**ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN**

**MÔN NGÔN NGỮ LẬP TRÌNH PYTHON**

**Đề tai: Xây dựng trò chơi bắn gà trong không gian**

**Thành viên nhóm:**

**Nguyễn Tuấn Đạt (Nhóm trưởng) 3123410070**

**Nguyễn Tuấn Tài 3123410318**

**Lưu Cơ Thành 3123410334**

**Dư Nguyễn Đăng Khoa 3122410183**

Giảng viên hướng dẫn: Ths. Lê Tấn Long

**Thành phố Hồ Chí Minh - Tháng 10/2024**

**MỞ ĐẦU**

Trong thời đại công nghệ số phát triển mạnh mẽ, lập trình game không chỉ là một lĩnh vực giải trí mà còn là một công cụ giúp rèn luyện tư duy logic, khả năng sáng tạo và kỹ năng lập trình. Để áp dụng kiến thức đã học vào thực tiễn, nhóm chúng em đã lựa chọn phát triển một trò chơi bắn súng đơn giản mang tên **"Bắn Gà Không Gian"**, được xây dựng bằng **Python** và thư viện **Pygame**.

Mục tiêu của trò chơi là điều khiển một con tàu vũ trụ di chuyển trong không gian, bắn hạ các kẻ địch xuất hiện ngẫu nhiên và ghi điểm. Trò chơi kết hợp nhiều yếu tố lập trình quan trọng như xử lý đồ họa, điều khiển nhân vật, xử lý va chạm, âm thanh và tối ưu hiệu suất.

Trong báo cáo này, chúng em sẽ trình bày chi tiết về quá trình phát triển trò chơi, từ ý tưởng ban đầu, phân tích thiết kế, triển khai mã nguồn cho đến những khó khăn gặp phải và hướng giải quyết. Chúng em hy vọng rằng thông qua dự án này, không chỉ cải thiện kỹ năng lập trình mà còn góp phần nâng cao hiểu biết về cách xây dựng một trò chơi điện tử đơn giản nhưng hấp dẫn.

**LỜI CẢM ƠN**

Chúng em xin chân thành cảm ơn Khoa Công nghệ thông tin, trường Đại học Sài Gòn đã tạo điều kiện cho chúng em được thực hiện đồ án môn học "Ngôn ngữ lập trình Python". Chúng em cũng xin chân thành cảm ơn thầy Lê Tấn Long, thầy đã giảng dạy cho chúng em những kiến thức cần thiết cho môn học này. Những kiến thức đó đã giúp cho chúng em rất nhiều trong quá trình làm đồ án báo cáo môn học. Chúng em xin chân thành cám ơn quý Thầy cô trong Khoa đã tận tình giảng dạy và trang bị cho chúng em những kiến thức cần thiết trong thời gian qua. Mặc dù nhóm đã cố gắng hoàn thành đồ án môn học với tất cả nổ lực của từng thành viên trong nhóm,nhưng đồ án chắc chắn không tránh khỏi nhữmg thiếu sót nhất định, rất mong nhận được sự cảm thông, chia sẻ và tận tình đóng góp chỉ bảo của quý Thầy Cô.Chúng em xin chân thành cảm ơn thầy

# 

# **NHẬN XÉT**

**( Của giảng viên hướng dẫn )**

………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….

………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..………………………………………………………………………………………………………………………………………………………………………………………………………..…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….

# 

# **CHƯƠNG 1: TỔNG QUAN**

* 1. Mục đích

Đồ án này được triển khai với mục tiêu tạo ra một trò chơi bắn súng không gian có lối chơi hấp dẫn và thú vị, lấy cảm hứng từ tựa game kinh điển Chicken Invaders. Mục đích của nhóm là xây dựng một sản phẩm giải trí chất lượng, phù hợp với mọi lứa tuổi, đồng thời áp dụng các kiến thức về lập trình game, đồ họa, và xử lý sự kiện đã học trong chương trình Lập trình Python.

1.2 Phân tích công trình có liên quan

Chicken Invaders hay được gọi với cái tên dân dã là Bắn Gà là một series trò chơi điện tử được phát triển bởi các studio InterAction.

Trò chơi đã được phát hành cho các nền tảng Microsoft Windows, OS X, Linux, iOS, Windows Phone và Android.

Có tổng cộng 6 phần game đã được phát hành trong series này:

* Chicken Invader được phát hành vào năm 1999
* Chicken Invaders: The Next Wave năm 2002
* Chicken Invaders : Revenge of the Yolk năm 2006
* Chicken Invaders: Ultimate Omeltte năm 2010
* Chicken Invaders: Cluck of the Dark Side năm 2014
* Chicken Invaders Universe năm 2018 cũng là phần mới nhất cho đến hiện tại

Trò chơi sẽ có lối chơi đơn giản, nơi người chơi điều khiển phi thuyền để tiêu diệt kẻ thù bay ngang qua màn hình. Các màn chơi được thiết kế đa dạng, độ khó tăng dần, và có cả những trận đánh trùm đầy thử thách. Hệ thống vũ khí nâng cấp sẽ giúp người chơi cảm thấy hứng thú hơn khi thu thập và cải tiến vũ khí để đối mặt với kẻ thù ngày càng mạnh.

1.3 Phương pháp thực hiện và phân tích đề tài

* Khảo sát các nghiệp vụ chính trong công tác xây dựng game.
* Nghiên cứu các công nghệ để áp dụng.
* Thiết kế và xây dựng ứng dụng game.
* Phát triển ứng dụng game.
* Triển khai và cài đặt ứng dụng và hệ thống.

# **CHƯƠNG 2: NGÔN NGỮ LẬP TRÌNH PYTHON**

2.1 Ngôn ngữ lập trình Python là gì?

# - Python là một ngôn ngữ lập trình bậc cao, được tạo ra bởi Guido van Rossum vào những năm 1980 và phát triển tiếp theo trong thập niên 1990.

