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

ĐỀ TÀI: XÂY GAME ASTROCRASH (CHAPTER 12) VỚI PYGAME: ĐIỀU KHIỂN TÀU, BẮN ASTEROID, ÂM THANH

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

ĐỀ TÀI: XÂY GAME ASTROCRASH (CHAPTER 12) VỚI PYGAME: ĐIỀU KHIỂN TÀU, BẮN ASTEROID, ÂM THANH

GIÁO VIÊN HƯỚNG DẪN	**:**	**TS.Nguyễn Văn Huy**
HỌ VÀ TÊN SINH VIÊN	**:**	**Nguyễn Đình Tú**
MSSV	**:**	**K225480106067**
LỚP		**K58.KTP**

Thái Nguyên 2025

TRƯỜNG ĐHKTCN
KHOA ĐIỆN TỬ

CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc lập - Tự do - Hạnh phúc

BÀI TẬP LỚN

MÔN HỌC : LẬP TRÌNH PYTHON

Họ tên sinh viên : Nguyễn Đình Tú MSV : k225480106067

Lớp : K58.KTP Ngành : Kỹ thuật máy tính

Giáo viên hướng dẫn: TS.Nguyễn Văn Huy

Đề bài : Xây game Astrocrash (Chapter 12) với pygame: điều khiển tàu, bắn asteroid, âm thanh

Đầu vào – đầu ra:

- Đầu vào: Phím mũi tên quay/move, phím cách bắn.
- Đầu ra: Điểm, hiệu ứng nổ, nhạc nền.

Tính năng yêu cầu:

- Quay sprite, di chuyển, bắn tên lửa (Missile).
- Collisions, di chuyển asteroid.
- Phát sound và music.

Kiểm tra & kết quả mẫu:

- Bắn trúng asteroid → nổ +10 điểm, có sound “boom”.

Các bước triển khai:

1. Tham khảo Astrocrash01→08.
2. Setup game classes: Ship, Asteroid, Missile, Explosion.
3. Xử lý keyboard, math cho di chuyển.
4. Tải và phát sound, music.

Ngày giao đề tài: Ngày hoàn thành:

GIÁO VIÊN HƯỚNG DẪN

(Ký ghi rõ họ tên)

Nguyễn Văn Huy

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

Thái Nguyên, ngày….tháng…..năm…..

GIÁO VIÊN HƯỚNG DẪN

(Ký ghi rõ họ tên)

LỜI NÓI ĐẦU

Trong thời đại công nghệ phát triển mạnh mẽ, lập trình trò chơi không chỉ là một lĩnh vực giải trí mà còn là môi trường học tập hiệu quả giúp người học rèn luyện tư duy logic, khả năng thiết kế phần mềm và áp dụng các kiến thức về cấu trúc dữ liệu, thuật toán, và lập trình hướng đối tượng.

Bài tập lớn *Astrocrash* được thực hiện nhằm mục đích áp dụng những kiến thức đã học về ngôn ngữ lập trình Python và thư viện Pygame để xây dựng một trò chơi 2D đơn giản. Thông qua việc thiết kế và lập trình trò chơi, em thực hiện đã có cơ hội củng cố lại kiến thức nền tảng, hiểu rõ hơn về quy trình phát triển phần mềm, cũng như kỹ năng giải quyết vấn đề thực tiễn.

Báo cáo này trình bày các bước thực hiện dự án từ việc tìm hiểu lý thuyết, thiết kế hệ thống cho đến lập trình, thử nghiệm và đánh giá kết quả. Mặc dù đã rất cố gắng trong quá trình thực hiện, nhưng không tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý từ quý thầy cô và bạn đọc để đồ án được hoàn thiện hơn.

Em xin chân thành cảm ơn sự hướng dẫn tận tình của quý thầy cô đã tạo điều kiện và hỗ trợ em trong suốt quá trình thực hiện đề tài.

LỜI CẢM ƠN

Trong suốt quá trình học tập và thực hiện bài tập lớn, em đã nhận được sự giúp đỡ tận tình của các thầy cô trong bộ môn Tin học Công nghiệp – Khoa Điện tử – Trường Đại học Kỹ thuật Công Nghiệp – Đại học Thái Nguyên, đặc biệt là Thầy Huy. Em xin bày tỏ lòng biết ơn sâu sắc tới thầy đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình thực hiện đề tài này.

Mặc dù đã cố gắng hoàn thành tốt nhất trong khả năng của mình, nhưng do hạn chế về thời gian và kinh nghiệm thực tế, bài làm không tránh khỏi những thiếu sót. Em rất mong nhận được sự góp ý của thầy và các bạn để hoàn thiện hơn.

Em xin chân thành cảm ơn!

Sinh viên thực hiện

MỤC LỤC

DANH MỤC VIẾT TẮT

2D: Two-Dimensional

OOP: Object-Oriented Programming

FPS: Frames Per Second

MP3: MPEG Audio Layer III

WAV: Waveform Audio File Format

UI: User Interface

DANH MỤC HÌNH ẢNH

CHƯƠNG I : GIỚI THIỆU TỔNG QUAN VỀ ĐỀ TÀI

1.1. Lý do chọn đề tài

Trong bối cảnh kỷ nguyên công nghệ số bùng nổ mạnh mẽ, ngành công nghiệp trò chơi điện tử đã không ngừng mở rộng và nhanh chóng trở thành một trong những lĩnh vực giải trí phổ biến nhất trên toàn cầu. Sự phát triển vượt bậc này không chỉ mang lại giá trị giải trí đơn thuần mà còn là một môi trường lý tưởng để người học rèn luyện và phát triển nhiều kỹ năng cốt lõi. Cụ thể, thông qua quá trình xây dựng game, người học có thể nâng cao năng lực **lập trình**, cải thiện **tư duy logic** trong việc giải quyết vấn đề, trau dồi kỹ năng **xử lý đồ họa** và tối ưu hóa **tương tác người dùng**. Nhận thấy tiềm năng giáo dục to lớn đó, đề tài **Astrocrash – Âm thanh & Chuyển động** đã được lựa chọn.

Mục tiêu chính của đề tài này là giúp người học tiếp cận một cách trực quan và thực tế với quy trình phát triển một game 2D cơ bản. Việc sử dụng ngôn ngữ lập trình **Python** kết hợp với thư viện **Pygame** cung cấp một nền tảng vững chắc để biến ý tưởng thành sản phẩm. Thông qua việc tự tay xây dựng trò chơi này, người thực hiện có cơ hội áp dụng và củng cố các kiến thức quan trọng đã học. Điều này bao gồm việc xử lý linh hoạt các **sự kiện bàn phím** để điều khiển nhân vật, lập trình chuyển động và **điều khiển đối tượng** trong không gian game, phát hiện và xử lý chính xác các tình huống **va chạm** giữa các vật thể. Hơn nữa, việc tính toán và **cập nhật điểm số**, cùng với khả năng **tích hợp âm thanh và nhạc nền** một cách hợp lý vào game, sẽ giúp người học tạo ra một trải nghiệm chơi game hoàn chỉnh và sống động. Đề tài không chỉ là một bài tập kỹ thuật mà còn là cơ hội để khám phá sự sáng tạo trong việc kết hợp các yếu tố lập trình và thiết kế âm thanh, hình ảnh.

1.2. Tổng quan về chương trình

Chương trình **Astrocrash** là một trò chơi bắn thiên thạch 2D kinh điển, được xây dựng sử dụng thư viện Pygame trong Python. Mục tiêu của trò chơi là điều khiển một con tàu vũ trụ để tiêu diệt các thiên thạch (asteroid) trôi nổi trong không gian.

1.2.1. Đầu vào và Đầu ra của chương trình

- **Đầu vào**: Người chơi tương tác với game thông qua **bàn phím**. Các phím mũi tên được sử dụng để điều khiển việc **quay** và **di chuyển** tàu vũ trụ, trong khi phím cách (spacebar) dùng để **bắn tên lửa**.
- **Đầu ra**: Chương trình cung cấp phản hồi cho người chơi dưới nhiều dạng. Điểm số sẽ được hiển thị trên màn hình, tăng lên mỗi khi thiên thạch bị tiêu diệt. Các **hiệu ứng nổ** đồ họa sẽ xuất hiện khi có va chạm hoặc thiên thạch bị phá hủy. Đặc biệt, trò chơi sẽ có các **hiệu ứng âm thanh** cho các sự kiện (ví dụ: tiếng "boom" khi bắn trúng asteroid) và một **bản nhạc nền** xuyên suốt để tạo không khí.

1.2.2. Các tính năng chính của chương trình

Chương trình được thiết kế với các tính năng chủ đạo sau:

- **Điều khiển và di chuyển đối tượng**: Người chơi có thể **quay** tàu vũ trụ của mình theo các hướng khác nhau và **di chuyển** tiến về phía trước trong không gian. Tàu cũng có khả năng **bắn tên lửa (Missile)** về phía các thiên thạch.
- **Xử lý va chạm và chuyển động**: Các **thiên thạch** trong game sẽ liên tục di chuyển trên màn hình với các hướng và tốc độ ngẫu nhiên. Hệ thống sẽ **phát hiện và xử lý va chạm** giữa tên lửa và thiên thạch, cũng như va chạm giữa tàu của người chơi và thiên thạch.
- **Tích hợp âm thanh và nhạc nền**: Để tăng cường trải nghiệm sống động, chương trình tích hợp các **hiệu ứng âm thanh** cho các hành động quan trọng như tiếng bắn tên lửa, tiếng nổ khi va chạm. Ngoài ra, một bản **nhạc nền** sẽ được phát liên tục xuyên suốt quá trình chơi, góp phần tạo không khí và thu hút người chơi.

