



TIỂU LUẬN CUỐI KỲ

Môn học: Trí tuệ nhân tạo

Tên tiểu luận: Xây dựng trò chơi The Last Knight bằng Pygame

Giảng viên: ThS. Phan Thị Huyền Trang

Danh sách sinh viên thực hiện

Mã số SV	Họ và tên	Mức độ đóng góp (%)
23110193	Đinh Nguyễn Đức Duy	100
23110292	Đặng Trần Anh Quân	100
23110333	Trần An Thiên	100

NHẬN XÉT CỦA GIẢNG VIÊN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

ĐIỂM:

Tp. Hồ Chí Minh, ngày ...tháng ...năm 2025

Giảng viên chấm điểm

MỤC LỤC

MỤC LỤC.....	3
DANH MỤC TỪ VIẾT TẮT.....	4
PHẦN 1. MỞ ĐẦU.....	5
1.1. Giới thiệu chung về đề án.....	5
1.2. Phát biểu bài toán.....	5
1.3. Mục đích và yêu cầu.....	6
1.4. Phạm vi và đối tượng.....	7
PHẦN 2. CƠ SỞ LÝ THUYẾT.....	9
2.1 Giới thiệu thư viện Pygame.....	9
2.2 Các module Pygame sử dụng.....	9
2.3 Giới thiệu về các kỹ thuật sử dụng.....	10
2.4 Thuật toán AC-3 Search.....	11
2.4.1. Trình bày ý tưởng.....	11
2.4.2. Các bước thực hiện.....	12
2.4.3. Áp dụng vào bài tập 8 ô chữ.....	12
PHẦN 3. PHÂN TÍCH VÀ THIẾT KẾ GIẢI PHÁP.....	15
3.1 Ý tưởng thuật toán.....	15
3.2 Chi tiết về các thuật toán chính đã sử dụng.....	17
PHẦN 4. THỰC NGHIỆM, ĐÁNH GIÁ, PHÂN TÍCH KẾT QUẢ.....	22
4.1 Môi trường thử nghiệm.....	22
4.2 Kết quả thử nghiệm.....	25
4.3 Kết quả giao diện và hướng dẫn sử dụng trò chơi.....	31
PHẦN 5. KẾT LUẬN.....	36
5.1 Tóm tắt.....	36
5.2 Bài học kinh nghiệm.....	36
5.3 Hướng phát triển.....	37
Tài liệu tham khảo.....	39

DANH MỤC TỪ VIẾT TẮT

STT	Ký hiệu chữ viết tắt	Chữ viết đầy đủ
1	FPS	Frames Per Second
2	OOP	Object-Oriented Programming
3	BFS	Breadth-First Search
4	A*	A Star
5	CPU	Central Processing Unit
6	GPU	Graphics Processing Unit
7	IDE	Integrated Development Environment

PHẦN 1. MỞ ĐẦU

1.1. Giới thiệu chung về đề án

Dự án tập trung vào việc áp dụng các thuật toán tìm kiếm đường đi hiệu quả trong môn học Trí tuệ nhân tạo thông qua xây dựng một trò chơi phiêu lưu 2D, nơi người chơi điều khiển nhân vật đi khám phá các bản đồ đa dạng, sử dụng tuyệt chiêu hay các đòn đánh thường để chiến đấu và tiêu diệt quái vật đồng thời có thể thu thập các vật phẩm có trong bản đồ để gia tăng các thuộc tính cho người chơi, giúp việc tiêu diệt quái vật dễ dàng hơn. Trò chơi được xây dựng trên nền tảng thư viện Pygame, tận dụng khả năng mạnh mẽ của thư viện này trong việc xử lý đồ họa, âm thanh và tương tác người dùng. Bên cạnh đó, là việc thiết kế giao diện menu đơn giản và trực quan, nhằm cung cấp trải nghiệm người dùng tối ưu ngay từ khi bắt đầu. Giao diện này, kết hợp với nhạc nền phù hợp, được kỳ vọng sẽ tạo ra cảm giác hứng thú và lôi cuốn cho người chơi.

1.2. Phát biểu bài toán

Bài toán chính được nhắc đến ở dự án là áp dụng các thuật toán tìm kiếm trong không gian trạng thái được học từ môn học Trí tuệ nhân tạo và từ đó tạo ra một trò chơi phiêu lưu 2D không chỉ hấp dẫn, có tính giải trí cao, đồng thời thể hiện khả năng ứng dụng được các thuật toán tìm kiếm phổ biến cũng như kỹ thuật lập trình và sử dụng thư viện Pygame một cách hiệu quả.

