nguyên lý khác góp phần tạo nên tính linh hoạt trong OOP là đa hình (polymorphism). Với đa hình, các lớp khác nhau có thể định nghĩa những phương thức cùng tên nhưng thực hiện chức năng khác nhau. Điều này cho phép người lập trình sử dụng chung một giao diện để thao tác với nhiều kiểu đối tượng khác nhau, từ đó giảm thiểu sự phụ thuộc cứng nhắc trong mã nguồn và tăng khả năng mở rộng.

Bên cạnh đó, OOP trong Python cũng nhấn mạnh đến tính trừu tượng (abstraction) – tức là che giấu những chi tiết cài đặt bên trong lớp và chỉ cung cấp những gì cần thiết để người dùng tương tác. Mặc dù Python không hỗ trợ trừu tượng hóa dữ liệu theo cách cứng nhắc như một số ngôn ngữ khác (ví dụ: Java, C++), nhưng vẫn có thể thực hiện được thông qua thiết kế lớp hợp lý và các quy ước truy cập thuộc tính.

Tóm lại, lập trình hướng đối tượng trong Python là một công cụ mạnh mẽ giúp người phát triển phần mềm xây dựng các ứng dụng có cấu trúc chặt chẽ, dễ kiểm soát, dễ mở rộng và phù hợp với các bài toán mô phỏng thế giới thực. Việc vận dụng thành thạo OOP không chỉ là một yêu cầu kỹ thuật mà còn là nền tảng tư duy để thiết kế những hệ thống phần mềm linh hoạt và bền vững.

2.2. Thư viện Tkinter và xử lý giao diện người dùng

Tkinter là thư viện tiêu chuẩn trong Python được sử dụng để xây dựng giao diện đồ họa người dùng (GUI – Graphical User Interface). Đây là một công cụ mạnh mẽ, đơn giản và dễ tiếp cận giúp lập trình viên tạo ra các ứng dụng tương tác trực tiếp với người dùng thông qua các thành phần đồ họa như nút bấm, hộp văn bản, nhãn hiển thị, hộp thoại thông báo, v.v.

Tkinter hoạt động như một lớp bao bọc (wrapper) của thư viện GUI gốc là Tcl/Tk, giúp người lập trình có thể sử dụng các chức năng đồ họa một cách thuận tiện trong Python. Khi sử dụng Tkinter, ứng dụng sẽ bao gồm một cửa sổ chính (main window) được tạo bởi đối tượng Tk(). Trên cửa sổ này, các widget (thành phần giao diện) được bố trí và quản lý theo các layout như pack, grid hoặc place, tùy thuộc vào mục đích và độ phức tạp của giao diện.

Một trong những điểm mạnh của Tkinter là khả năng kết nối giữa các thành phần giao diện và logic xử lý bên trong chương trình thông qua sự kiện (events) và hàm callback. Người dùng có thể gán các hành động cụ thể (như nhấn nút, nhập dữ liệu, di

chuyển chuột...) với các hàm xử lý được định nghĩa sẵn. Điều này tạo nên tính tương tác, phản hồi theo thời gian thực và trải nghiệm người dùng tốt hơn.

Quá trình xây dựng giao diện với Tkinter thường bao gồm các bước cơ bản sau: khởi tạo cửa sổ chính, tạo và cấu hình các widget (ví dụ: Button, Label, Entry, Text, Frame,...), gắn các sự kiện (event binding), và cuối cùng là khởi chạy vòng lặp sự kiện bằng phương thức mainloop(). Vòng lặp này giữ cho giao diện luôn hiển thị và chờ người dùng thao tác.

Ngoài ra, Tkinter còn hỗ trợ hiển thị thông báo bằng các hộp thoại (messagebox), tổ chức giao diện theo nhiều khung (Frame) để chia layout hợp lý, và cho phép cập nhật trạng thái các thành phần một cách linh hoạt thông qua các biến StringVar, IntVar, v.v. Những tính năng này rất hữu ích trong việc xây dựng các ứng dụng có trạng thái động, cần phản hồi nhanh theo hành động của người dùng.

Mặc dù không phải là thư viện GUI hiện đại nhất, Tkinter vẫn giữ vai trò quan trọng trong giáo dục và các ứng dụng nhỏ đến trung bình nhờ tính dễ học, gọn nhẹ và có sẵn trong hầu hết các bản cài đặt Python. Việc sử dụng thành thạo Tkinter không chỉ giúp sinh viên hiểu rõ nguyên lý xử lý giao diện mà còn là nền tảng để tiếp cận các công cụ nâng cao hơn như PyQt, Kivy, hoặc web frontend.

Tóm lại, Tkinter cung cấp một môi trường phát triển GUI hiệu quả, cho phép kết nối mạch lạc giữa giao diện và xử lý logic bên trong chương trình. Trong bối cảnh đề tài "Critter Caretaker", việc vận dụng Tkinter giúp xây dựng một ứng dụng có khả năng tương tác trực tiếp với người dùng, thông qua các nút lệnh điều khiển sinh vật ảo và cập nhật trạng thái theo thời gian thực.

2.3. Quản lý trạng thái đối tượng

Trong lập trình hướng đối tượng, trạng thái của đối tượng là tập hợp các giá trị thuộc tính tại một thời điểm cụ thể. Việc quản lý trạng thái hiệu quả đóng vai trò thiết yếu trong việc xây dựng các ứng dụng tương tác, đặc biệt là những chương trình mô phỏng, như ứng dụng Critter Caretaker.

Mỗi đối tượng (instance) được tạo ra từ một lớp (class) đều có các thuộc tính riêng biệt phản ánh trạng thái nội tại. Các thuộc tính này có thể thay đổi liên tục trong suốt vòng đời hoạt động của đối tượng. Ví dụ, một sinh vật ảo có thể có các trạng thái như

mức độ đói, sự nhàm chán hoặc năng lượng. Khi người dùng tương tác (cho ăn, cho chơi, cho ngủ...), trạng thái của đối tượng sẽ thay đổi tương ứng.

Python cho phép kiểm soát và cập nhật các trạng thái này thông qua phương thức (method) của lớp. Việc thiết kế tốt các phương thức như feed(), play() hay sleep() sẽ giúp đảm bảo rằng mọi thay đổi trạng thái được diễn ra đúng logic, an toàn và nhất quán.