# - Python là một ngôn ngữ lập trình đơn giản, dễ học và dễ sử dụng, có thể được sử dụng để xây dựng các ứng dụng máy tính, trang web, các hệ thống đám mây, trí tuệ nhân tạo, máy học và các ứng dụng phân tích dữ liệu. Python cũng là một ngôn ngữ lập trình đa nền tảng, có thể chạy trên các hệ điều hành khác nhau như Windows, Linux và MacOS.

# - Python có một cộng đồng lớn và đa dạng, với hàng ngàn thư viện và framework hỗ trợ cho việc phát triển ứng dụng

2.2 Ưu và nhược điểm của Python

a) Ưu điểm của Python

* **Cú pháp đơn giản, dễ học**: Dễ đọc, viết, giúp lập trình viên tập trung vào giải pháp thay vì cú pháp.
* **Miễn phí, mã nguồn mở**: Hoàn toàn miễn phí, có thể chỉnh sửa, phát triển và phân phối thoải mái.
* **Khả năng di chuyển**: Chạy trên nhiều nền tảng mà không cần chỉnh sửa code.
* **Mở rộng và nhúng**: Hỗ trợ nhúng C, C++ để tăng hiệu suất.
* **Thông dịch cấp cao**: Không cần quản lý bộ nhớ, tự động chuyển đổi sang ngôn ngữ máy.
* **Thư viện phong phú**: Hỗ trợ nhiều thư viện, giảm thời gian viết code.
* **Hướng đối tượng**: Dễ tổ chức và tái sử dụng code.
* **Tính linh hoạt cao**: Có thể dùng cho nhiều lĩnh vực như web, AI, dữ liệu, game, tự động hóa.
* **Cộng đồng hỗ trợ lớn**: Luôn có tài liệu, diễn đàn, khóa học hỗ trợ người dùng.
* **Khả năng tương thích cao**: Tích hợp tốt với nhiều công nghệ và hệ thống khác nhau.

b) Nhược điểm của Python

* **Tốc độ chậm**: Chạy chậm hơn so với các ngôn ngữ biên dịch như C++, Java.
* **Không phù hợp cho lập trình di động**: Ít được sử dụng để phát triển ứng dụng di động.
* **Quản lý bộ nhớ kém hơn C/C++**: Tiêu tốn bộ nhớ nhiều hơn do sử dụng cơ chế quản lý tự động.
* **Không chặt chẽ như Java**: Dễ viết code lộn xộn, khó bảo trì với các dự án lớn.
* **Bảo mật không cao**: Là mã nguồn mở nên dễ bị tấn công nếu không bảo vệ tốt.
* **Không tối ưu cho đa luồng**: Trình thông dịch GIL hạn chế hiệu suất của ứng dụng đa luồng.
* **Thư viện không tối ưu**: Một số thư viện tiêu tốn tài nguyên, ảnh hưởng hiệu suất.
* **Gặp lỗi trên một số nền tảng**: Chạy tốt trên Windows, Linux nhưng có thể gặp lỗi ở hệ thống khác.
* **Cần hiểu biết về C/C++ để tối ưu**: Khi cần tăng hiệu suất, phải kết hợp với C/C++.

2.3 Ứng dụng của Python

- **Phát triển ứng dụng web**: Python có thể được sử dụng để phát triển các ứng dụng web, với các framework như Django, Flask và Pyramid

- **Trí tuệ nhân tạo và máy học**: Python là một trong những ngôn ngữ phổ biến nhất trong lĩnh vực trí tuệ nhân tạo và máy học, với các thư viện như TensorFlow, Keras và PyTorch.

- **Phân tích dữ liệu**: Python có các thư viện như Pandas và NumPy để xử lý và phân tích dữ liệu, và Matplotlib và Seaborn để tạo đồ thị và biểu đồ.

- **Game development**: Python có thể được sử dụng để phát triển game với các thư viện như Pygame và PyOpenGL.

- **Đám mây**: Python có thể được sử dụng để quản lý và triển khai các hệ thống đám mây với các framework như OpenStack và Ansible.

- **Hệ thống nhúng**: Python có thể được sử dụng để phát triển các hệ thống nhúng, với các framework như MicroPython và CircuitPython.

- **Ứng dụng máy tính**: Python có thể được sử dụng để phát triển các ứng dụng máy tính trên nhiều nền tảng khác nhau, với các thư viện như PyQt và wxPython

# **CHƯƠNG 3: KHẢO SÁT VÀ PHÂN TÍCH YÊU CẦU**

## 3.1 Mục tiêu và phạm vi của dự án

* Trước khi phát triển game Chicken Invaders bằng Python, cần xác định rõ mục tiêu và phạm vi của dự án để đảm bảo quá trình thiết kế và lập trình đi đúng hướng.
  + 1. Mục tiêu của dự án
* Dự án nhằm xây dựng một trò chơi bắn súng không gian dựa trên **Chicken Invaders**, với các mục tiêu cụ thể như sau:

**+ Xây dựng game hoàn chỉnh**: Thiết kế một trò chơi có thể chơi được với đầy đủ chức năng cơ bản.

**+ Áp dụng kiến thức lập trình**: Sử dụng Python và thư viện Pygame để lập trình game.

**+ Phát triển kỹ năng làm game**: Hiểu rõ quy trình phát triển game từ thiết kế, lập trình đến tối ưu hiệu suất.

**+ Mang lại trải nghiệm giải trí**: Game có lối chơi thú vị, hấp dẫn và thử thách tăng dần.

* + 1. Phạm vi của dự án
* Do giới hạn thời gian và nguồn lực, dự án tập trung vào:

+ Cốt lõi gameplay: Chỉ phát triển chế độ chơi đơn với các cấp độ khó tăng dần.