1.2.3. Các thách thức trong quá trình thực hiện

Trong quá trình phát triển chương trình, một số thách thức đáng kể đã được đặt ra:

- **Quản lý trạng thái và tương tác đối tượng**: Thách thức đầu tiên là làm sao để quản lý hiệu quả trạng thái của nhiều đối tượng khác nhau (tàu, thiên thạch, tên lửa, vụ nổ) đồng thời và đảm bảo chúng tương tác mượt mà với nhau. Điều này đòi hỏi việc thiết kế lớp (class) cho từng đối tượng một cách hợp lý và quản lý vòng đời của chúng (tạo mới, cập nhật vị trí, xóa bỏ khi không còn cần thiết) trong game loop.
- **Xử lý va chạm chính xác và hiệu quả**: Việc phát hiện va chạm giữa các hình ảnh không đồng nhất (ví dụ: tàu hình tam giác, thiên thạch hình khối bất định) đòi hỏi việc sử dụng các kỹ thuật như **kiểm tra va chạm pixel hoàn hảo** (pixel-perfect collision) hoặc ít nhất là va chạm dựa trên mặt nạ (mask-based collision) để đảm bảo độ chính xác, tránh các trường hợp va chạm giả hoặc bỏ sót. Tối ưu hóa việc này để không ảnh hưởng đến hiệu suất game cũng là một điểm cần lưu ý.
- **Tích hợp âm thanh đồng bộ**: Việc phát các hiệu ứng âm thanh và nhạc nền một cách đồng bộ với các sự kiện trong game (ví dụ: tiếng bắn phải phát ngay khi nhấn phím, tiếng nổ phải khớp với hiệu ứng hình ảnh) đòi hỏi sự hiểu biết về cách pygame.mixer hoạt động và quản lý các kênh âm thanh để tránh bị chồng chéo hoặc độ trễ.

1.2.4. Các kiến thức đã vận dụng để thực hiện đề tài

Để hoàn thành đề tài này, người thực hiện đã vận dụng một cách tổng hợp nhiều kiến thức quan trọng từ môn học và các lĩnh vực liên quan:

- **Lập trình hướng đối tượng (OOP)**: Đây là kiến thức cốt lõi được áp dụng xuyên suốt để thiết kế cấu trúc chương trình. Mỗi thực thể trong game như **Ship**, **Asteroid**, **Missile**, **Explosion** đều được mô hình hóa thành các **lớp (class)** riêng biệt. Việc sử

dụng các thuộc tính (variables) và phương thức (methods) trong các lớp này giúp quản lý code gọn gàng, dễ bảo trì và mở rộng.

- **Thao tác với thư viện Pygame**: Người thực hiện đã nắm vững và vận dụng các chức năng chính của thư viện Pygame, bao gồm:
    - **Quản lý cửa sổ và hiển thị đồ họa**: Sử dụng pygame.display để tạo và quản lý cửa sổ game, pygame.image để tải và biến đổi hình ảnh (bao gồm cả việc xoay sprite theo hướng tàu), và pygame.Surface để vẽ các đối tượng lên màn hình.
    - **Xử lý sự kiện**: Vận dụng pygame.event để bắt các sự kiện từ bàn phím (nhấn phím, thả phím), từ đó điều khiển hành động của tàu vũ trụ.
    - **Quản lý thời gian và vòng lặp game**: Sử dụng pygame.time.Clock để điều khiển tốc độ khung hình (FPS) và đảm bảo game chạy mượt mà, đồng bộ trên các hệ thống khác nhau.
    - **Xử lý âm thanh**: Áp dụng pygame.mixer để tải và phát các file âm thanh (.wav, .mp3) cho hiệu ứng và nhạc nền.
- **Kiến thức về toán học và vật lý cơ bản**: Để xử lý chuyển động và xoay các đối tượng, các kiến thức về **vectơ**, **phép quay**, và **phép tịnh tiến** trong không gian 2D đã được áp dụng. Điều này bao gồm việc tính toán hướng di chuyển dựa trên góc quay của tàu và áp dụng các công thức lượng giác để cập nhật vị trí.
- **Giải thuật phát hiện va chạm**: Vận dụng các thuật toán cơ bản để phát hiện sự giao nhau giữa các đối tượng. Trong trường hợp này, việc sử dụng các chức năng va chạm có sẵn của Pygame (ví dụ: rect.colliderect cho va chạm hình chữ nhật hoặc pygame.sprite.collide_mask cho va chạm chi tiết hơn) là rất quan trọng.
- **Tư duy logic và giải quyết vấn đề**: Đây là yếu tố không thể thiếu, được thể hiện trong việc thiết kế luồng game, xử lý các trường hợp đặc biệt (ví dụ: thiên thạch ra khỏi màn hình sẽ tái tạo ở vị trí khác, điểm số tăng khi tiêu diệt), và debugging khi chương trình gặp lỗi để đảm bảo tính ổn định và chính xác của trò chơi.

1.3. Phạm vi thực hiện

Đề tài này được thực hiện trong khuôn khổ môn học **Lập trình ứng dụng đa phương tiện** và được giới hạn trong việc tập trung vào việc triển khai các chức năng chính yếu của một trò chơi 2D đơn giản, nhằm đảm bảo tính khả thi và hoàn thành trong thời gian cho phép. Dự án **không phát triển các tính năng phức tạp** như khả năng lưu trữ trạng thái game (save game), tải lại game (load game) hoặc hệ thống nhiều màn chơi với độ khó tăng dần. Mục tiêu chính là làm chủ các cơ chế cơ bản thay vì mở rộng quy mô.

Thứ hai, trò chơi chỉ được thiết kế để hoạt động ở **chế độ người chơi đơn**. Điều này có nghĩa là không có chế độ chơi mạng hoặc chơi cùng nhiều người trên cùng một thiết bị. Mọi thao tác điều khiển nhân vật chính đều được thực hiện thông qua **bàn phím**. Để đảm bảo tính khả thi và tập trung vào phần lập trình cốt lõi, đề tài sẽ **sử dụng các hình ảnh và âm thanh có sẵn hoặc miễn phí** từ các nguồn tài nguyên công cộng. Việc này giúp tiết kiệm thời gian thiết kế đồ họa và âm thanh, cho phép người thực hiện dành nhiều công sức hơn vào việc phát triển logic game. Cuối cùng, hệ thống sẽ được phát triển và chạy hoàn toàn trên môi trường lập trình **Python 3.x**, tận dụng sức mạnh và tính linh hoạt của ngôn ngữ này cùng với thư viện chuyên dụng **Pygame**, vốn nổi tiếng với khả năng hỗ trợ phát triển game 2D hiệu quả.

1.4. Phương pháp thực hiện

Để hoàn thành đề tài này một cách hiệu quả và có hệ thống, người thực hiện đã áp dụng một chuỗi các bước phương pháp luận được thiết kế rõ ràng. Bước đầu tiên và rất quan trọng là **nghiên cứu sâu rộng về thư viện Pygame**. Quá trình này bao gồm việc tìm hiểu chi tiết các module chính của Pygame, đặc biệt là những module liên quan trực tiếp đến việc xử lý đồ họa (như pygame.display để quản lý cửa sổ hiển thị, pygame.image để tải và xử lý hình ảnh), âm thanh (module pygame.mixer để quản lý nhạc nền và hiệu ứng âm thanh), và các module xử lý **sự kiện bàn phím** (như pygame.event để nắm bắt các sự kiện người dùng và pygame.key để kiểm tra trạng thái phím). Ngoài ra, việc tham khảo các

ví dụ và tài liệu từ các phiên bản Astrocrash trước đó (như **Astrocrash01 đến Astrocrash08**) đã cung cấp một lộ trình rõ ràng để bắt đầu.

Sau khi nắm vững công cụ và tham khảo cấu trúc mẫu, bước tiếp theo là **phân tích và thiết kế đối tượng**, cũng như **setup các lớp game**. Dựa trên yêu cầu của trò chơi, người thực hiện đã tiến hành xây dựng các lớp (class) đối tượng chính bao gồm **Ship** (tàu vũ trụ của người chơi), **Asteroid** (thiên thạch), **Missile** (tên lửa), và **Explosion** (hiệu ứng nổ). Một sơ đồ tương tác giữa các lớp này cũng được xây dựng để định hình rõ ràng mối quan hệ và cách chúng tương tác với nhau trong game, đảm bảo tính nhất quán và logic của toàn bộ hệ thống.

Tiếp theo là giai đoạn **cài đặt và kiểm thử** liên tục. Người thực hiện đã tiến hành viết mã nguồn dựa trên thiết kế đã định, đặc biệt chú trọng vào việc **xử lý đầu vào từ bàn phím** và áp dụng **các công thức toán học** để tính toán chuyển động và xoay của các đối tượng. Sau mỗi phần mã được cài đặt, trò chơi sẽ được chạy thử nghiệm để kiểm tra chức năng. Quá trình này liên tục bao gồm việc **sửa lỗi** (debugging) để khắc phục các vấn đề phát sinh và **tinh chỉnh gameplay** để đảm bảo trò chơi hoạt động mượt mà, cân bằng và mang lại trải nghiệm tốt nhất cho người chơi (ví dụ: bắn trúng asteroid phải nổ và cộng 10 điểm, kèm theo sound "boom").