Để đảm bảo quản lý trạng thái hiệu quả, cần chú ý đến một số nguyên tắc sau:

- Đóng gói (Encapsulation): Trạng thái nên được ẩn khỏi truy cập trực tiếp từ bên ngoài lớp. Việc thay đổi giá trị chỉ nên thực hiện thông qua các phương thức chuyên biệt, giúp tránh lỗi hoặc thay đổi ngoài ý muốn.
- Ràng buộc trạng thái: Các thuộc tính như mức độ đói hoặc chán cần được giới hạn trong một phạm vi hợp lý, ví dụ từ 0 đến 10, nhằm đảm bảo không xảy ra tình trạng bất hợp lệ như giá trị âm.
- Kiểm tra trạng thái trước khi thao tác: Hệ thống nên xác minh rằng đối tượng đã được khởi tạo trước khi cho phép người dùng tương tác. Việc này giúp xử lý ngoại lệ kịp thời và tạo ra trải nghiệm thân thiện hơn cho người dùng.
- Cập nhật giao diện real-time: Trong môi trường đồ họa sử dụng Tkinter, mỗi thay đổi trạng thái cần được cập nhật ngay trên màn hình. Có thể sử dụng biến liên kết như StringVar hoặc gọi lại các hàm cập nhật để đồng bộ hóa dữ liệu và hiển thị.

Ngoài ra, lập trình viên cần đảm bảo rằng các hành vi thay đổi trạng thái không xung đột nhau và được thực hiện theo thứ tự rõ ràng. Trong trường hợp nhiều hành động xảy ra liên tiếp, chương trình nên xác định ưu tiên cập nhật hoặc đưa ra các giới hạn để tránh rối loạn trạng thái.

Tóm lại, quản lý trạng thái đối tượng là trung tâm trong các ứng dụng hướng đối tượng có tính tương tác cao. Khi được triển khai đúng cách, nó không chỉ duy trì tính toàn vẹn của chương trình mà còn nâng cao trải nghiệm người dùng, đảm bảo rằng mọi thao tác đều có phản hồi trực quan và chính xác.

CHƯƠNG 3. THIẾT KẾ VÀ XÂY DỰNG CHƯƠNG TRÌNH

3.1. Sơ đồ khối hệ thống

Trong quá trình thiết kế và phát triển ứng dụng Critter Caretaker với giao diện đồ họa (GUI), việc xây dựng sơ đồ khối hệ thống giúp hình dung tổng thể các thành phần chính và mối quan hệ giữa chúng. Sơ đồ khối đóng vai trò là cầu nối giữa ý tưởng phần mềm và quá trình hiện thực hóa trong lập trình, từ đó hỗ trợ tổ chức mã nguồn theo hướng có cấu trúc và dễ bảo trì.

Hệ thống được chia thành ba khối chức năng chính:

(1) Giao diện người dùng (GUI): Đây là lớp trực tiếp tương tác với người dùng, được xây dựng bằng thư viện Tkinter. Giao diện bao gồm các thành phần chính như:

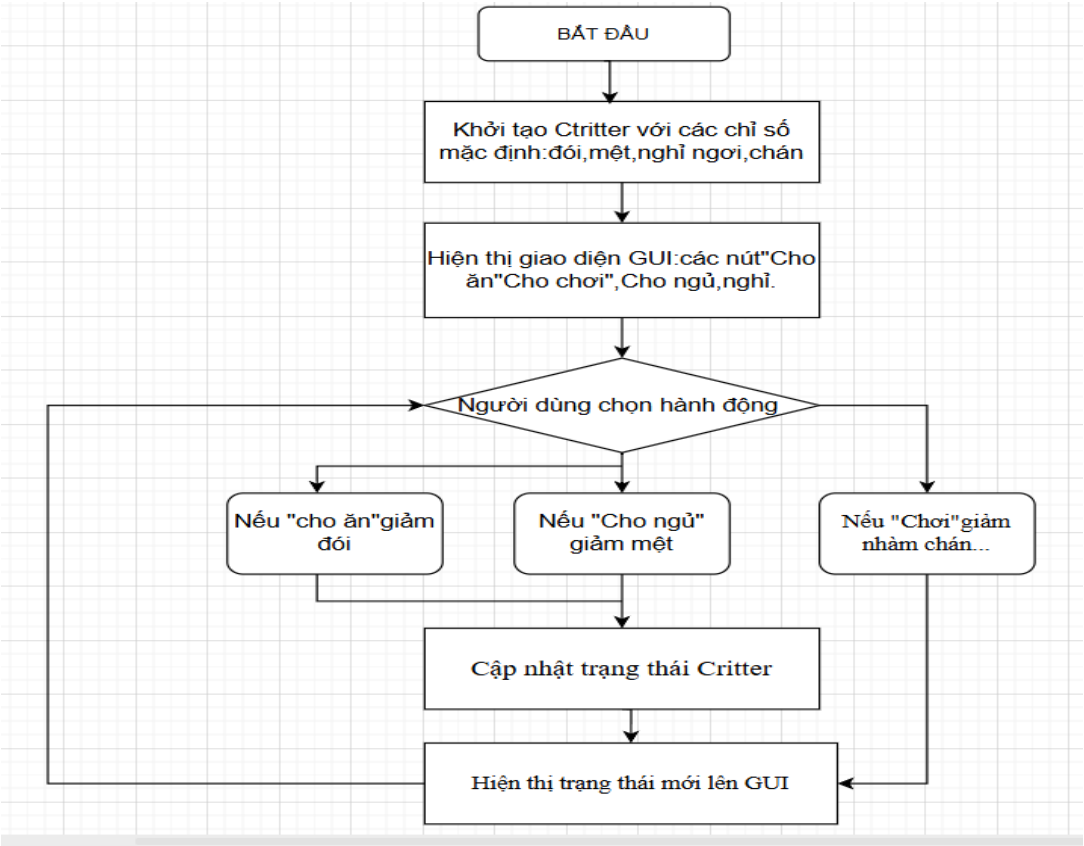
a) Ô nhập tên Critter (Entry)
b) Các nút chức năng: "Tạo", "Cho ăn", "Chơi", "Ngủ"
c) Nhãn hiển thị trạng thái của Critter (mức độ đói và chán) GUI không chỉ đóng vai trò hiển thị thông tin mà còn là điểm khởi phát của mọi sự kiện – các hành động người dùng sẽ được truyền xuống lớp xử lý logic phía sau.

(2) Lớp xử lý logic (Critter Class): Thành phần trung tâm của hệ thống là lớp Critter, đại diện cho sinh vật ảo. Lớp này chịu trách nhiệm lưu trữ trạng thái (các thuộc tính như hunger, boredom) và cung cấp các phương thức để cập nhật trạng thái (ăn, chơi, ngủ…). Critter đồng thời đóng vai trò mô hình hóa hành vi và đảm bảo quy tắc tương tác hợp lệ.