+ Hệ thống kẻ địch đơn giản: Gà sẽ di chuyển theo các mô hình cố định, chưa có AI nâng cao.

+ Vũ khí và nâng cấp: Cung cấp một số loại đạn cơ bản, chưa phát triển hệ thống nâng cấp phức tạp.

+ Hiệu ứng âm thanh và đồ họa: Sử dụng các hình ảnh và âm thanh có sẵn thay vì tự thiết kế hoàn toàn.

+ Không có chế độ nhiều người chơi: Chỉ tập trung vào trải nghiệm chơi đơn.

## Phân tích yêu cầu của hệ thống

### Yêu cầu chức năng

* Điều khiển nhân vật: Người chơi có thể di chuyển tàu bằng các phím mũi tên < và > và bắn bằng phím Space.
* Hệ thống kẻ địch: Đàn gà di chuyển từ trên xuống với nhiều kiểu tấn công khác nhau.
* Vũ khí và vật phẩm: Người chơi có thể thu thập khiên bảo vệ, mạng sống để hỗ trợ trong quá trình chơi.
* Hệ thống điểm số: Hiển thị điểm số của người chơi, điểm càng cao càng chứng tỏ khả năng tốt .
* Cấp độ khó tăng dần: Gà sẽ xuất hiện ngày càng nhiều và mạnh hơn theo thời gian.
* Âm thanh và hiệu ứng: Khi bắn súng, tiêu diệt kẻ địch hoặc va chạm, sẽ có âm thanh và hiệu ứng đi kèm.
* Màn hình Game Over: Khi người chơi hết mạng, hiển thị thông báo và điểm số
  + 1. Yêu cầu phi chức năng
* Hiệu suất ổn định: Game cần chạy mượt mà trên các hệ thống có cấu hình trung bình.
* Dễ mở rộng: Dễ dàng thêm boss, vũ khí mới, cấp độ mới trong tương lai.
* Giao diện thân thiện: Thiết kế đơn giản, dễ nhìn, dễ sử dụng.

# **CHƯƠNG 4: PHÂN TÍCH VÀ THIẾT KẾ**

## 4.1 Kiến trúc tổng thể của hệ thống

- Hệ thống game **Chicken Invaders** được xây dựng theo mô hình **lập trình hướng đối tượng (OOP)**, giúp chia trò chơi thành nhiều thành phần riêng biệt, dễ quản lý và mở rộng.

- Thành phần chính của game bao gốm như sau:

**+ Vòng lặp game (Game Loop)**: Kiểm soát toàn bộ trò chơi, cập nhật trạng thái và hiển thị hình ảnh.

**+ Người chơi (Player)**: Điều khiển tàu vũ trụ, di chuyển và bắn kẻ địch.

**+ Kẻ địch (Enemy)**: Các đàn gà xuất hiện ngẫu nhiên và có hành vi tấn công khác nhau.

**+ Đạn (Bullet)**: Quản lý việc bắn đạn và xử lý va chạm với kẻ địch.

**+ Hệ thống điểm số (Score System)**: Cập nhật điểm số khi tiêu diệt kẻ địch.

**+ Giao diện người dùng (UI)**: Hiển thị thông tin như điểm số, thanh máu, màn hình Game Over.

## 4.2 Mô hình dữ liệu và các lớp trong game

- Game được thiết kế theo mô hình hướng đối tượng với các lớp chính như sau:

### 4.2.1 Lớp Người chơi ( Player )

- Thuộc tính:

+ x : Tọa độ ngang (hoành độ) của tàu vũ trụ trên màn hình.

+ y : Tọa độ dọc (tung độ) của tàu vũ trụ trên màn hình.

+ speed : Tốc độ di chuyển của tàu, xác định quãng đường tàu dịch chuyển sau mỗi khung hình (frame).

+ lives : Số mạng sống của người chơi. Khi bị kẻ địch tấn công, số mạng giảm. Khi lives = 0, trò chơi kết thúc.

- Phương thức:

+ move\_left(): Di chuyển tàu sang bên trái theo giá trị speed, đồng thời kiểm tra không cho phép tàu đi ra ngoài màn hình.

+ move\_right(): Di chuyển tàu sang bên phải theo giá trị speed, tương tự đảm bảo tàu vẫn nằm trong màn hình.

+ shoot(): Tạo ra một đối tượng Bullet mới tại vị trí hiện tại của tàu để bắn về phía trước (lên trên).

### 4.2.2 Lớp Kẻ địch ( Enemy )

- Thuộc tính:

+ x: Tọa độ ngang của kẻ địch trên màn hình.

+ y: Tọa độ dọc của kẻ địch.

+ speed: Tốc độ di chuyển của kẻ địch, thường di chuyển theo hướng từ trên xuống dưới

+ health: Lượng máu còn lại của kẻ địch. Khi bị trúng đạn, máu giảm. Khi máu giảm về 0, kẻ địch bị tiêu diệt.

- Phương thức:

+ move(): Thực hiện việc di chuyển của kẻ địch theo chiều dọc (hướng xuống dưới). Có thể thêm dao động ngang nhẹ để tăng độ khó.

+ attack\_player(): Kiểm tra va chạm giữa kẻ địch và tàu vũ trụ. Nếu va chạm xảy ra, giảm số mạng (lives) của người chơi, đồng thời loại bỏ kẻ địch.

### 4.2.3 Lớp Đạn ( Bullet )

- Thuộc tính:  
 + x: Tọa độ ngang của viên đạn.

+ y: Tọa độ dọc của viên đạn.

+ speed: Tốc độ di chuyển của đạn, thường cao hơn tốc độ di chuyển của tàu hoặc kẻ địch để đảm bảo kịp tiêu diệt mục tiêu.