Một yếu tố quan trọng để hoàn thiện trải nghiệm game là **tải và phát các sound và music**. Trong giai đoạn này, các chức năng của pygame.mixer được sử dụng để phát các hiệu ứng âm thanh phù hợp, ví dụ như tiếng bắn khi người chơi khai hỏa, tiếng nổ khi tên lửa va chạm với thiên thạch, và một bản nhạc nền xuyên suốt để tạo không khí. Việc tích hợp âm thanh được thực hiện một cách có chiến lược, đảm bảo chúng được phát đúng thời điểm và tăng cường sự sống động cho trò chơi.

Tóm tắt chuong

Trong chương này em trình bày lý do chọn đề tài Astrocrash – một trò chơi bắn thiên thạch 2D được phát triển bằng Python và thư viện Pygame nhằm giúp người học rèn luyện kỹ năng lập trình, xử lý đồ họa và âm thanh. Nội dung bao gồm tổng quan chương trình, đầu vào/đầu ra, các tính năng chính như điều khiển tàu, bắn tên lửa, xử lý va chạm và âm thanh. Chương cũng nêu rõ các thách thức kỹ thuật, phạm vi thực hiện trong khuôn khổ môn học và phương pháp triển khai theo hướng tiếp cận từng bước, từ phân tích đến kiểm thử.

CHƯƠNG II : CƠ SỞ LÝ THUYẾT

2.1. Ngôn ngữ lập trình Python và thư viện Pygame

Để phát triển Astrocrash, sự kết hợp giữa ngôn ngữ lập trình Python và thư viện Pygame đóng vai trò chủ chốt, tạo nên một môi trường mạnh mẽ và linh hoạt.

2.1.1. Ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình bậc cao, thông dịch và đa nền tảng, nổi bật với cú pháp tường minh và dễ đọc, gần gũi với ngôn ngữ tự nhiên. Sự phổ biến của Python được minh chứng qua ứng dụng rộng rãi trong nhiều lĩnh vực, từ phát triển web (với các framework như Django, Flask), khoa học dữ liệu (NumPy, Pandas, SciPy), trí tuệ nhân tạo (TensorFlow, PyTorch) cho đến tự động hóa, và đặc biệt là phát triển các ứng dụng đa phương tiện cũng như trò chơi 2D.

Những đặc điểm nổi bật của Python đã khiến nó trở thành lựa chọn lý tưởng cho dự án này:

- **Dễ học và dễ sử dụng**: Cú pháp đơn giản, ít quy tắc phức tạp giúp người lập trình tập trung vào logic vấn đề hơn là các chi tiết kỹ thuật của ngôn ngữ.
- **Tính linh hoạt và đa năng**: Python hỗ trợ nhiều paradigms lập trình như hướng đối tượng, hướng thủ tục và hướng chức năng, cho phép người phát triển lựa chọn phong cách phù hợp nhất.
- **Thư viện phong phú**: Python sở hữu một hệ sinh thái thư viện khổng lồ, được phát triển và duy trì bởi cộng đồng lớn mạnh. Đối với game, Pygame là một ví dụ điển hình.
- **Khả năng tương thích cao**: Python có thể chạy trên nhiều hệ điều hành khác nhau như Windows, macOS, Linux, đảm bảo tính di động cho ứng dụng.

Trong Astrocrash, toàn bộ **logic game**, từ việc khởi tạo màn hình, quản lý vòng lặp trò chơi, xử lý đầu vào từ người dùng, cập nhật trạng thái của các đối tượng, cho đến việc hiển thị đồ họa và phát âm thanh, đều được viết bằng Python.

2.1.2. Thư viện Pygame

Pygame là một bộ thư viện được viết bằng Python, được thiết kế đặc biệt để hỗ trợ việc phát triển trò chơi điện tử 2D. Nó đóng vai trò là một lớp bao bọc (wrapper) cho Simple DirectMedia Layer (SDL), một thư viện đa nền tảng viết bằng C/C++ được sử dụng rộng rãi để truy cập trực tiếp vào phần cứng đa phương tiện. Nhờ Pygame, các tác vụ phức tạp liên quan đến đồ họa, âm thanh, và sự kiện được đơn giản hóa đáng kể, giúp người phát triển tập trung vào ý tưởng game.

Các module chính của Pygame được sử dụng trong Astrocrash bao gồm:

- **pygame.display**: Module này chịu trách nhiệm quản lý cửa sổ game và màn hình hiển thị. Nó cho phép **khởi tạo kích thước và tiêu đề cửa sổ**, **cập nhật nội dung hiển thị** (chuyển đổi giữa các buffer để tạo hiệu ứng động mượt mà), và quản lý các chế độ hiển thị như toàn màn hình hoặc cửa sổ. Đây là nơi mọi hình ảnh và hiệu ứng sẽ được vẽ lên.
- **pygame.image**: Module này cung cấp các chức năng để **tải hình ảnh** từ các tệp tin (ví dụ: .png, .jpg), **chuyển đổi định dạng** hình ảnh để tối ưu hóa hiệu suất hiển thị, và thực hiện các phép biến đổi hình học cơ bản như **xoay (pygame.transform.rotate())**, **co giãn (pygame.transform.scale())** hoặc **lật (pygame.transform.flip())** các sprite. Trong Astrocrash, nó được dùng để tải các hình ảnh của tàu, thiên thạch, tên lửa và hiệu ứng nổ.
- **pygame.event**: Đây là trái tim của tương tác người dùng. Module này liên tục **lắng nghe và thu nhận các sự kiện** xảy ra từ hệ thống, chẳng hạn như nhấn phím bàn phím (KEYDOWN, KEYUP), di chuyển hoặc click chuột (MOUSEMOTION, MOUSEBUTTONDOWN), hoặc các sự kiện hệ thống như đóng cửa sổ (QUIT).

Việc xử lý các sự kiện này là cần thiết để người chơi có thể điều khiển tàu vũ trụ (xoay, di chuyển, bắn) và chương trình có thể phản ứng lại.

- **pygame.mixer**: Module này quản lý tất cả các khía cạnh liên quan đến âm thanh trong game. Nó cho phép **tải các file âm thanh** dưới dạng hiệu ứng ngắn (pygame.mixer.Sound) cho các sự kiện như tiếng bắn, tiếng nổ, và **quản lý nhạc nền (pygame.mixer.music)** cho toàn bộ trò chơi. pygame.mixer cũng hỗ trợ quản lý các **kênh âm thanh** để có thể phát nhiều hiệu ứng đồng thời mà không bị chồng chéo hay gián đoạn.
- **pygame.time**: Module này cung cấp các chức năng liên quan đến thời gian, đặc biệt là **pygame.time.Clock**. Clock được sử dụng để **giới hạn tốc độ khung hình (FPS)** của game, đảm bảo rằng trò chơi chạy với tốc độ ổn định và mượt mà trên các cấu hình phần cứng khác nhau, tránh tình trạng game chạy quá nhanh hoặc quá chậm. Nó cũng cho phép tính toán thời gian trôi qua giữa các khung hình để đảm bảo các chuyển động diễn ra đồng bộ, không phụ thuộc vào tốc độ xử lý của máy tính.

2.2. Lập trình hướng đối tượng (OOP)

Lập trình hướng đối tượng (OOP) là một mô hình lập trình mạnh mẽ, dựa trên khái niệm "đối tượng", trong đó mỗi đối tượng là một thực thể độc lập có thể chứa dữ liệu (được gọi là **thuộc tính** hay **trạng thái**) và các hàm hoặc thủ tục xử lý dữ liệu đó (được gọi là **phương thức** hay **hành vi**). OOP giúp tổ chức mã nguồn một cách có hệ thống, tăng tính tái sử dụng, dễ bảo trì và mở rộng chương trình.

Trong dự án Astrocrash, mô hình OOP được áp dụng triệt để để xây dựng cấu trúc của trò chơi:

- **Các lớp (Classes)**: Mỗi loại thực thể trong game được định nghĩa là một **lớp**. Ví dụ, trong Astrocrash, chúng ta có các lớp sau:

- **Ship**: Đại diện cho tàu vũ trụ của người chơi. Lớp này có các thuộc tính như vị trí (x, y), góc quay, tốc độ, hình ảnh sprite, và các phương thức như move(), rotate(), shoot(), update(), draw().
- **Asteroid**: Đại diện cho các thiên thạch. Các thuộc tính bao gồm vị trí, tốc độ, hướng di chuyển, kích thước, hình ảnh sprite. Các phương thức có thể là move(), update(), draw().
- **Missile**: Đại diện cho tên lửa được bắn ra từ tàu. Thuộc tính bao gồm vị trí, tốc độ, hướng. Phương thức update() sẽ tính toán vị trí mới của tên lửa sau mỗi khung hình.
- **Explosion**: Đại diện cho hiệu ứng nổ. Thuộc tính có thể là vị trí, giai đoạn nổ (để hiển thị các khung hình động của vụ nổ), hình ảnh các khung hình nổ. Phương thức update() sẽ chuyển đổi giữa các khung hình nổ cho đến khi hiệu ứng kết thúc.