(3) Quản lý sự kiện và cập nhật trạng thái: Đây là lớp trung gian kết nối GUI và Critter class. Mỗi hành động của người dùng (nhấn nút) sẽ kích hoạt một hàm callback tương ứng, từ đó:

a) Kiểm tra điều kiện hợp lệ (đã tạo Critter chưa, tên có hợp lệ không…)
b) Gọi phương thức thích hợp từ lớp Critter
c) Cập nhật trạng thái trên giao diện

Mối quan hệ giữa các khối được mô tả như sau:



Hệ thống hoạt động theo mô hình phản hồi sự kiện: mọi tương tác của người dùng đều được ghi nhận qua GUI, chuyển đến lớp xử lý logic, thay đổi trạng thái của đối tượng, và kết quả được hiển thị ngược lại lên giao diện. Cơ chế cập nhật liên tục giúp tạo ra trải nghiệm người dùng liền mạch và trực quan.

Tổng thể, sơ đồ khối này giúp xác định rõ ràng vai trò của từng thành phần và đảm bảo rằng hệ thống được triển khai theo hướng tách biệt giữa giao diện, logic và dữ liệu – một nguyên tắc quan trọng trong thiết kế phần mềm.

3.2. Sơ đồ thuật toán chính

Thuật toán chính của chương trình Critter Caretaker được thiết kế theo mô hình xử lý sự kiện tuần tự, với sự kết hợp giữa giao diện đồ họa Tkinter và lớp đối tượng Critter. Dưới đây là mô tả sơ đồ thuật toán:

(1) Bắt đầu chương trình, hệ thống khởi tạo cửa sổ giao diện chính, bao gồm các thành phần: ô nhập tên (Entry), các nút lệnh (Buttons), và nhãn hiển thị trạng thái (Labels).

(2) Người dùng nhập tên sinh vật ảo (Critter) vào ô Entry.

(3) Khi nhấn nút "Tạo", chương trình tạo một đối tượng Critter với tên tương ứng. Nếu không nhập tên hoặc chưa nhấn nút tạo, các hành động tiếp theo sẽ bị chặn lại.

(4) Sau khi tạo Critter, hệ thống hiển thị trạng thái ban đầu (đói, chán) lên nhãn giao diện.

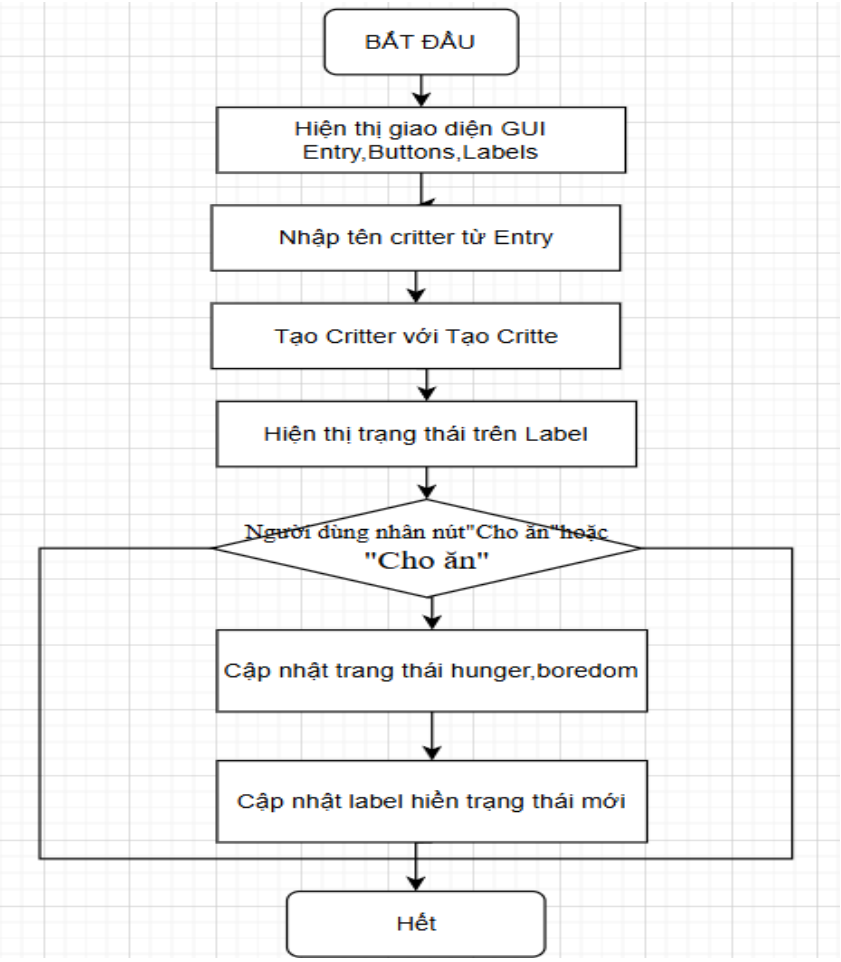
(5) Người dùng có thể tương tác với Critter bằng cách nhấn các nút như "Cho ăn", "Chơi" hoặc "Ngủ".

(6) Mỗi khi nút được nhấn, chương trình:

- Cập nhật trạng thái nội tại của Critter (hunger, boredom)
- Đồng thời cập nhật giao diện để hiển thị trạng thái mới

(7) Quá trình này được lặp lại liên tục trong vòng lặp sự kiện của giao diện cho đến khi chương trình kết thúc.

Hình sơ đồ thuật toán minh họa luồng xử lý như sau:



3.3. Cấu trúc dữ liệu

Trong chương trình Critter Caretaker, cấu trúc dữ liệu được thiết kế theo hướng lập trình hướng đối tượng, kết hợp với các widget giao diện đồ họa của thư viện Tkinter. Dữ liệu không được lưu trữ trong cơ sở dữ liệu dạng bảng như SQL, mà được quản lý động thông qua các lớp, biến thuộc tính và liên kết logic giữa giao diện và lớp đối tượng.

3.3.1. Lớp đối tượng Critter

Trung tâm của cấu trúc dữ liệu là lớp Critter, đại diện cho sinh vật ảo do người dùng tạo ra. Lớp này định nghĩa các thuộc tính nội tại và hành vi của Critter.

- Các thuộc tính chính:
  - name: Tên của Critter, được nhập từ người dùng.
  - hunger: Mức độ đói, là một số nguyên từ 0–10. Càng cao nghĩa là càng đói.
  - boredom: Mức độ chán, cũng là số nguyên từ 0–10.
  - image_file: Đường dẫn đến hình ảnh đại diện cho Critter.
- Các phương thức:
  - feed(): Giảm mức đói.
  - play(): Giảm mức chán.
  - sleep(): Làm Critter nghỉ ngơi, nhưng đồng thời tăng hunger và boredom (mô phỏng thời gian trôi qua).
  - get_status(): Trả về chuỗi mô tả trạng thái hiện tại, được sử dụng để hiển thị trên giao diện.