Để áp dụng các thuật toán tìm kiếm cho hành vi của quái vật trong trò chơi “The Last Knight”, cần xác định rõ các thành phần của bài toán tìm kiếm thông tin trạng thái như sau:

Không gian trạng thái: Mỗi trạng thái đại diện cho một vị trí hay tọa độ cụ thể mà quái vật có thể chiếm giữ trên bản đồ của trò chơi. Tập hợp tất cả các vị trí hợp lệ trên bản đồ tạo thành không gian trạng thái.

Trạng thái ban đầu: Là vị trí xuất hiện đầu tiên của quái vật khi bắt đầu quá trình tìm kiếm đường đi đến người chơi.

Hành động: Là tập hợp các trạng thái di chuyển được thiết lập sẵn cho quái vật để có thể từ một vị trí hiện tại chuyển sang một vị trí mới. Trong môi trường bản đồ của trò chơi, thì tùy thuộc vào loại quái vật, các hành động thường bao gồm: di chuyển sang trái, phải, lên, xuống. Mỗi hành động này chỉ hợp lệ nếu vị trí đích đi đến không phải là các ô vật cản và nằm trong phạm vi của bản đồ.

Hàm chuyển trạng thái: Mô tả kết quả của việc thực hiện một hành động từ một trạng thái.

Hàm kiểm tra đích: Xác định xem đó có phải là trạng thái đích không, trạng thái đích lúc này là vị trí hiện tại của nhân vật, nếu vị trí của nhân vật và quái đủ gần trong tầm nhìn của quái vật thì xem như là tìm thấy đích.

Hàm chi phí bước: Ở đây nhóm cho chi phí cho mỗi hành động di chuyển (lên, xuống, qua trái, qua phải) là một. Chi phí của một đường đi lúc này là tổng chi phí các bước di chuyển trong đường đi đó.

Cụ thể, ở trò chơi cần giải quyết các vấn đề sau để có thể áp dụng các nhóm thuật toán:

Thiết kế thể giới trò chơi: Tạo ra các bản đồ đa dạng, có cấu trúc và thử thách khác nhau.

Xây dựng cơ chế di chuyển và chiến đấu: Triển khai các hành động linh hoạt cho nhân vật người chơi (chạy, nhảy, tấn công, sử dụng kỹ năng, né tránh, đỡ đòn, hồi phục) và hành vi chiến đấu thông minh cho quái vật, bao gồm cả việc tìm đường đi hiệu quả.

Quản lý tương tác: Xử lý các tương tác giữa người chơi, quái vật và môi trường, bao gồm cả việc thu thập vật phẩm, nhận sát thương và tương tác với các yếu tố môi trường (điểm thoát để chuyển qua một bản đồ mới).

Tối ưu hóa hiệu suất: Đảm bảo trò chơi chạy mượt mà và hiệu quả trên nhiều dạng cấu hình máy, đặc biệt là trong các tình huống có nhiều đối tượng trên màn hình.

Tạo giao diện người dùng: Thiết kế menu chính và các menu phụ (cài đặt bật/tắt âm thanh và nhạc nền, thoát trò chơi) để người chơi có thể dễ dàng điều hướng và tùy chỉnh trò chơi theo sở thích.

1.3. Mục đích và yêu cầu

Mục đích:

Đầu tiên là, xây dựng một trò chơi phiêu lưu 2D hoàn chỉnh, có khả năng chơi được và mang tính giải trí.

Thứ hai là, chứng minh khả năng áp dụng các kiến thức và kỹ năng lập trình Pygame vào thực tế.

Thứ ba là, tạo ra một sản phẩm có tính hoàn thiện tốt, với đồ họa và âm thanh hấp dẫn.

Cuối cùng là, nghiên cứu và triển khai các thuật toán tìm kiếm để cải thiện trí tuệ nhân tạo của quái vật trong việc tìm kiếm người chơi.

Yêu cầu: Tổng cộng có 6 yêu cầu chính:

Thứ nhất, cơ chế di chuyển và chiến đấu: Nhân vật người chơi có thể di chuyển linh hoạt theo chiều ngang, nhảy và thực hiện đa dạng các hành động. Hệ thống chiến đấu hỗ trợ các đòn đánh thường và kỹ năng đặc biệt, có hiệu ứng hình ảnh và âm thanh. Cơ chế va chạm chính xác để phát hiện tương tác giữa người chơi, quái vật và môi trường. Hệ thống quản lý máu và sát thương cho cả người chơi và quái vật.