- **Đối tượng (Objects)**: Là các thể hiện cụ thể của một lớp. Khi trò chơi khởi chạy, một đối tượng Ship sẽ được tạo ra. Trong suốt quá trình chơi, nhiều đối tượng Asteroid và Missile sẽ liên tục được tạo ra và hủy bỏ. Mỗi đối tượng là một thực thể độc lập với các giá trị thuộc tính riêng biệt (ví dụ: thiên thạch A ở vị trí này với tốc độ này, thiên thạch B ở vị trí khác với tốc độ khác) nhưng chia sẻ cùng một bộ hành vi được định nghĩa trong lớp của chúng.
- **Tính đóng gói (Encapsulation)**: Mỗi lớp tự quản lý dữ liệu và hành vi của mình. Ví dụ, lớp Ship chứa tất cả logic để điều khiển tàu, bắn tên lửa, và cập nhật trạng thái của nó mà không làm ảnh hưởng đến các phần khác của chương trình. Điều này giúp tách biệt các thành phần, dễ dàng sửa đổi một phần mà không gây ảnh hưởng đến tổng thể.
- **Tính đa hình (Polymorphism) và Kế thừa (Inheritance)**: Mặc dù trong một game đơn giản như Astrocrash, việc sử dụng kế thừa có thể không quá phức tạp, nhưng trong các game lớn hơn, các lớp đối tượng thường kế thừa từ một lớp cơ sở chung (ví dụ: GameObject hoặc Sprite) để chia sẻ các thuộc tính và phương thức chung

như position, image, update(), draw(). Tính đa hình cho phép xử lý các đối tượng thuộc các lớp khác nhau thông qua một giao diện chung, ví dụ: một danh sách các GameObject có thể được duyệt để gọi phương thức update() và draw() cho từng đối tượng mà không cần biết chính xác loại đối tượng đó là gì.

Việc áp dụng OOP giúp code trở nên có tổ chức, dễ đọc, dễ kiểm thử và cho phép tái sử dụng code hiệu quả khi mở rộng game sau này, ví dụ thêm các loại thiên thạch mới hoặc tàu vũ trụ mới với các đặc điểm khác nhau.

2.3. Cấu trúc dữ liệu và giải thuật cơ bản

Trong quá trình phát triển Astrocrash, việc lựa chọn và sử dụng các cấu trúc dữ liệu cùng với các giải thuật cơ bản đã giúp quản lý tài nguyên và xử lý logic game một cách hiệu quả.

- **Danh sách (List)**: Đây là một trong những cấu trúc dữ liệu cơ bản và linh hoạt nhất trong Python, cho phép lưu trữ một tập hợp các phần tử theo thứ tự. Trong Astrocrash, list được sử dụng rộng rãi để quản lý các nhóm đối tượng đồng nhất đang tồn tại trong game:
    - all_sprites_group: Một danh sách chứa tất cả các đối tượng (Ship, Asteroid, Missile, Explosion) cần được cập nhật và vẽ trên màn hình. Pygame cung cấp pygame.sprite.Group là một lớp chuyên biệt để quản lý các sprite hiệu quả hơn.
    - asteroid_group: Chứa tất cả các đối tượng thiên thạch hiện đang xuất hiện trong game.
    - missile_group: Chứa tất cả các đối tượng tên lửa đang bay trên màn hình.
    - explosion_group: Chứa các đối tượng hiệu ứng nổ đang diễn ra. Trong mỗi vòng lặp game, chương trình sẽ duyệt qua các danh sách này để gọi phương thức update() cho từng đối tượng (cập nhật vị trí, trạng thái) và phương thức draw() để vẽ chúng lên màn hình. Các đối tượng không còn hoạt động (ví dụ:

thiên thạch bị phá hủy, tên lửa bay ra khỏi màn hình) sẽ được loại bỏ khỏi danh sách để giải phóng bộ nhớ.

- **Dictionary (Từ điển)**: Cấu trúc dữ liệu này lưu trữ dữ liệu dưới dạng các cặp **khóa-giá trị**, cho phép truy cập phần tử nhanh chóng thông qua khóa của nó. Trong Astrocrash, dictionary có thể được sử dụng để:
    - Lưu trữ các tài nguyên game như hình ảnh hoặc âm thanh đã được tải vào bộ nhớ, ví dụ: images = {"ship": ship_image, "asteroid": asteroid_image} hoặc sounds = {"boom": boom_sound_effect}. Điều này giúp dễ dàng truy cập và tái sử dụng các tài nguyên.
    - Lưu trữ các cấu hình game, ví dụ: game_settings = {"screen_width": 800, "screen_height": 600, "player_speed": 5}.
- **Giải thuật kiểm tra va chạm (Collision Detection Algorithms)**: Đây là một trong những giải thuật quan trọng nhất trong mọi trò chơi. Astrocrash sử dụng các phương pháp va chạm dựa trên hình dạng của sprite:
    - **Va chạm hình chữ nhật bao bọc (Bounding Box Collision)**: Đây là phương pháp đơn giản và nhanh nhất. Mỗi sprite được gán một hình chữ nhật bao quanh nó (sprite.rect trong Pygame). Va chạm được phát hiện khi các hình chữ nhật của hai sprite chồng lấn lên nhau. Phương pháp này hiệu quả cho các đối tượng có hình dạng gần giống chữ nhật, nhưng có thể kém chính xác đối với các sprite có hình dạng phức tạp và nhiều khoảng trống trong suốt, dẫn đến va chạm giả (collision with empty space).
    - **Va chạm dựa trên mặt nạ (Mask-based Collision)**: Để tăng độ chính xác, đặc biệt với các sprite có hình dạng bất quy tắc như thiên thạch hoặc tàu vũ trụ, Pygame cung cấp các chức năng va chạm dựa trên mặt nạ (pygame.sprite.collide_mask hoặc pygame.mask.from_surface). Phương pháp này tạo ra một "mặt nạ" bit (mask) từ các pixel không trong suốt của hình ảnh. Va chạm chỉ được xác định khi các pixel thực sự của hai mặt nạ

chồng lấn lên nhau. Điều này cung cấp độ chính xác pixel-perfect, giúp trải nghiệm game chân thực hơn.

- **Giải thuật sinh ngẫu nhiên (Random Number Generation)**: Được sử dụng để tạo ra sự đa dạng và không thể đoán trước trong game. Trong Astrocrash, giải thuật này được áp dụng để:
    - **Tạo vị trí ban đầu ngẫu nhiên** cho các thiên thạch khi chúng xuất hiện trên màn hình.
    - **Xác định hướng và tốc độ di chuyển ngẫu nhiên** cho từng thiên thạch, làm cho mỗi lần chơi trở nên khác biệt.
    - Có thể dùng để chọn ngẫu nhiên loại thiên thạch nếu có nhiều loại khác nhau.

2.4. Khái niệm về đồ họa 2D trong game

Để tạo ra một môi trường tương tác trực quan, việc hiểu rõ các khái niệm cơ bản về đồ họa 2D là điều cần thiết.

- **Sprite**: Là một hình ảnh 2D nhỏ, thường được sử dụng để biểu diễn một đối tượng hoặc một nhân vật trong trò chơi. Trong Pygame, sprite thường được quản lý thông qua đối tượng pygame.Surface. Mỗi sprite có thể có nhiều trạng thái hoặc khung hình (ví dụ: các khung hình của hiệu ứng nổ) để tạo ra hoạt ảnh (animation).
- **Tọa độ và Hệ tọa độ**: Trong đồ họa máy tính 2D, đặc biệt là với Pygame, hệ tọa độ được sử dụng là hệ tọa độ Descartes với một quy ước đặc biệt:
    - **Gốc (0,0)**: Nằm ở góc trên bên trái của màn hình.
    - **Trục x**: Tăng dần về phía phải màn hình.
    - **Trục y**: Tăng dần về phía dưới màn hình (ngược với hệ tọa độ toán học thông thường). Vị trí của mọi đối tượng trên màn hình đều được xác định bằng một cặp tọa độ (x, y). Việc điều chỉnh các giá trị x và y là cách để di chuyển các đối tượng.