Thông tin trạng thái được quản lý trực tiếp bên trong đối tượng Critter, giúp dễ dàng kiểm soát và cập nhật liên tục mà không cần đến cơ sở dữ liệu bên ngoài.

3.3.2. Biến liên kết và dữ liệu giao diện

Ngoài dữ liệu bên trong Critter, lớp CritterApp sử dụng nhiều thành phần Tkinter để thu thập và hiển thị thông tin:

- self.entry_name: Widget Entry dùng để nhập tên Critter.

- self.critter_type: Biến StringVar dùng để chọn loại Critter từ menu.
- self.status_label: Label hiển thị trạng thái hunger và boredom.
- self.critter: Biến tạm lưu trữ thể hiện hiện tại của lớp Critter.
- self.critter_img: Dữ liệu ảnh của Critter đang được hiển thị trên giao diện.

Dữ liệu giao diện được đồng bộ với dữ liệu logic bằng cách gọi các hàm như update_status() sau mỗi thao tác hoặc sau mỗi chu kỳ thời gian.

3.3.3. Cập nhật dữ liệu tự động

Một cơ chế quan trọng của chương trình là cập nhật trạng thái Critter theo thời gian thực. Hàm auto_update_state() được gọi định kỳ 10 giây/lần bằng hàm after() của Tkinter. Mỗi lần gọi, hunger và boredom của Critter sẽ tăng lên một đơn vị, mô phỏng hiệu ứng thời gian trôi qua mà không chăm sóc. Nhờ đó, trạng thái của Critter luôn thay đổi linh hoạt, giống như một thú cưng ảo thực thụ.

Tóm tắt cấu trúc dữ liệu:

Thành phần	Kiểu dữ liệu	Mục đích
name	str	Tên Critter do người dùng nhập
hunger	int (0–10)	Mức đói của Critter
boredom	int (0–10)	Mức chán của Critter
image_file	str	Đường dẫn đến hình ảnh tương ứng
self.critter	Critter	Đối tượng Critter hiện tại
self.status_label	Label	Hiển thị trạng thái hunger và boredom

Nhìn chung, cấu trúc dữ liệu của chương trình được xây dựng đơn giản nhưng hiệu quả, tận dụng tối đa sức mạnh của lập trình hướng đối tượng và khả năng liên kết dữ liệu của giao diện Tkinter. Điều này giúp hệ thống hoạt động nhẹ, phản hồi tức thời, và có thể mở rộng dễ dàng trong các phiên bản tiếp theo.

3.4. Các hàm chính trong chương trình

Chương trình Critter Caretaker được tổ chức thành hai lớp chính: lớp logic Critter và lớp giao diện CritterApp. Mỗi lớp chứa các hàm (phương thức) riêng phục vụ nhiệm vụ quản lý trạng thái sinh vật ảo, xử lý sự kiện người dùng và cập nhật giao diện. Dưới đây là mô tả các hàm chính của chương trình:

3.4.1. Các hàm trong lớp Critter

Lớp này chứa các hành vi cơ bản của sinh vật ảo, mỗi hàm tương ứng với một thao tác hoặc trạng thái nội tại.

- __init__(self, name, image_file): Phương thức khởi tạo một đối tượng Critter mới. Thiết lập tên, hình ảnh, mức độ đói (hunger) và chán (boredom) ban đầu.
- feed(self): Hàm xử lý hành động "cho ăn", giảm giá trị hunger đi 1 đơn vị, tối thiểu là 0.
- play(self): Hàm xử lý hành động "chơi", giảm boredom đi 1 đơn vị, không thấp hơn 0.
- sleep(self): Hàm xử lý hành động "ngủ", mô phỏng quá trình hồi phục nhưng kèm hệ quả: tăng đói và chán thêm 1 đơn vị mỗi cái.
- get_status(self): Trả về chuỗi mô tả trạng thái hiện tại của Critter, gồm tên, độ đói và độ chán. Chuỗi này được hiển thị trên giao diện chính.

3.4.2. Các hàm trong lớp CritterApp

Đây là lớp điều khiển toàn bộ giao diện Tkinter và kết nối giữa người dùng với logic của Critter.

- __init__(self, master): Hàm khởi tạo giao diện chính. Thiết lập cửa sổ, hình nền, các thành phần giao diện (nút, ô nhập liệu, vùng ảnh, nhãn trạng thái). Đồng thời khởi chạy nhạc nền và gọi hàm tự động cập nhật trạng thái định kỳ.
- create_critter(self): Được gọi khi người dùng nhấn nút "Tạo". Lấy tên và loại Critter từ giao diện, khởi tạo một đối tượng Critter mới, hiển thị hình ảnh và cập nhật trạng thái ban đầu.
- load_critter_image(self, image_path): Đọc file ảnh của Critter, chuyển thành ImageTk.PhotoImage và hiển thị lên vùng Canvas.

- update_status(self): Cập nhật nhãn trạng thái dựa trên trạng thái mới của Critter. Nếu hunger hoặc boredom vượt quá ngưỡng (>= 8), chữ sẽ chuyển sang màu đỏ để cảnh báo.
- auto_update_state(self): Hàm quan trọng dùng để cập nhật tự động trạng thái Critter sau mỗi khoảng thời gian. Cứ sau 10 giây, chương trình sẽ tăng mức hunger và boredom thêm 1 đơn vị (tối đa 10). Hàm này dùng phương pháp after() của Tkinter để lặp vô hạn.
- feed_critter(self): Gọi phương thức feed() của Critter, phát âm thanh tương ứng và cập nhật giao diện.
- play_critter(self): Gọi phương thức play() của Critter, phát âm thanh tương ứng và cập nhật trạng thái.
- sleep_critter(self): Gọi phương thức sleep() của Critter, phát âm thanh tương ứng và hiển thị kết quả thay đổi lên màn hình.