- Phương thức:

+ move(): Di chuyển đạn theo chiều dọc hướng lên trên màn hình (giảm giá trị y theo mỗi khung hình).

+ check\_collision(enemy): Kiểm tra va chạm giữa đạn và một kẻ địch cụ thể. Nếu va chạm xảy ra, giảm máu (health) của kẻ địch và hủy đối tượng đạn.

### 4.2.4 Lớp Trò chơi ( Game )

- Thuộc tính:

+ score: Tổng số điểm mà người chơi đã tích lũy được trong suốt quá trình chơi. Điểm số tăng lên mỗi khi tiêu diệt kẻ địch thành công.

+ level: Cấp độ hiện tại của trò chơi. Sau mỗi một ngưỡng điểm số đạt được, cấp độ tăng lên, dẫn đến việc kẻ địch di chuyển nhanh hơn hoặc xuất hiện nhiều hơn.

- Phương thức:

+ update(): Cập nhật trạng thái toàn bộ trò chơi mỗi khung hình, bao gồm:

* Di chuyển tàu, kẻ địch, đạn.
* Kiểm tra va chạm giữa đạn và kẻ địch.
* Kiểm tra va chạm giữa kẻ địch và tàu.
* Tạo thêm kẻ địch mới nếu cần thiết.
* Cập nhật điểm số và cấp độ.

+ render(screen): Vẽ toàn bộ các đối tượng (tàu, đạn, kẻ địch, điểm số, mạng sống,...) lên màn hình.

+ handle\_events(): Xử lý tất cả các sự kiện tương tác của người dùng như:

* + Nhấn phím di chuyển tàu.
  + Bấm phím bắn đạn.
  + Thoát trò chơi

### 4.2.5 Mở rộng

- Tương tác giữa các lớp:

+ Player sẽ sử dụng shoot() để tạo ra các đối tượng Bullet.

+ Bullet sẽ tương tác với Enemy thông qua colliderect().

+ Game sẽ là lớp trung tâm, gọi đến các phương thức cập nhật (update) và vẽ (render) của từng đối tượng.

+ Có thể bổ sung thêm các tính năng như: đạn đôi, nâng cấp vũ khí, nhiều loại kẻ địch khác nhau, thêm boss cuối mỗi màn,...

## 4.3 Màn hình giao diện của trò chơi ( Background )

## 



* Khai báo thư viện:

+ pygame: thư viện hỗ trợ lập trình game 2D.

+ random: dùng để sinh số ngẫu nhiên.

+ math: hỗ trợ các phép toán.

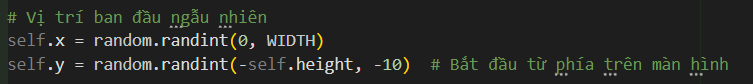
+ os: xử lý đường dẫn file.



* Khai báo lớp Planet đại diện cho hành tinh bay trong nền.
* Nhận vào 1 hình ảnh img là hình hành tinh.



* Lưu hình ảnh và lấy kích thước cho ảnh ( rộng, cao ).



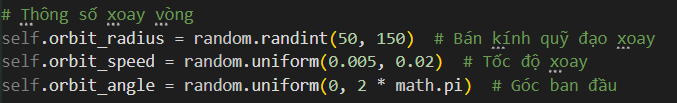
* Vị trí ban đầu:

+ x: ngẫu nhiên ngang màn hình.

+ y: nằm trên màn hình một chút.



* Hành tinh rơi xuống với tốc độ chậm trong khoảng 0.2 – 0.8

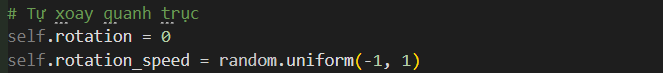


* Cách hành tinh xoay quanh một tâm:

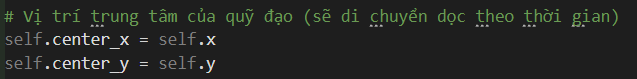
+ orbit\_radius: bán kính quỹ đạo ( 50 – 150 ).

+ orbit\_speed: tốc độ quay ( 0.005 – 0.02 ).

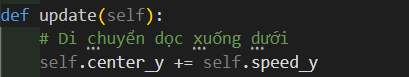
+ orbit\_angle: góc hiện tại trên quỹ đạo ( 0 – 2pi ).



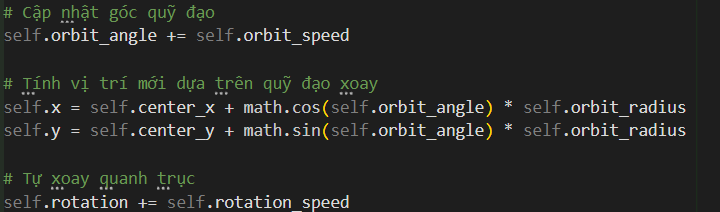
* Self.rotation = 0, xác định góc quay ban đầu.
* Self.rotation\_speed: tốc độ quay ngẫu nhiên từ -1 đến 1.
* Làm cho hành tinh quay theo tốc độ và hướng ngẫu nhiên



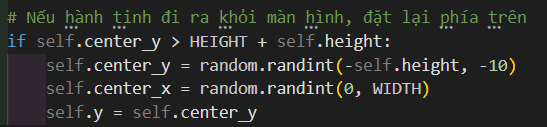
* center\_x và center\_y là tọa độ tâm quỹ đạo.