- **Phép biến đổi hình học (Geometric Transformations)**: Là các thao tác toán học được áp dụng lên các sprite để thay đổi vị trí, hướng hoặc kích thước của chúng.
    - **Tịnh tiến (Translation)**: Là phép dịch chuyển một đối tượng từ vị trí này sang vị trí khác bằng cách cộng thêm một giá trị vào tọa độ x và y hiện tại của nó. Đây là phép biến đổi cơ bản nhất để tạo ra chuyển động cho tàu, tên lửa và thiên thạch.
    - **Quay (Rotation)**: Là phép xoay một đối tượng quanh một điểm cố định (thường là tâm của đối tượng). Trong Astrocrash, tàu vũ trụ cần được xoay theo góc mà người chơi điều khiển để hướng về phía mong muốn. Pygame cung cấp hàm pygame.transform.rotate() để thực hiện thao tác này, trả về một bề mặt mới đã được xoay.
    - **Co giãn (Scaling)**: Là phép thay đổi kích thước của đối tượng (phóng to hoặc thu nhỏ). Mặc dù không phải là tính năng chính trong Astrocrash, nhưng đây là một phép biến đổi cơ bản có thể được sử dụng để điều chỉnh kích thước của các thiên thạch (ví dụ: tạo các thiên thạch lớn, nhỏ) hoặc hiệu ứng nổ.
- **Game Loop (Vòng lặp trò chơi)**: Đây là kiến trúc xương sống của mọi trò chơi. Game loop là một vòng lặp vô hạn chạy liên tục cho đến khi trò chơi kết thúc. Trong mỗi lần lặp (tức là mỗi khung hình), game loop thực hiện một chuỗi các bước quan trọng để cập nhật và hiển thị trạng thái trò chơi:
    1. **Xử lý đầu vào (Input Handling)**: Lắng nghe và xử lý tất cả các sự kiện từ người dùng (nhấn phím, di chuyển chuột).
    2. **Cập nhật logic game (Update Game Logic)**: Tính toán và cập nhật trạng thái của tất cả các đối tượng trong game: di chuyển tàu, thiên thạch, tên lửa; kiểm tra và xử lý va chạm; cập nhật điểm số; quản lý các hiệu ứng nổ.
    3. **Vẽ/Hiển thị (Drawing/Rendering)**: Xóa toàn bộ màn hình hoặc các vùng cần vẽ lại, sau đó vẽ lại tất cả các đối tượng đã được cập nhật lên một bộ đệm (buffer) ẩn.

4. **Cập nhật màn hình (Display Update)**: Chuyển đổi bộ đệm đã vẽ lên màn hình thực tế để người chơi có thể nhìn thấy những thay đổi.
5. **Điều khiển thời gian (Time Management)**: Sử dụng pygame.time.Clock để giới hạn tốc độ vòng lặp, đảm bảo game chạy ở một tốc độ khung hình (FPS) ổn định, không quá nhanh hay quá chậm.

2.5. Xử lý âm thanh trong game

Âm thanh là một yếu tố không thể thiếu để tạo nên một trải nghiệm game hoàn chỉnh và hấp dẫn. Nó không chỉ cung cấp phản hồi tức thì cho người chơi mà còn giúp tạo không khí và chiều sâu cho thế giới game. Pygame cung cấp module pygame.mixer để quản lý các tác vụ này một cách hiệu quả.

- **Nhạc nền (Background Music)**: Là bản nhạc được phát liên tục trong suốt quá trình chơi game để tạo bầu không khí và cảm xúc chung. Trong Pygame, nhạc nền được quản lý thông qua đối tượng pygame.mixer.music. Các chức năng chính bao gồm:
    - pygame.mixer.music.load(filename): Tải một tệp nhạc (thường là .mp3, .ogg, .mid) vào bộ nhớ.
    - pygame.mixer.music.play(loops=0): Bắt đầu phát nhạc. Tham số loops xác định số lần lặp lại (0 để phát một lần, -1 để lặp vô hạn).
    - pygame.mixer.music.stop(): Dừng phát nhạc nền.
    - pygame.mixer.music.set_volume(value): Điều chỉnh âm lượng của nhạc nền (từ 0.0 đến 1.0).
- **Hiệu ứng âm thanh (Sound Effects - SFX)**: Là các âm thanh ngắn, được phát ra khi có một sự kiện cụ thể xảy ra trong game, cung cấp phản hồi tức thì cho hành động của người chơi hoặc các sự kiện trong game. Ví dụ điển hình trong Astrocrash là tiếng bắn tên lửa, tiếng nổ khi thiên thạch bị phá hủy. Các hiệu ứng âm thanh được quản lý bằng đối tượng pygame.mixer.Sound:

    - pygame.mixer.Sound(filename): Tải một tệp âm thanh (thường là .wav, .ogg) vào bộ nhớ. Do các hiệu ứng âm thanh thường ngắn và cần được phát ngay lập tức, chúng thường được tải toàn bộ vào RAM.
    - sound_object.play(): Phát hiệu ứng âm thanh.
    - sound_object.set_volume(value): Điều chỉnh âm lượng của hiệu ứng âm thanh cụ thể.
- **Quản lý kênh âm thanh (Channels)**: Khi nhiều hiệu ứng âm thanh cần được phát đồng thời (ví dụ: vừa bắn, vừa có tiếng nổ, vừa có tiếng tàu bay), việc quản lý kênh âm thanh trở nên quan trọng. Pygame cho phép cấu hình số lượng kênh âm thanh (pygame.mixer.set_num_channels()). Mỗi hiệu ứng âm thanh có thể được gán cho một kênh riêng biệt để đảm bảo chúng không làm gián đoạn lẫn nhau. Ví dụ, tiếng bắn có thể được phát trên một kênh, tiếng nổ trên kênh khác, và nhạc nền có một kênh riêng.

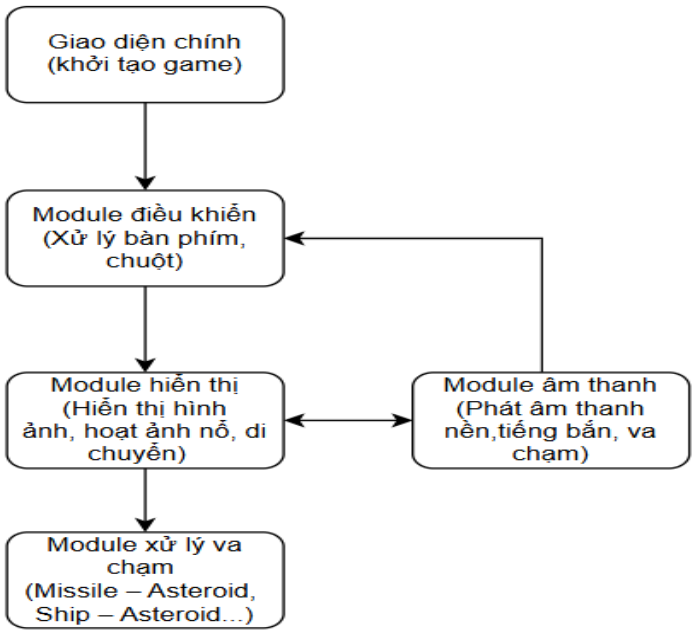
Tóm tắt chương

Chương này em trình bày cơ sở lý thuyết về ngôn ngữ lập trình Python và thư viện Pygame, nền tảng chính để phát triển game Astrocrash. Python với cú pháp đơn giản, hỗ trợ lập trình hướng đối tượng giúp dễ dàng xây dựng game. Pygame cung cấp các module xử lý đồ họa, âm thanh và sự kiện, tạo điều kiện thuận lợi cho phát triển game 2D. Các khái niệm về lập trình hướng đối tượng, xử lý sự kiện và quản lý tài nguyên cũng được đề cập để làm nền tảng kỹ thuật cho dự án

CHƯƠNG III : THIẾT KẾ VÀ XÂY DỰNG CHƯƠNG TRÌNH

3.1. Sơ đồ khối hệ thống

Hệ thống trò chơi được tổ chức theo cấu trúc module, nhằm đảm bảo tính rõ ràng, dễ quản lý và mở rộng trong quá trình phát triển. Các module chính bao gồm: Giao diện chính, Điều khiển, Hiển thị, Âm thanh và Xử lý va chạm. Các thành phần này tương tác với nhau theo sơ đồ sau:



Hình 3.1 : Sơ đồ hệ thống

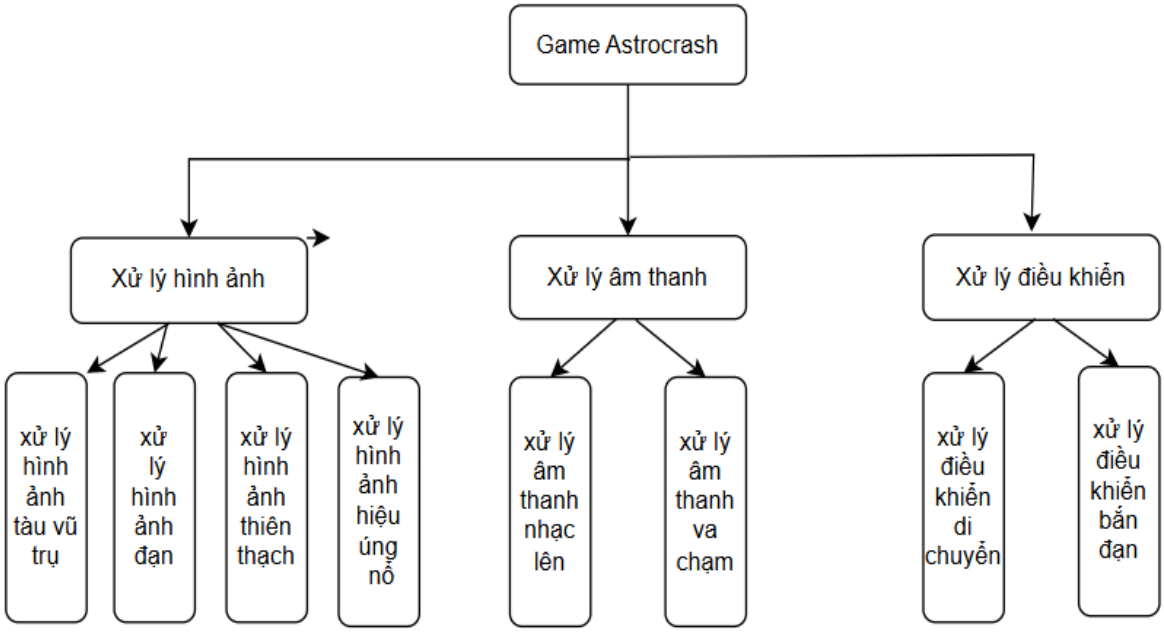
3.1.2. Các module chính trong chương trình