3.4.3. Các hàm hỗ trợ toàn cục

Dưới đây là các hàm hỗ trợ toàn cục của chương trình:

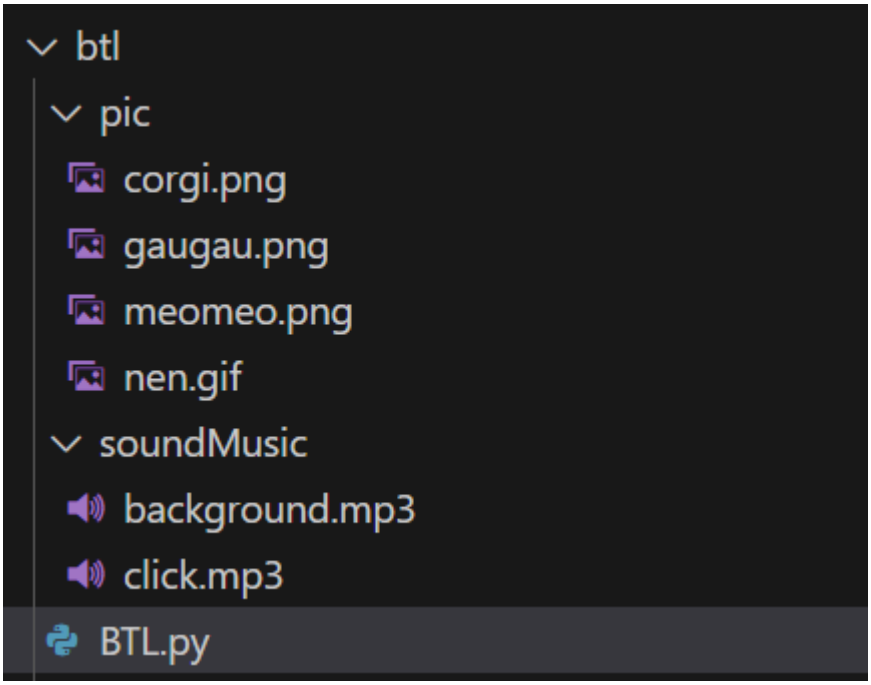
- play_background_music():
  Phát nhạc nền (background.mp3) ở chế độ lặp vô hạn thông qua thư viện pygame.
- play_sound(file_name):
  Phát hiệu ứng âm thanh khi người dùng tương tác (ví dụ: click, ăn, chơi...).

Tổng thể, các hàm trong chương trình được tổ chức rõ ràng, phân tách hợp lý giữa xử lý logic và xử lý giao diện. Việc kết hợp vòng lặp sự kiện Tkinter (after()) với logic cập nhật trạng thái giúp mô phỏng sinh vật sống một cách hiệu quả, tạo cảm giác phản hồi liên tục và tự nhiên cho người dùng.

3.5. Cài đặt chương trình

3.5.1. Cấu trúc thư mục

Chương trình Critter Caretaker được tổ chức với cấu trúc thư mục rõ ràng, đảm bảo dễ triển khai, dễ quản lý và thuận tiện khi mở rộng. Toàn bộ mã nguồn và tài nguyên được lưu trữ trong thư mục chính tên là btl, bao gồm các thành phần như sau:



Giải thích các thành phần:

- BTL.py: File mã nguồn chính của chương trình, chứa toàn bộ logic xử lý và giao diện.
- pic/: Thư mục chứa hình ảnh sử dụng trong chương trình.
  - corgi.png, gaugau.png, meomeo.png: Ảnh đại diện của các loại Critter.
  - nen.gif: Hình nền cho giao diện chính của ứng dụng.
- soundMusic/: Thư mục chứa các tệp âm thanh sử dụng trong chương trình.
  - background.mp3: Nhạc nền được phát liên tục khi ứng dụng chạy.
  - click.mp3: Hiệu ứng âm thanh khi người dùng thao tác với các nút chức năng.

Thư mục được tổ chức tách biệt giữa mã nguồn, hình ảnh và âm thanh nhằm tăng tính mô-đun, giúp dễ dàng thay đổi hoặc mở rộng chương trình trong tương lai.

3.5.2. Code chương trình

```
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk
```

```
import pygame
import os
# Khởi động hệ thống âm thanh
pygame.mixer.init()
# Đường dẫn thư mục
PATH_IMG = os.path.join("btl", "pic")
PATH_SOUND = os.path.join("btl", "soundMusic")
# Lớp Critter
class Critter:
    def __init__(self, name, image_file):
        self.name = name
        self.hunger = 5
        self.boredom = 5
        self.image_file = image_file

    def feed(self):
        self.hunger = max(0, self.hunger - 1)

    def play(self):
        self.boredom = max(0, self.boredom - 1)

    def sleep(self):
        self.hunger = min(10, self.hunger + 1)
        self.boredom = min(10, self.boredom + 1)

    def get_status(self):
        return f"{self.name} - Đói: {self.hunger} | Chán:
{self.boredom}"
```

```
# Hàm phát nhạc nền
def play_background_music():
    bg_path = os.path.join(PATH_SOUND, "background.mp3")
    pygame.mixer.music.load(bg_path)
    pygame.mixer.music.play(-1)  # Lặp vô hạn
def play_sound(file_name):
    sound_path = os.path.join(PATH_SOUND, file_name)
    sound = pygame.mixer.Sound(sound_path)
    sound.play()
# Giao diện
class CritterApp:
    def __init__(self, master):
        self.master = master
        master.title("Critter Caretaker")
        master.geometry("740x450")
        master.resizable(False, False)
        self.critter = None
        self.critter_img = None
        # Hình nền
        bg_image = Image.open(os.path.join(PATH_IMG,
"nen.gif")).resize((740, 450))
        self.bg = ImageTk.PhotoImage(bg_image)
        bg_label = tk.Label(master, image=self.bg)
        bg_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Nhập tên
        self.entry_name = tk.Entry(master, font=("Arial", 14),
width=20, justify="center")
        self.entry_name.place(x=150, y=20)
```

```
        # Dropdown chọn loại
        self.critter_type = tk.StringVar(value="corgi")
        self.dropdown = tk.OptionMenu(master, self.critter_type,
"corgi", "gaugau", "meomeo")
        self.dropdown.config(font=("Arial", 12))
        self.dropdown.place(x=380, y=18)
        # Nút tạo Critter
   self.btn_create = tk.Button(master, text="Tạo",
command=self.create_critter, bg="#28a745", fg="white", width=10,
font=("Arial", 12))
        self.btn_create.place(x=250, y=60)
        # Vùng hiển thị hình
        self.canvas = tk.Canvas(master, width=200, height=200, bd=0,
highlightthickness=0, bg="white")
        self.canvas.place(x=200, y=100)
        # Trạng thái
        self.status_label = tk.Label(master, text="", font=("Arial",
14), bg="black", fg="white")
        self.status_label.place(x=200, y=320)
        # Các nút hành động
   self.btn_feed = tk.Button(master, text="Cho ăn",
command=self.feed_critter, bg="#007bff", fg="white", width=12,
font=("Arial", 12))
        self.btn_feed.place(x=70, y=400)
   self.btn_play = tk.Button(master, text="Chơi",
command=self.play_critter, bg="#ffc107", fg="black", width=12,
font=("Arial", 12))
        self.btn_play.place(x=230, y=400)
   self.btn_sleep = tk.Button(master, text="Ngủ",
command=self.sleep_critter, bg="#6c757d", fg="white", width=12,
font=("Arial", 12))
```

```
        self.btn_sleep.place(x=390, y=400)
        # Phát nhạc nền
        play_background_music()
        # Bắt đầu cập nhật trạng thái tự động
        self.master.after(10000, self.auto_update_state)
    def create_critter(self):
        name = self.entry_name.get().strip()
        critter_type = self.critter_type.get()
        if not name:
            messagebox.showwarning("Lỗi", "Vui lòng nhập tên
Critter.")
            return
        image_path = os.path.join(PATH_IMG, f"{critter_type}.png")
        self.critter = Critter(name, image_path)
        self.load_critter_image(image_path)
        self.update_status()
    def load_critter_image(self, image_path):
        image = Image.open(image_path).resize((180, 180))
        self.critter_img = ImageTk.PhotoImage(image)
        self.canvas.delete("all")
        self.canvas.create_image(100, 100, image=self.critter_img)
    def update_status(self):
        if self.critter:
            status = self.critter.get_status()
            self.status_label.config(text=status)
            # Thay đổi màu cảnh báo nếu trạng thái xấu
            if self.critter.hunger >= 8 or self.critter.boredom >= 8:
                self.status_label.config(fg="red")
            else:
```

```
                self.status_label.config(fg="white")
    def auto_update_state(self):
        if self.critter:
            # Tăng hunger và boredom mỗi 10s
            self.critter.hunger = min(10, self.critter.hunger + 1)
            self.critter.boredom = min(10, self.critter.boredom + 1)
            self.update_status()
        # Lặp lại sau 10s
        self.master.after(10000, self.auto_update_state)
    def feed_critter(self):
        if self.critter:
            self.critter.feed()
            self.update_status()
            play_sound("click.mp3")
        else:
            messagebox.showinfo("Thông báo", "Hãy tạo Critter
trước.")
    def play_critter(self):
        if self.critter:
            self.critter.play()
            self.update_status()
            play_sound("click.mp3")
        else:
            messagebox.showinfo("Thông báo", "Hãy tạo Critter
trước.")
    def sleep_critter(self):
        if self.critter:
            self.critter.sleep()
            self.update_status()
```

```
            play_sound("click.mp3")
        else:
            messagebox.showinfo("Thông báo", "Hãy tạo Critter
trước.")


# Chạy chương trình
if __name__ == "__main__":
    root = tk.Tk()
    app = CritterApp(root)
    root.mainloop()
```