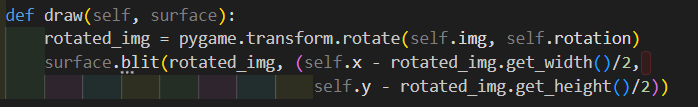
* Phương thức update – cập nhật chuyển động.
* Tâm quỹ đạo trôi xuống theo tốc độ speed\_y



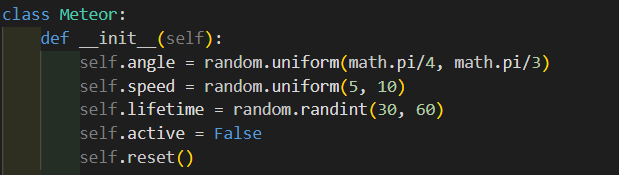
* Di chuyển theo một quỹ đạo tròn quanh một điểm cố định với tốc độ quay self.orbit\_speed và bán kính self.orbit\_radius
* Chuyển động quay tự thân với tốc độ self.rotation\_speed, làm cho nó xoay quanh chính mình trong quá trình di chuyển



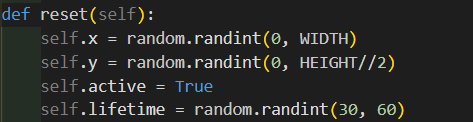
* Khi self.center\_y vượt qua dưới đáy màn hình (HEIGHT + self.height), ta biết rằng hành tinh đã đi ra ngoài màn hình.
* Sau đó, hành tinh được đặt lại ở một vị trí ngẫu nhiên phía trên màn hình và ở một vị trí ngẫu nhiên theo chiều ngang



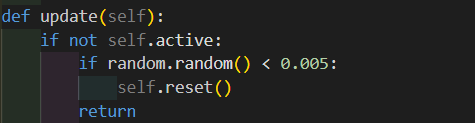
* Phương thức draw – vẽ hành tinh
* Xoay hình ảnh gốc (self.img) theo góc self.rotation để tạo ra hình ảnh đã xoay (rotated\_img)
* Sau đó vẽ hình ảnh đã xoay lên bề mặt surface tại vị trí (self.x, self.y)



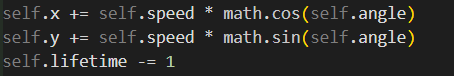
* Lớp Meteor – tạo sao băng
* angle: góc bay chéo chéo. ( pi/4 đến pi/3 khoảng 45 đến 60°)
* speed: tốc độ nhanh ( 5 – 10 ).
* lifetime: thời gian tồn tại ( 30 – 60 ).
* active: sao băng đang bay hay không.
* reset(): khởi tạo vị trí mới.



* Phương thức reset dùng để random vị trí và thời gian tồn tại mới.
* Đặt lại vị trí của sao băng tại một điểm ngẫu nhiên trên màn hình (theo chiều X và Y).
* Kích hoạt sao băng (set active = True).
* Đặt lại thời gian sống của sao băng thành một giá trị ngẫu nhiên trong khoảng từ 30 - 60.



* Phương thức update – thực hiện công việc cập nhật trạng thái của sao băng nếu sao băng đang không hoạt động
* Khi sao băng không hoạt động, có một xác suất 0.5% mỗi lần phương thức update được gọi, sao băng sẽ được tái tạo lại thông qua reset().

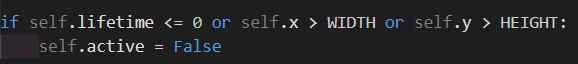


* Sao băng di chuyển theo hướng và tốc độ đã được xác định bởi self.angle và self.speed. Tọa độ (self.x, self.y) của sao băng được cập nhật theo công thức:

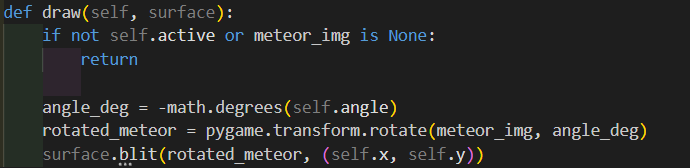
+ self.x += self.speed \* math.cos(self.angle) (di chuyển theo chiều ngang).

+ self.y += self.speed \* math.sin(self.angle) (di chuyển theo chiều dọc).

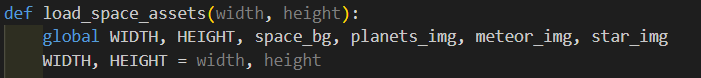
* Giảm thời gian sống của sao băng, điều này giúp xác định sao băng sẽ tồn tại bao lâu trước khi biến mất.



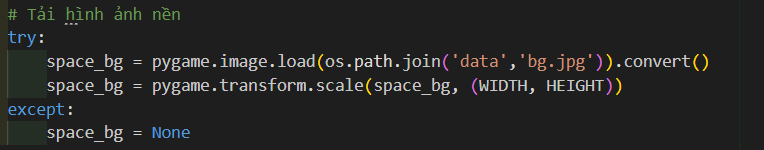
* Khi thời gian sống của sao băng kết thúc hoặc khi sao băng ra ngoài màn hình (vượt quá chiều rộng hoặc chiều cao màn hình), sao băng sẽ bị hủy kích hoạt.



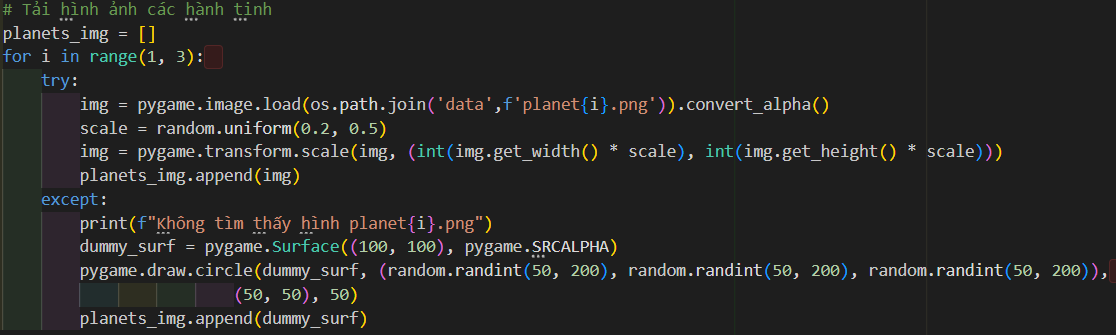
* Phương thức draw để vẽ sao băng
* self.active kiểm tra có hoạt động hay không, nếu = False, tức bị tắt, không vẽ.
* meteor\_img is None: kiểm tra hình ảnh có hợp lệ không, nếu không thì không vẽ.
* return: Nếu một trong hai điều kiện trên là đúng thì không vẽ.
* angle\_deg = -math.degrees(self.angle): Chuyển từ radian sang độ và đảo ngược hướng xoay.
* pygame.transform.rotate(meteor\_img, angle\_deg): Xoay hình ảnh thiên thạch (meteor\_img) theo góc angle\_deg. Hình ảnh được xoay sẽ được lưu vào biến rotated\_meteor.