Thứ hai, hệ thống bản đồ: Trò chơi có nhiều bản đồ khác nhau, được thiết kế với các thử thách riêng. Bản đồ được tải từ các tệp dữ liệu bên ngoài (tệp CSV) để dễ

dàng mở rộng và chỉnh sửa. Có sử dụng cơ chế chuyển đổi giữa các bản đồ khi kết thúc một bản đồ hay một màn.

Thứ ba, hệ thống quái vật: Có nhiều loại quái vật khác nhau, mỗi loại có hành vi, hình ảnh động và thuộc tính riêng. Quái vật có khả năng di chuyển, tấn công và bị tiêu diệt. Trí tuệ nhân tạo cơ bản được triển khai để điều khiển hành vi của quái vật, bao gồm cả việc tìm đường đi đến người chơi khi đi vào tầm nhìn của quái vật.

Thứ tư, hệ thống vật phẩm: Người chơi có thể thu thập các vật phẩm để tăng cường thuộc tính (tốc độ, sức tấn công, hồi máu). Các loại vật phẩm khác nhau có tác dụng khác nhau và được hiển thị rõ ràng trên từng bản đồ.

Thứ năm, giao diện người dùng: Menu chính cho phép bắt đầu chơi, truy cập cài đặt và thoát game. Ở menu cài đặt, thì cho phép điều chỉnh âm lượng nhạc nền và âm thanh. Các menu được thiết kế đơn giản, trực quan và dễ sử dụng.

Thứ sáu, thuật toán tìm kiếm, nhóm đã cân nhắc và chọn ra một thuật toán nằm trong từng nhóm để áp dụng vào trò chơi: Thuật toán A* (A star) ở nhóm thuật toán tìm kiếm có thông tin (Informed Search), BFS (Breadth-First Search) ở nhóm thuật toán tìm kiếm không có thông tin (Uninformed Search), Steepest-Ascent Hill Climbing ở nhóm thuật toán tìm kiếm cục bộ (Local Search), And Or Search ở nhóm thuật toán tìm kiếm trong môi trường phức tạp, Backtracking ở nhóm tìm kiếm trong môi trường có ràng buộc và cuối cùng Q-Learning ở nhóm Reinforcement Learning đã được triển khai để quái vật có thể tìm đường đi đến người chơi một cách hiệu quả. Các thuật toán này được tối ưu hóa để giảm thiểu việc sử dụng tài nguyên bộ nhớ, tránh sự ổn định không tốt đối với người chơi.

1.4. Phạm vi và đối tượng

Phạm vi: Dự án này tập trung vào việc tìm hiểu và ứng dụng các thuật toán tìm kiếm trong không gian trạng thái để giải quyết bài toán cốt lõi là định vị và di chuyển thông minh của các thực thể nhân tạo (AI - cụ thể là quái vật) trong môi trường trò chơi. Phạm vi này bao gồm:

Nghiên cứu và lựa chọn thuật toán: Phân tích các thuật toán là BFS, A*, Steepest-Ascent Hill Climbing, And Or Search, Backtracking và Q-Learning để xác định sự phù hợp của từng thuật toán đối với yêu cầu của trò chơi về hiệu quả tìm đường, chi phí tính toán và độ phức tạp của bản đồ.

Triển khai thuật toán: Áp dụng các thuật toán đã chọn vào cho các hành động của quái vật để chúng có khả năng tự tìm đường đến vị trí của người chơi, vượt qua rào cản một cách hợp lý.

Tối ưu và đánh giá: Xem xét các yếu tố ảnh hưởng đến hiệu suất của thuật toán trong môi trường trò chơi thời gian thực, chẳng hạn như kích thước bản đồ, số lượng quái vật, và tần suất cập nhật đường đi. Đánh giá hiệu quả của thuật toán thông qua

quan sát hành vi của quái vật và thời gian tìm được nhân vật.

Đối tượng: Dự án tập trung chính vào những người yêu thích thể loại trò chơi phiêu lưu 2D, đặc biệt là những người có sở thích khám phá, chiến đấu và phát triển nhân vật. Đồng thời, cũng hướng đến việc ứng dụng các thuật toán tìm kiếm trong không gian trạng thái vào trò chơi.

PHẦN 2. CƠ SỞ LÝ THUYẾT

2.1 Giới thiệu thư viện Pygame

Pygame là một tập hợp các module Python đa nền tảng, được thiết kế chuyên biệt để hỗ trợ việc phát triển trò chơi điện tử [1]. Nó cung cấp các công cụ và chức năng cần thiết để xử lý đồ họa, âm thanh, thiết bị đầu vào như chuột, bàn phím và nhiều yếu tố khác của trò chơi. Pygame được xây dựng dựa trên thư viện Simple DirectMedia Layer (SDL), cho phép các trò chơi được phát triển bằng Pygame có thể hoạt động trên nhiều hệ điều hành khác nhau, mang lại tính di động cao. Nhờ tính dễ học, dễ sử dụng và khả năng tạo ra các trò chơi 2D chất lượng, Pygame đã trở thành lựa chọn phổ biến cho các nhà phát triển game độc lập cũng như trong lĩnh vực giáo dục.