CHƯƠNG 4. THỰC NGHIỆM VA KẾT LUẬN

4.1. Kết quả thực nghiệm

Sau khi hoàn thiện chương trình Critter Caretaker, hệ thống đã được kiểm thử trên môi trường Python 3.12 với thư viện hỗ trợ là Tkinter, Pillow và Pygame. Quá trình thực nghiệm được tiến hành trực tiếp trên máy tính cá nhân với giao diện người dùng đầy đủ và các thao tác tương tác chính.

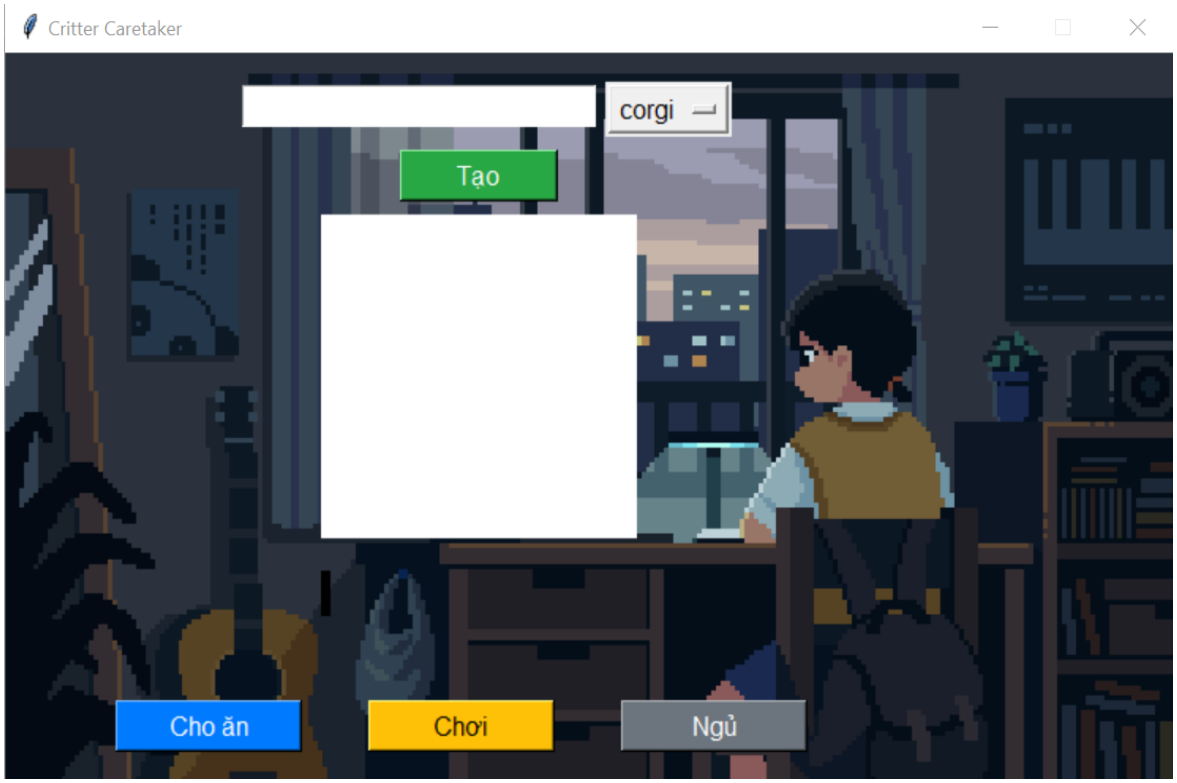
Kết quả thu được cho thấy:

- Giao diện hiển thị ổn định, có hình nền động, ảnh sinh vật rõ nét và các nút chức năng được bố trí hợp lý.
- Người dùng có thể tạo Critter mới bằng cách nhập tên và chọn loại sinh vật, sau đó thực hiện các thao tác tương tác như “Cho ăn”, “Chơi”, “Ngủ”.
- Hệ thống âm thanh hoạt động tốt, bao gồm nhạc nền và các hiệu ứng riêng biệt cho từng hành động, giúp tăng tính sống động và hấp dẫn của ứng dụng.
- Cơ chế cập nhật tự động mỗi 10 giây hoạt động đúng như mong đợi: nếu người dùng không chăm sóc, trạng thái đói và chán của Critter sẽ tăng dần theo thời gian.
- Thông tin trạng thái được hiển thị rõ ràng, có màu sắc cảnh báo khi Critter bị đói hoặc chán ở mức cao, giúp người dùng dễ dàng nhận biết và phản hồi.
- Không phát sinh lỗi trong quá trình thao tác, chương trình xử lý tốt các tình huống như không nhập tên, chưa tạo Critter mà đã nhấn nút hành động,...