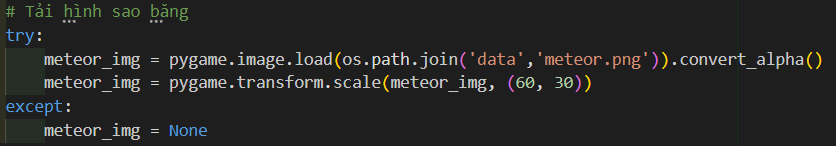
* Hàm load\_space\_assets để tải các tài nguyên đồ họa và thiết lập kích thước màn hình.



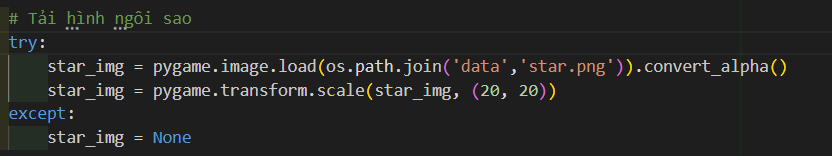
* Tải hình ảnh nền từ một tệp (bg.jpg), chuyển đổi nó để tối ưu hiệu suất và thay đổi kích thước để phù hợp với kích thước cửa sổ trò chơi.
* Nếu không thể tài hình ảnh space\_bg sẽ được đặt thành None để không làm gián đoạn trò chơi.



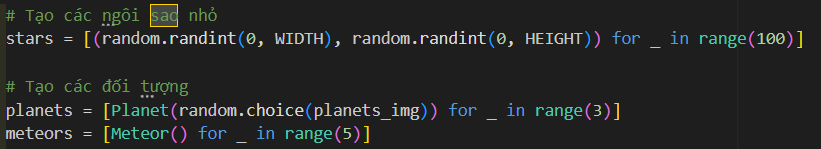
* Tải hình ảnh các hành tinh, thay đổi kích thước ngẫu nhiên và xử lý lỗi khi không thể tải được hình ảnh.
* Nếu không thể tải được hình ảnh từ thư mục data, nó sẽ tạo ra hình ảnh giả (một hình tròn màu sắc ngẫu nhiên) để thay thế, giúp trò chơi không bị gián đoạn.



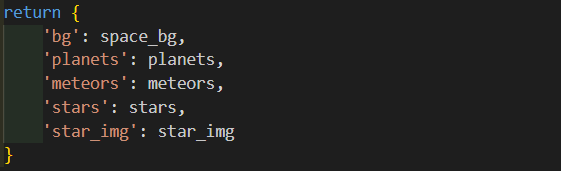
* Tải hình ảnh sao băng từ tệp meteor.png trong data.
* Covert\_alpha(): Chuyển đổi bề mặt nền của sao băng trong suốt.
* Thay đổi kích thước gốc thành 60x30
* Nếu không tải được hình ảnh thì sẽ được gán bằng None



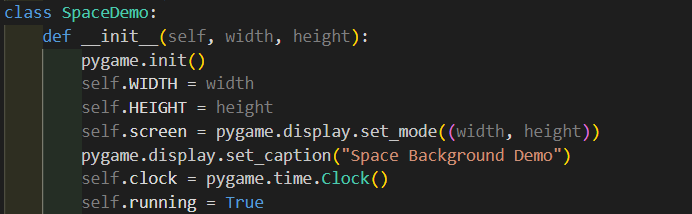
* Tương tự như sao băng, tải lên hình ảnh ngôi sao từ tệp star.png trong data.
* Thay đổi nền trong suốt bằng convert\_alpha().
* Thay đổi kích thước thành 20x20.
* Nếu không tải được hình ảnh thì gán bằng None.



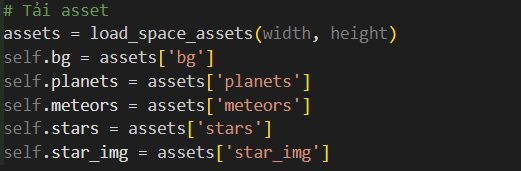
* Tạo ra một danh sách gồm 100 ngôi sao nhỏ với tọa độ ngẫu nhiên trên màn hình.
* Tạo ra 3 hành tinh, mỗi hành tinh có một hình ảnh ngẫu nhiên từ planets\_img.
* Tạo ra 5 sao băng ( meteor ).



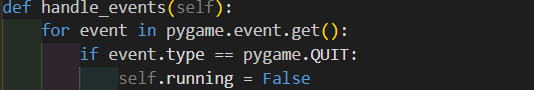
* Trả về các tài nguyên và đối tượng cần thiết trong trò chơi.
* Chứa các tài nguyên như hình nền, hành tinh, sao băng, các ngôi sao nhỏ, và hình ảnh ngôi sao.