1. Giao diện chính (Main Interface)
    - Đóng vai trò là điểm khởi đầu của trò chơi.
    - Thực hiện khởi tạo cửa sổ game, thiết lập kích thước, tốc độ khung hình (FPS), màu nền.
    - Tải tài nguyên đồ họa và âm thanh từ thư mục assets/images và assets/sounds.
    - Tạo vòng lặp chính của trò chơi để duy trì hoạt động liên tục.
2. Module điều khiển (Control Module)
    - Tiếp nhận và xử lý các sự kiện từ bàn phím và chuột.
    - Cụ thể, sử dụng các phím điều hướng hoặc tổ hợp phím để di chuyển tàu vũ trụ.
    - Nhấn phím cách (SPACE) để bắn đạn, từ đó kích hoạt tạo đối tượng Missile mới.
    - Liên tục cập nhật trạng thái điều khiển vào mỗi khung hình.
3. Module hiển thị (Graphics/Render Module)
    - Đảm nhiệm việc hiển thị tất cả các đối tượng như: tàu vũ trụ, thiên thạch, đạn, vụ nổ... lên màn hình.
    - Dùng các hình ảnh đã chuẩn bị như ship.png, missile.png, asteroid.png, explosion.png.
    - Hiển thị điểm số, thông báo kết thúc trò chơi, và các hiệu ứng hoạt ảnh.
4. Module âm thanh (Sound Module)
    - Phát nhạc nền (background.mp3) khi bắt đầu trò chơi.
    - Phát các hiệu ứng âm thanh như:
        - Âm thanh khi bắn (shoot.wav),
        - Âm thanh khi va chạm nổ (boom.wav).
    - Được đồng bộ với các sự kiện tương ứng từ các module khác như điều khiển và va chạm.
5. Module xử lý va chạm (Collision Detection Module)
    - Kiểm tra va chạm giữa các đối tượng trong trò chơi, chẳng hạn:

        - Đạn (Missile) va chạm với thiên thạch (Asteroid): thiên thạch biến mất, phát hiệu ứng nổ và âm thanh, cộng điểm.
        - Tàu (Ship) va chạm với thiên thạch: phát hiệu ứng nổ, trừ máu hoặc kết thúc trò chơi nếu hết lượt.
    - Tạo các đối tượng Explosion khi xảy ra va chạm để hiển thị hiệu ứng.

3.1.2.Biểu đồ phân cấp chức năng

Biểu đồ phân cấp chức năng mô tả các chức năng chính của trò chơi *Astrocrash* theo cấu trúc dạng cây, từ chức năng tổng quát đến các chức năng con cụ thể. Cách phân chia này giúp tổ chức chương trình theo hướng module rõ ràng, thuận tiện cho việc phát triển và bảo trì hệ thống.



Hình 3.2 : Biểu đồ phân cấp chức năng

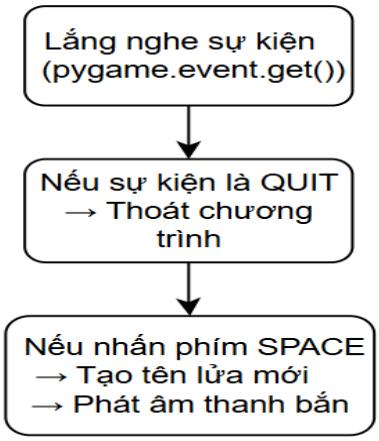
Giải thích các chức năng:

- Game Astrocrash: Chức năng tổng thể điều hành trò chơi.
    - Xử lý hình ảnh: Quản lý việc hiển thị các đối tượng trong game.
        - Vẽ tàu vũ trụ: Hiển thị tàu người chơi.

        - Vẽ đạn, thiên thạch, hiệu ứng nổ: Hiển thị các vật thể và hiệu ứng liên quan trong game.
    - Xử lý âm thanh: Phát âm thanh nền và hiệu ứng khi xảy ra va chạm, bắn đạn.
        - Nhạc nền: Phát liên tục trong quá trình chơi.
        - Âm thanh va chạm: Phát khi có vụ nổ hoặc va chạm.
    - Xử lý điều khiển: Nhận và xử lý các thao tác của người chơi.
        - Di chuyển tàu: Điều hướng tàu qua phím bấm.
        - Bắn đạn: Tạo đạn khi người chơi nhấn nút.

3.2. Sơ đồ khối các thuật toán chính

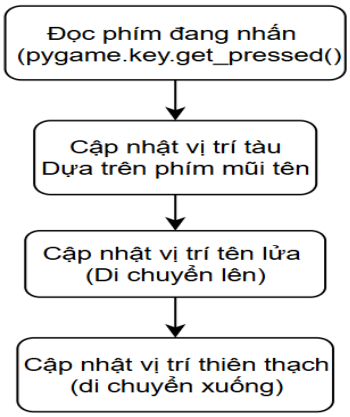
1. Xử lý sự kiện người chơi



Hình 3.3 : Xử lý sự kiện người chơi

- **Chức năng**: Lắng nghe và xử lý các sự kiện như nhấn phím, đóng cửa sổ game.
- **Đầu vào**: Sự kiện từ bàn phím và hệ thống (từ pygame.event.get()).
- **Đầu ra**: Tạo đạn (nếu nhấn SPACE), thoát game (nếu nhấn QUIT), chuyển trạng thái (Game Over, Victory).
- **Quan hệ**: Kết nối trực tiếp với các thuật toán **bắn tên lửa**, **chuyển trạng thái game**.

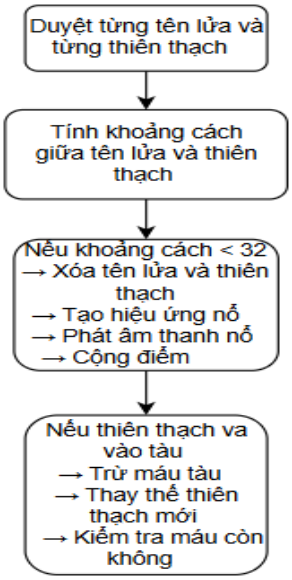
2. Cập nhật vị trí đối tượng



Hình 3.4 : Cập nhật vị trí đối tượng

- **Chức năng**: Tính toán và cập nhật vị trí của tàu, tên lửa và thiên thạch theo thời gian.
- **Đầu vào**: Trạng thái bàn phím (di chuyển tàu), danh sách đạn, thiên thạch hiện tại.
- **Đầu ra**: Vị trí mới của các đối tượng trên màn hình.
- **Quan hệ**: Tác động đến **vẽ đối tượng**, **kiểm tra va chạm**.

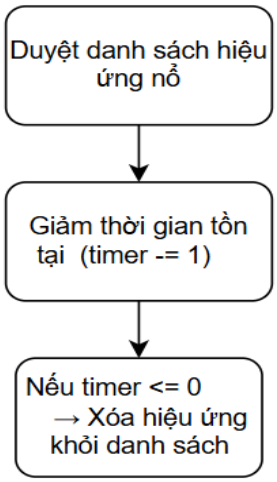
3. Kiểm tra va chạm



Hình 3.5 : Kiểm tra va chạm

- **Chức năng**: Kiểm tra va chạm giữa đạn và thiên thạch, tàu và thiên thạch.
- **Đầu vào**: Tọa độ tàu, đạn, thiên thạch.
- **Đầu ra**: Cập nhật điểm số, trừ máu tàu, tạo hiệu ứng nổ, âm thanh.
- **Quan hệ**: Gọi đến **cập nhật hiệu ứng**, liên kết với **trạng thái game** và **hiển thị giao diện**.

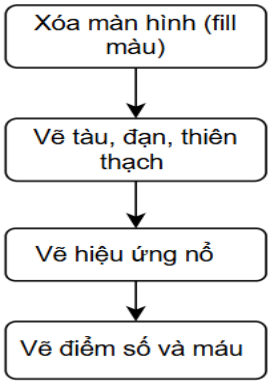
4. Cập nhật hiệu ứng nổ



Hình 3.6 : Cập nhật hiệu ứng nổ

- **Chức năng**: Hiển thị hiệu ứng nổ trong thời gian ngắn sau va chạm.
- **Đầu vào**: Vị trí va chạm.
- **Đầu ra**: Vẽ hiệu ứng nổ và xóa sau khi hết thời gian.
- **Quan hệ**: Được gọi từ **kiểm tra va chạm**, kết nối với **hiển thị**.