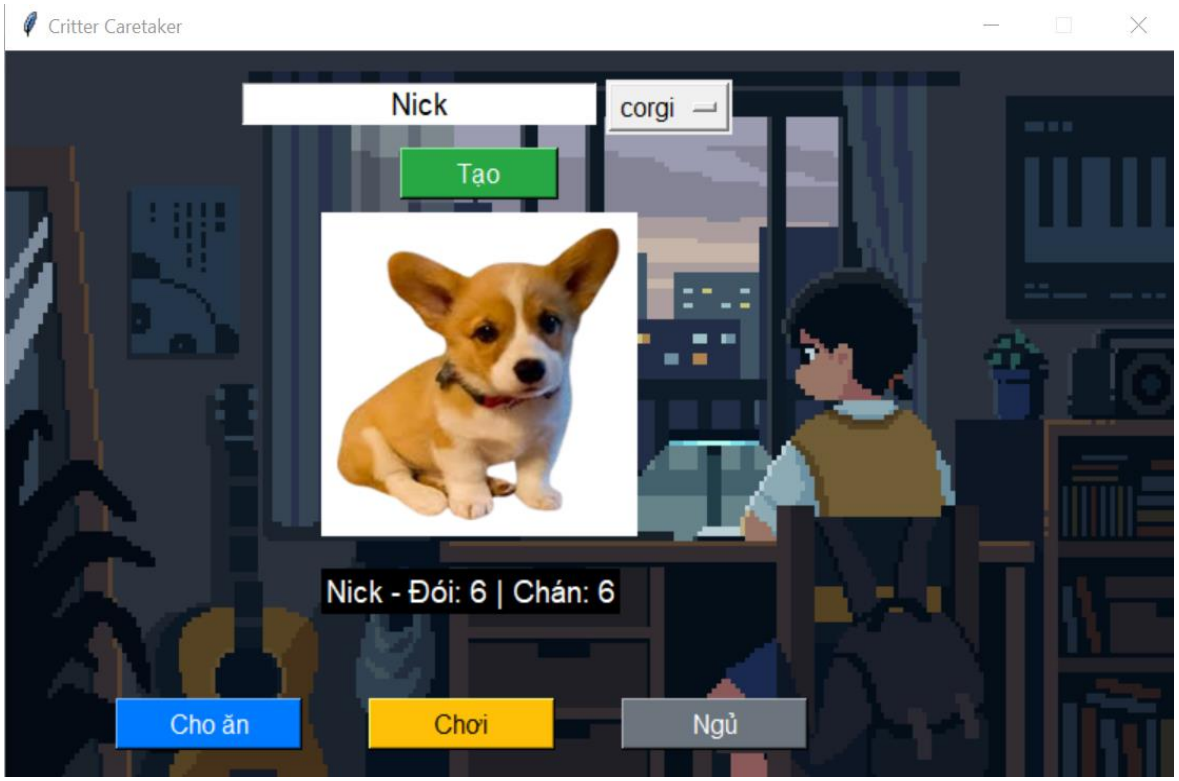
Các hình ảnh dưới đây ghi lại giao diện thực tế của chương trình khi vận hành. Người dùng có thể nhập tên, chọn loại Critter và thực hiện các thao tác chăm sóc thông qua các nút chức năng. Giao diện hiển thị hình ảnh sinh vật, trạng thái đói/chán và phản hồi âm thanh tương ứng. Màu sắc trạng thái sẽ thay đổi khi Critter bị đói hoặc chán quá mức. Đây là minh chứng trực quan cho khả năng tương tác và mô phỏng sinh vật ảo của hệ thống.

Ảnh minh họa thực nghiệm:

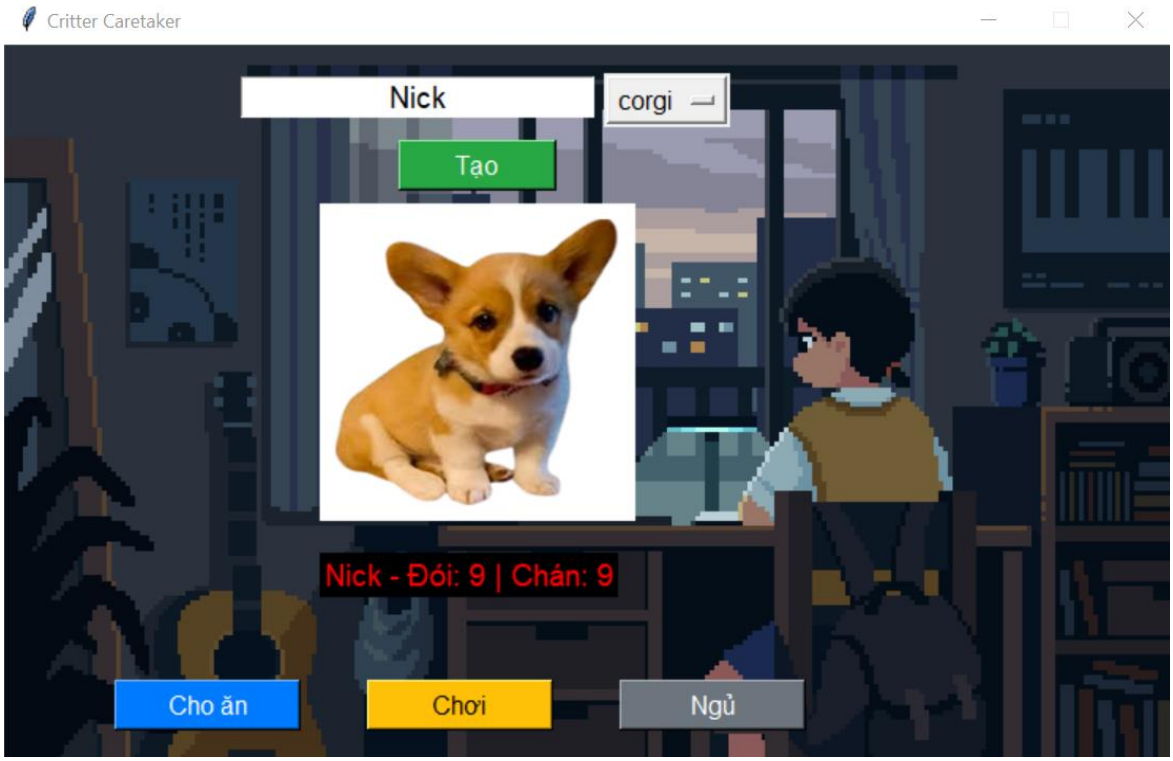
- Giao diện khi vừa khởi động chương trình