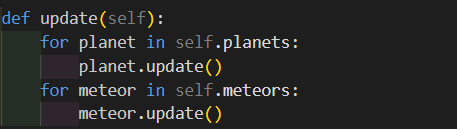
* Lớp SpaceDemo để khởi tạo môi trường cho trò chơi.
* pygame.init(): khởi tạo các mô-đun pygame cần thiết.
* pygame.display.set\_mode((width, height): tạo cửa sổ có kích thước width x height
* pygame.time.Clock() tạo một đối tượng đồng hồ để điều chỉnh tốc độ khung hình (FPS) trong trò chơi.
* self.running = True là điều kiện để vòng lặp chính của trò chơi tiếp tục chạy.



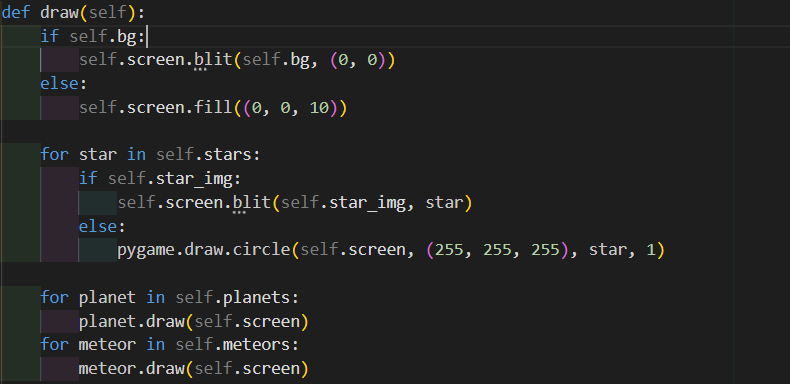
* Tải tất cả tài nguyên không gian từ các tệp tin ảnh.
* self.bg: Lưu trữ nền không gian.
* self.planets: Lưu trữ danh sách các hành tinh.
* self.meteors: Lưu trữ danh sách các sao băng.
* self.stars: Lưu trữ danh sách các vị trí sao nhỏ.
* self.star\_img: Lưu trữ hình ảnh của ngôi sao.



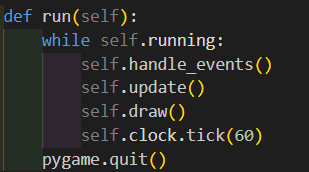
* Kiểm tra sự kiện từ người dùng.
* pygame.event.get(): trả về một danh sách các sự kiện đang chờ xử lý.
* for event in pygame.event.get() lặp qua tất cả các sự kiện đang chờ xử lý.
* if event.type == pygame.QUIT: kiểm tra xem có phải sự kiện là đóng cửa sổ hay không.
* self.running = False sẽ đặt biến self.running thành False, làm cho vòng lặp chính trong phương thức run() kết thúc, và trò chơi sẽ dừng lại.



* for planet in self.planets:: Lặp qua danh sách các hành tinh (self.planets).
* planet.update(): Gọi phương thức update() của đối tượng planet để cập nhật trạng thái của hành tinh.
* for meteor in self.meteors:: Lặp qua danh sách các sao băng (self.meteors).
* meteor.update(): Gọi phương thức update() của đối tượng meteor để cập nhật trạng thái của sao băng.



* if self.bg: kiểm tra xem có nền không. Nếu có, nó sẽ vẽ nền lên màn hình bằng self.screen.blit(self.bg, (0, 0)). Nếu không có nền (else), nó sẽ vẽ màu đen (hoặc bất kỳ) bằng self.screen.fill((0, 0, 10)).
* for star in self.stars: Lặp qua các ngôi sao trong self.stars.
* if self.star\_img: kiểm tra xem có hình ảnh của ngôi sao không. Nếu có, nó sẽ vẽ ngôi sao sử dụng blit, nếu không nó sẽ ngôi sao dưới dạng hình tròn nhỏ màu trắng với bán kính 1.
* for planet in self.planets: Lặp qua các hành tinh trong self.planets và gọi phương thức draw(self.screen) của mỗi hành tinh để vẽ nó lên màn hình.
* for meteor in self.meteor: Lặp qua các sao băng trong self.meteors và gọi phương thức draw(self.screen) của mỗi sao băng để vẽ nó lên màn hình
* pygame.display.flip() sẽ cập nhật cửa sổ trò chơi, hiển thị tất cả các thay đổi và đối tượng mới đã được vẽ lên màn hình.



* while self.running: vòng lặp chính chạy liên tục với self.running là True.
* self.handle\_events(): để kiểm tra và xử lý tất cả các sự kiện.
* self.update(): để cập nhật trạng thái của tất cả các đối tượng trong trò chơi
* self.draw(): Gọi phương thức draw() để vẽ các đối tượng lên màn hình
* self.clock.tick(60): kiểm soát tốc độ của vòng lặp chính và giới hạn số khung hình mỗi giây (FPS) là 60.
* pygame.quit(): Được gọi để thoát khỏi Pygame và dọn dẹp tài nguyên.

## **c) Sơ đồ luồng hoạt động của hệ thống**

- Dưới đây là mô tả luồng hoạt động của game từ lúc bắt đầu đến khi kết thúc:

### c.1) Bắt đầu game

* Hiển thị màn hình chào mừng.
* Người chơi nhấn **Start** để bắt đầu.

### c.2) Chơi game

* Người chơi **điều khiển tàu**, né tránh và **bắn gà**.
* Đàn gà di chuyển và **bắn đạn** **tấn công người chơi**.
* Nếu viên đạn trúng gà, gà sẽ bị tiêu diệt.
* Nếu tàu bị gà bắn trúng đạn , người chơi **mất máu**.

### c.3) Kết thúc game

* Nếu hết mạng, hiển thị màn hình **Game Over** và điểm số.
* Người chơi có thể chọn **chơi lại hoặc thoát game**.