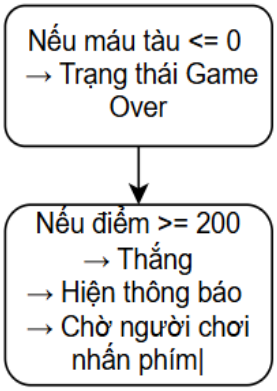
5. Hiển thị giao diện



Hình 3.7 : Hiển thị trò chơi

- **Chức năng**: Vẽ toàn bộ màn hình game gồm tàu, đạn, thiên thạch, hiệu ứng, điểm và máu.
- **Đầu vào**: Trạng thái hiện tại của các đối tượng.
- **Đầu ra**: Cập nhật hình ảnh hiển thị trên màn hình.
- **Quan hệ**: Nhận dữ liệu từ hầu hết các thuật toán khác.

6. Kiểm tra điều kiện kết thúc



Hình 3.8 : Kiểm tra điều kiện kết thúc

- **Chức năng**: Kiểm tra xem game đã thua (hết máu) hay thắng (đủ điểm).
- **Đầu vào**: Biến điểm (score) và máu (health).

- **Đầu ra**: Hiện thông báo chiến thắng hoặc thua, đợi nhấn ENTER hoặc ESC.
- **Quan hệ**: Tác động đến **vòng lặp chính**, liên kết với **xử lý sự kiện**.

3.3. Cấu trúc dữ liệu

Trong quá trình phát triển trò chơi "Astrocrash", chúng tôi đã xây dựng một hệ thống dữ liệu được tổ chức theo hướng đối tượng, mô phỏng tương đương với các bảng dữ liệu logic trong một hệ thống quản lý dữ liệu. Mỗi lớp đối tượng trong game được xem như một bảng dữ liệu với các thuộc tính riêng biệt, nhằm phục vụ cho việc quản lý trạng thái, xử lý logic và hiển thị giao diện. Các lớp dữ liệu chính bao gồm: Ship, Missile, Asteroid và Explosion.

1. Lớp dữ liệu Ship (Tàu vũ trụ)

Lớp Ship đại diện cho nhân vật chính mà người chơi điều khiển. Tàu có thể di chuyển theo bốn hướng (trái, phải, lên, xuống) và sẽ bị mất máu nếu va chạm với thiên thạch. Các thuộc tính chính của lớp bao gồm:

- image: Hình ảnh của tàu vũ trụ, dùng để hiển thị lên màn hình.
- x: Tọa độ ngang của tàu trên màn hình (kiểu số nguyên).
- y: Tọa độ dọc của tàu (kiểu số nguyên).
- speed: Tốc độ di chuyển của tàu (kiểu số nguyên).
- health: Mức máu còn lại của tàu, mặc định là 5 đơn vị (kiểu số nguyên).

2. Lớp dữ liệu Missile (Tên lửa)

Lớp Missile đại diện cho các tên lửa được bắn ra từ tàu khi người chơi nhấn phím SPACE. Tên lửa có khả năng phá hủy thiên thạch khi va chạm. Các thuộc tính chính gồm:

- image: Hình ảnh đại diện của tên lửa.
- x: Vị trí ngang ban đầu của tên lửa tại thời điểm bắn ra.
- y: Vị trí dọc ban đầu của tên lửa.
- dy: Tốc độ di chuyển theo trục dọc, thường là giá trị âm để tên lửa bay lên phía trên.

3. Lớp dữ liệu Asteroid (Thiên thạch)

Thiên thạch là những chướng ngại vật trong trò chơi, rơi ngẫu nhiên từ phía trên màn hình xuống phía dưới. Nếu thiên thạch va chạm với tàu, người chơi sẽ bị trừ máu. Nếu bị bắn trúng, thiên thạch sẽ biến mất và tạo ra hiệu ứng nổ. Các thuộc tính của lớp Asteroid bao gồm:

- image: Hình ảnh của thiên thạch.
- x: Tọa độ ngang ngẫu nhiên của thiên thạch trên màn hình.

- y: Tọa độ dọc ban đầu, thường bắt đầu từ vị trí âm (trên màn hình).
- dy: Tốc độ rơi xuống, là một số thực (float), được tạo ngẫu nhiên trong khoảng từ 1 đến 4

4. Lớp dữ liệu Explosion (Hiệu ứng nổ)

Lớp Explosion đại diện cho hiệu ứng nổ ngắn khi một thiên thạch bị tên lửa phá hủy. Mục đích của lớp này là tăng tính sinh động cho trò chơi và cung cấp phản hồi trực quan cho người chơi. Các thuộc tính bao gồm:

- image: Hình ảnh hiệu ứng nổ.
- x: Tọa độ ngang nơi vụ nổ diễn ra.
- y: Tọa độ dọc nơi vụ nổ diễn ra.
- timer: Thời gian tồn tại của hiệu ứng nổ, thường là 20 khung hình (frames).

5. Dữ liệu phụ trợ

Bên cạnh các lớp chính, chương trình còn sử dụng một số biến phụ trợ để quản lý trạng thái trò chơi và hiển thị thông tin:

- score: Biến số nguyên lưu trữ điểm của người chơi, tăng lên mỗi khi bắn trúng thiên thạch.
- missiles: Danh sách các tên lửa đang tồn tại trên màn hình.
- asteroids: Danh sách các thiên thạch đang rơi.
- explosions: Danh sách các hiệu ứng nổ đang hiển thị.
- font: Đối tượng phông chữ dùng để hiển thị điểm số, số máu còn lại và các thông báo khác trên giao diện.

6. Tài nguyên âm thanh và hình ảnh

Mặc dù không trực tiếp tham gia vào xử lý logic của trò chơi, nhưng dữ liệu hình ảnh và âm thanh đóng vai trò quan trọng trong việc nâng cao trải nghiệm của người chơi. Các tài nguyên này được lưu trữ trong thư mục assets/ và bao gồm:

ship.png hình ảnh ,ai trò: hình ảnh đại diện cho tàu

missile.png loại: hình ảnh vai trò: hình ảnh của tên lửa

asteroid.png loại: hình ảnh vai trò: hình ảnh thiên thạch

explosion.png loại: hình ảnh vai trò: hình ảnh vụ nổ

background.mp3 loại: nhạc nền vai trò: nhạc nền được phát xuyên suốt game

shoot.wav loại: âm thanh vai trò: âm thanh khi bắn tên lửa

boom.wav loại: âm thanh vai trò: âm thanh khi thiên thạch bị phá hủy

3.4. Chương trình

Chương trình được viết bằng ngôn ngữ Python, sử dụng thư viện **Pygame** để xử lý đồ họa, âm thanh và các sự kiện bàn phím. Dưới đây là phần trình bày chi tiết các hàm trong chương trình chính:

1. show_start_screen()

- **Chức năng:** Hiển thị màn hình bắt đầu trò chơi với thông báo "Bấm phím bất kỳ để bắt đầu".
- **Hoạt động:**
  - Xóa màn hình và vẽ nền màu tối.
  - Hiển thị thông báo ở giữa màn hình.
  - Chờ người chơi nhấn bất kỳ phím nào để bắt đầu trò chơi.

2. game_loop()

- **Chức năng:** Hàm chính xử lý toàn bộ vòng lặp của trò chơi.
- **Hoạt động chính gồm:**
  - Tạo đối tượng tàu (Ship), thiên thạch (Asteroid), tên lửa (Missile), hiệu ứng nổ (Explosion).
  - Kiểm tra và xử lý các sự kiện như nhấn phím, thoát game, bắn tên lửa.
  - Cập nhật vị trí tàu, tên lửa và thiên thạch.
  - Kiểm tra va chạm giữa tên lửa và thiên thạch hoặc thiên thạch và tàu.
  - Phát âm thanh và hiển thị hiệu ứng nổ khi có va chạm.
  - Hiển thị điểm số, máu (health) và thông báo khi thắng hoặc thua game.
  - Quay lại đầu game nếu người chơi chọn chơi lại (ENTER) hoặc thoát khỏi trò chơi (ESC).

3. Lớp Ship

- **Chức năng:** Đại diện cho tàu của người chơi.
- **Các phương thức:**
  - __init__: Khởi tạo vị trí, hình ảnh, tốc độ và máu của tàu.
  - draw(): Vẽ hình ảnh tàu lên màn hình.
  - update(keys): Cập nhật vị trí tàu dựa trên phím mũi tên nhấn.

4. Lớp Missile

- **Chức năng:** Đại diện cho tên lửa được bắn ra từ tàu.
- **Các phương thức:**
  - __init__(x, y): Khởi tạo vị trí tên lửa tại vị trí tàu.
  - update(): Di chuyển tên lửa theo trục Y (lên trên).
  - draw(): Vẽ tên lửa lên màn hình.

5. Lớp Asteroid

- **Chức năng:** Đại diện cho các thiên thạch rơi xuống.
- **Các phương thức:**
  - __init__(): Khởi tạo vị trí ngẫu nhiên và tốc độ rơi.
  - update(): Di chuyển thiên thạch theo trục Y. Tự động tạo lại nếu rơi khỏi màn hình.
  - draw(): Vẽ thiên thạch lên màn hình.

6. Lớp Explosion

- **Chức năng:** Hiệu ứng nổ khi tên lửa bắn trúng thiên thạch.
- **Các phương thức:**
  - __init__(x, y): Xác định vị trí nổ và thời gian tồn tại hiệu ứng.
  - update(): Giảm thời gian tồn tại mỗi khung hình.

- draw(): Hiển thị hiệu ứng nổ tại vị trí va chạm.