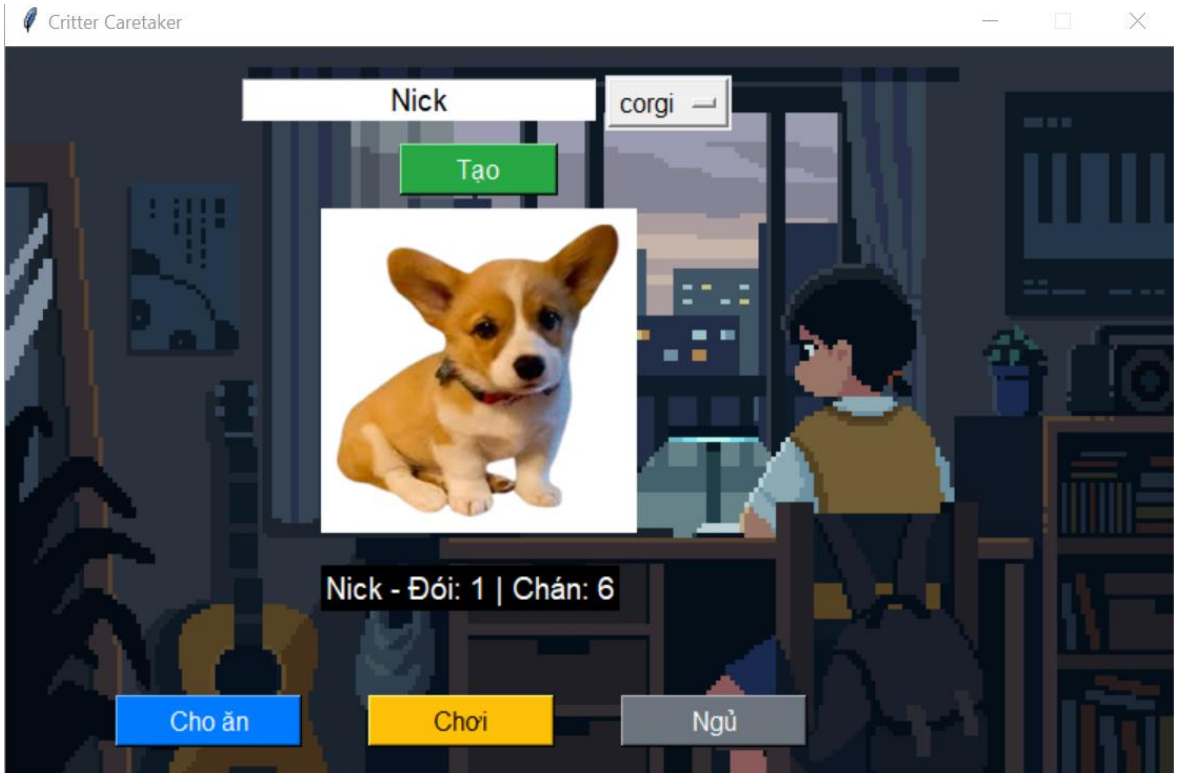
- Giao diện sau khi tạo Critter



- Giao diện khi Critter bị đói quá mức (cảnh báo màu đỏ)



- Trạng thái thay đổi khi nhấn "Cho ăn" hoặc "Chơi"



4.2. Kết luận và hướng phát triển

Đề tài Critter Caretaker đã mô phỏng thành công một hệ thống chăm sóc sinh vật ảo đơn giản nhưng sinh động. Thông qua sự kết hợp giữa lập trình hướng đối tượng, thư viện Tkinter cho giao diện, và Pygame cho xử lý âm thanh, chương trình đạt được các mục tiêu cơ bản:

- Cho phép người dùng tạo, tương tác và theo dõi trạng thái Critter
- Mô phỏng trạng thái sinh học qua hunger và boredom
- Phản hồi theo thời gian và thao tác thực tế từ người dùng

Việc áp dụng kỹ thuật cập nhật trạng thái tự động, cùng khả năng phản hồi tức thì qua giao diện và âm thanh, đã giúp hệ thống trở nên giống một sinh vật ảo "thật" hơn, tăng tính hấp dẫn và tương tác.

Trong tương lai, hệ thống có thể được mở rộng theo nhiều hướng:

- Đa Critter: Cho phép người dùng chăm sóc nhiều sinh vật cùng lúc, chuyển đổi giữa chúng dễ dàng.
- Trạng thái nâng cao: Bổ sung thêm các yếu tố như sức khỏe, năng lượng, cảm xúc, hoặc bệnh tật.
- Biểu cảm động: Thay đổi hình ảnh Critter theo trạng thái (vui, buồn, mệt mỏi…).
- Lưu dữ liệu: Tích hợp chức năng lưu/đọc trạng thái Critter từ file hoặc cơ sở dữ liệu.
- Ứng dụng di động: Chuyển đổi giao diện sang dạng touch-screen hoặc phát triển trên nền tảng như Kivy để hỗ trợ Android.

TÀI LIỆU THAM KHẢO

[1] Real Python. *Object-Oriented Programming (OOP) in Python 3*. Truy cập tại: https://realpython.com/python3-object-oriented-programming/

[2] Python Software Foundation. *The Python Standard Library – Tkinter*. Truy cập tại: https://docs.python.org/3/library/tkinter.html

[3] Pygame Community. *Pygame Documentation*. Truy cập tại: https://www.pygame.org/docs/

[4] Nguyễn Văn Linh. *Giáo trình Lập trình Python cơ bản*. NXB Đại học Quốc gia Hà Nội, 2022.

[5] Khoa Điện tử – Truyền thông, Đại học Kỹ thuật Công nghiệp – ĐH Thái Nguyên. *Tài liệu hướng dẫn thực hành Lập trình Python*. Nội bộ, 2024.

[6] Stack Overflow. *Various discussions and solutions related to Tkinter and Pygame*. Truy cập tại: https://stackoverflow.com