## **d) Thiết kế giao diện người dùng (UI/UX)**

- Giao diện game cần trực quan, dễ điều khiển, gồm:

### d.1) Màn hình chính

* Hiển thị **nút Bắt đầu (Start)** và **nút Thoát (Exit)**.

### d.2) Màn hình chơi game

* Tàu vũ trụ ở dưới cùng, đàn gà ở trên.
* Hiển thị **điểm số**, **số mạng sống**, và **cấp độ**.

### d.3) Màn hình Game Over

* Hiển thị điểm số và nút **Chơi lại hoặc Thoát**.

**Chương IV: Triển khai và xây dựng ứng dụng**

### a) Công cụ và môi trường phát triển

#### a.1) Ngôn ngữ lập trình

**- Python**: Phiên bản Python 3.x được sử dụng để đảm bảo tính tương thích với các thư viện mới nhất.

**- Pygame**: Thư viện hỗ trợ đồ họa và âm thanh giúp phát triển game dễ dàng hơn.

#### a.2) Công cụ phát triển

**- Visual Studio Code hoặc PyCharm**: IDE hỗ trợ lập trình Python.

**- Git và GitHub**: Quản lý phiên bản và cộng tác phát triển.

**- Photoshop hoặc GIMP**: Thiết kế hình ảnh và giao diện đồ họa.

**- Audacity**: Chỉnh sửa và tối ưu hóa âm thanh game.

#### a.3) Môi trường thực thi

**- Hệ điều hành**: Windows

**- Yêu cầu phần cứng**: RAM tối thiểu 2GB, GPU hỗ trợ đồ họa cơ bản.

**b) Quá trình xây dựng ứng dụng**

#### b.1) Cấu trúc thư mục dự án

ChickenInvaders/

│-- main.py # Tập tin chạy chính

│-- player.py # Xử lý logic người chơi

│-- enemy.py # Xử lý logic kẻ địch

│-- bullet.py # Xử lý logic đạn

│-- game.py # Quản lý trò chơi

│-- assets/ # Chứa hình ảnh và âm thanh

│-- levels/ # Chứa dữ liệu cấp độ

│-- README.md # Hướng dẫn sử dụng

#### b.2) Quá trình lập trình

**- Bước 1**: Cài đặt môi trường phát triển bằng Python và Pygame.

**- Bước 2**: Thiết kế giao diện cơ bản với Pygame.

**- Bước 3**: Xây dựng các lớp đối tượng như Player, Enemy, Bullet.

**- Bước 4**: Viết vòng lặp game để cập nhật và hiển thị màn hình.

**- Bước 5**: Thêm các yếu tố như hiệu ứng âm thanh, điểm số, giao diện kết thúc.

**- Bước 6**: Tinh chỉnh gameplay và tối ưu hiệu suất.

**Chương V: Kiểm thử và đánh giá hệ thống**

### a) Kiểm thử hệ thống

#### a.1) Các phương pháp kiểm thử

**- Kiểm thử đơn vị (Unit Testing)**: Kiểm tra từng chức năng nhỏ trong game.

**- Kiểm thử tích hợp (Integration Testing)**: Kiểm tra sự tương tác giữa các thành phần như đạn và kẻ địch.

**- Kiểm thử hệ thống (System Testing)**: Kiểm tra toàn bộ game trong môi trường thực tế.

**- Kiểm thử hồi quy (Regression Testing)**: Đảm bảo các thay đổi không làm hỏng các tính năng trước đó.

#### a.2) Kịch bản kiểm thử

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Chức năng** | **Mô tả kiểm thử** | **Kết quả mong đợi** |
| 1 | Di chuyển tàu | Nhấn phím trái/phải | Tàu di chuyển mượt mà |
| 2 | Bắn đạn | Nhấn phím Space | Đạn xuất hiện và bay lên |
| 3 | Hủy diệt gà | Đạn chạm vào gà | Gà biến mất, điểm số tăng |
| 4 | Game Over | Hết mạng sống | Hiển thị màn hình kết thúc |

### b) Đánh giá hiệu suất

#### b.1) Tiêu chí đánh giá

**- Hiệu suất đồ họa**: Đảm bảo FPS ổn định (trên 30 FPS).

**- Độ mượt của gameplay**: Không có hiện tượng giật lag khi chơi.

**- Thời gian tải**: Game khởi động trong vòng 3 giây.

#### b.2) Cải thiện hiệu suất

- Giảm số lượng đối tượng xuất hiện cùng lúc.

- Tối ưu hóa hình ảnh và âm thanh để giảm tải CPU.

- Sử dụng thuật toán tối ưu cho xử lý va chạm.

### c) Phản hồi từ người chơi

**- Ưu điểm**:

* Gameplay hấp dẫn, dễ làm quen.
* Hình ảnh và âm thanh sinh động.
* Hệ thống điều khiển đơn giản.

**- Nhược điểm**:

* Cấp độ khó chưa đa dạng.
* Chưa có nhiều loại vũ khí.

#### - Kế hoạch cải tiến

* **Bổ sung AI cho kẻ địch**.
* **Thêm chế độ chơi mới** như Survival Mode.
* **Mở rộng hệ thống vũ khí** và kỹ năng cho người chơi.

# Tài liệu tham khảo

Trang điện tử [1]: [**https://www.w3schools.com/**](https://www.w3schools.com/)

Trang điện tử [2]: [PyGame: A Primer on Game Programming in Python – Real Python](https://realpython.com/pygame-a-primer/)

Tựa game đang làm theo: <https://www.interactionstudios.com>

[1] Mark Lutz, "Programming Python", 4th Edition, O'Reilly Media Inc, 2010.

[2] Brett Slatkin, “Effective Python: 90 Specific Ways to Write Better Python”, 2nd Edition, Addison-Wesley Professional, 2019.

[3] Van Rossum G, Drake FL. “Python 3 Reference Manual”. Scotts Valley, CA: CreateSpace; 2009.