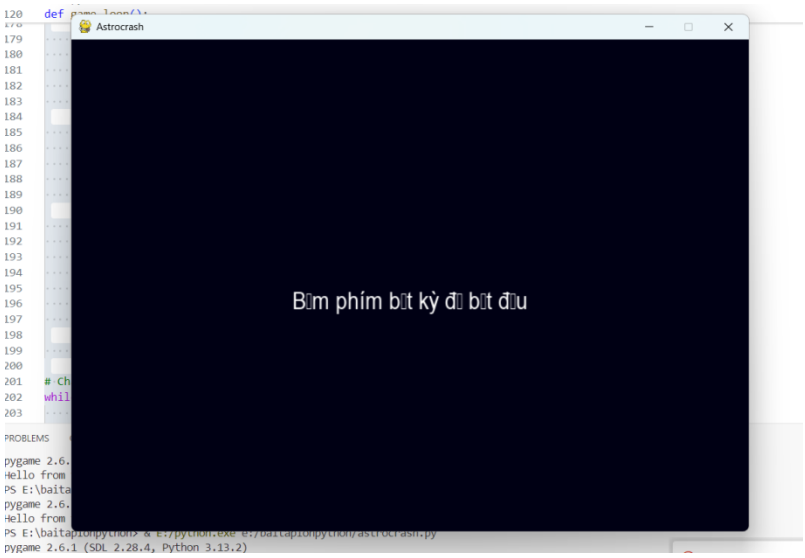
Tóm tắt chương

Trong chương này em trình bày tổng quan chức năng và cấu trúc của game ***Astrocrash****. Trò chơi sử dụng Python và thư viện Pygame để xây dựng một hệ thống điều khiển tàu vũ trụ, bắn tên lửa, tránh thiên thạch và tạo hiệu ứng nổ. Các thuật toán chính bao gồm xử lý sự kiện người chơi, cập nhật vị trí đối tượng, kiểm tra va chạm và hiển thị giao diện. Dữ liệu được tổ chức hướng đối tượng qua các lớp như Ship, Missile, Asteroid, và Explosion, kết hợp với tài nguyên âm thanh, hình ảnh để nâng cao trải nghiệm người chơi. Game hoạt động trong vòng lặp chính với logic rõ ràng, mạch lạc và dễ mở rộng.*

CHƯƠNG VI: THỰC NGHIỆM VÀ KẾT LUẬN

4.1. Thực nghiệm

- Chạy và ghi lại kết quả các bài test các tính năng của sản phẩm, đánh giá chất lượng



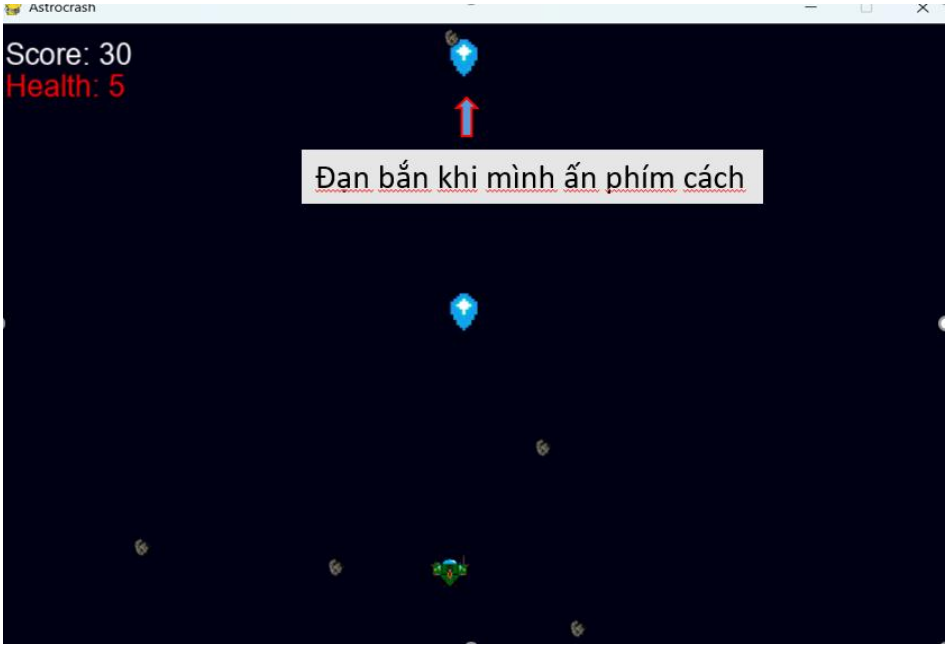
Hình 4.1 : Giao diện khởi động game

Mới khi chạy code thì hiện ra giao diện và có tính năng nhấp phím bất kỳ từ bàn phím thì bắt đầu chơi.

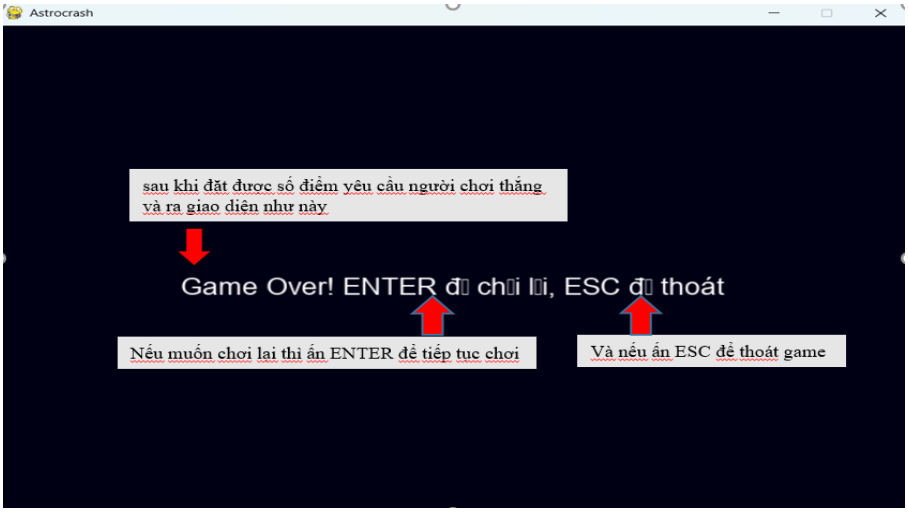
Hình : 4.2 : Điểm và máu

-Hình ảnh test

Hình 4.3 : Test khi tàu bị thiên thạch rơi vào mất máu và bắn thiên thạch được điểm



Hình 4.4 : Tàu bắn đạn



Hình 4.5 : Chiến thắng và muốn chơi lại hay thoát

4.2. Kết luận

- **Sản phẩm đã làm được những gì**:

  Qua quá trình thực hiện, sản phẩm đã hoàn thành đúng theo yêu cầu đề ra. Cụ thể, hệ thống đã thực hiện được các chức năng chính như [liệt kê các chức năng tiêu biểu – ví dụ: đăng nhập, quản lý người dùng, ghi log tập trung, phân tích truy cập

trái phép...]. Giao diện được xây dựng đơn giản, dễ sử dụng, đảm bảo tính trực quan và thân thiện với người dùng. Ngoài ra, sản phẩm cũng đảm bảo hoạt động ổn định trong môi trường kiểm thử thực tế.

- **Học được gì**:

  Trong quá trình thực hiện đề tài, em đã nâng cao được nhiều kiến thức và kỹ năng, bao gồm:

  - Hiểu sâu hơn về [nêu công nghệ chính: ví dụ Python, Flask, SQL Server, Pygame, Windows Server, Splunk...].
  - Cải thiện kỹ năng lập trình, thiết kế hệ thống, xử lý dữ liệu và giao tiếp giữa các thành phần trong phần mềm.
  - Nâng cao khả năng làm việc nhóm, quản lý thời gian và giải quyết vấn đề thực tiễn.

- **Sẽ cải tiến gì**:

  Mặc dù sản phẩm đã đáp ứng các yêu cầu cơ bản, tuy nhiên vẫn còn một số điểm có thể cải tiến để nâng cao hiệu quả và trải nghiệm người dùng:

  - Tối ưu hóa hiệu suất hệ thống khi xử lý dữ liệu lớn.
  - Bổ sung thêm các chức năng nâng cao như [liệt kê ví dụ: thống kê, tìm kiếm nâng cao, phân quyền chi tiết, giao diện thân thiện hơn...].
  - Cải thiện bảo mật hệ thống nhằm ngăn chặn các truy cập trái phép hoặc lỗ hổng tiềm ẩn.

TÀI LIỆU THAM KHẢO

1. Nguyễn Văn Hiển (2020), *Giáo trình Python cơ bản*, NXB Bách Khoa Hà Nội.
2. Sweigart, A. (2015), *Making Games with Python & Pygame*, Invent with Python. Truy cập tại: https://inventwithpython.com/pygame/
3. Pygame Documentation (2023), *Pygame 2.0 Documentation*, Pygame.org. Truy cập tại: https://www.pygame.org/docs/
4. Lutz, M. (2013), *Learning Python*, 5th Edition, O'Reilly Media.
5. Downey, A. (2015), *Think Python: How to Think Like a Computer Scientist*, 2nd Edition, O'Reilly Media.

link youtube : https://youtu.be/2lPvbxkOXA8?si=SiuJ3GxqRAtVPsWF

Link github : https://github.com/dinhtu1102/baitaplon-python/upload