2.2 Các module Pygame sử dụng

Dự án đã sử dụng các module cốt lõi sau của thư viện Pygame:

Thứ nhất, `pygame.init()`: Hàm này là hàm khởi tạo tất cả các module cần thiết nằm trong thư viện của Pygame và rất quan trọng trước khi sử dụng bất kỳ chức năng nào của thư viện này. Việc gọi hàm `pygame.init()` là bắt buộc để Pygame hoạt động chính xác.

Thứ hai, `pygame.display`: Module này giúp quản lý cửa sổ trò chơi và các chức năng hiển thị. Trong đó có các hàm quan trọng bao gồm:

`pygame.display.set_mode()`: Tạo cửa sổ trò chơi với kích thước và chế độ hiển thị được chỉ định trước.

`pygame.display.set_caption()`: Đặt tiêu đề cho cửa sổ trò chơi.

`pygame.display.update()`: Cập nhật toàn bộ hoặc một phần màn hình để hiển thị những thay đổi.

Thứ ba, `pygame.image`: Module này cung cấp các công cụ để tải và xử lý hình ảnh. Trong đó, hàm `pygame.image.load()` được sử dụng để tải hình ảnh từ tệp và hàm `.convert_alpha()` được sử dụng để duy trì độ trong suốt của hình ảnh (đặc biệt với định dạng PNG) nhằm đảm bảo hiển thị hình ảnh chất lượng cao và tránh các vấn đề về màu sắc.

Thứ tư, `pygame.transform`: Module này cho phép biến đổi hình ảnh, bao gồm các thao tác như:

`pygame.transform.scale()`: Thay đổi kích thước của hình ảnh. Trong dự án, nó được dùng để điều chỉnh kích thước của các thành phần giao diện, khối bản đồ, nhân vật và quái vật trong trò chơi, đảm bảo tất cả hiển thị đúng với mong muốn trên màn hình.

`pygame.transform.flip()`: Lật hình ảnh theo chiều ngang hoặc chiều dọc.

`pygame.transform.rotate()`: Xoay hình ảnh theo một góc nhất định.

Thứ năm, pygame.font: Module này được sử dụng để hiển thị văn bản lên màn hình. Các hàm quan trọng bao gồm:

pygame.font.SysFont(): Chọn kiểu chữ mà hệ thống có sẵn.

.render(): Tạo một bề mặt chứa văn bản với màu sắc và khổ rộng của được chỉ định.

screen.blit(): Vẽ bề mặt chứa văn bản lên màn hình tại một vị trí nhất định.

Thứ sáu, pygame.mixer: Đây là module giúp quản lý âm thanh trong trò chơi. Bên trong có các hàm quan trọng như:

pygame.mixer.music.load(): Tải một tệp nhạc nền.

pygame.mixer.music.set_volume(): Điều chỉnh âm lượng của nhạc nền.

pygame.mixer.music.play(): Phát nhạc nền (có thể lặp lại).

Thứ bảy, pygame.time: Ở module này, chúng ta được cung cấp các hàm để quản lý thời gian trong trò chơi, chẳng hạn như:

pygame.time.delay(): Tạo độ trễ trong chương trình (tạm dừng thực thi).

pygame.time.get_ticks(): Lấy số milliseconds đã trôi qua kể từ khi Pygame được khởi tạo.

pygame.time.Clock(): Giúp quản lý tốc độ khung hình (FPS) của trò chơi, đảm bảo trò chơi chạy ổn định.

Thứ tám, pygame.event: Module này cho phép xử lý các sự kiện đầu vào từ người dùng, bao gồm:

pygame.event.get(): Lấy danh sách các sự kiện đã xảy ra.

pygame.event.type: Thuộc tính của một sự kiện cho biết loại sự kiện (ví dụ: `pygame.QUIT`, `pygame.KEYDOWN`, `pygame.MOUSEBUTTONDOWN`).

pygame.event.key: Thuộc tính của sự kiện `KEYDOWN` hoặc `KEYUP` cho biết phím nào đã được nhấn hoặc nhả.

pygame.event.MOUSEBUTTONDOWN: Thuộc tính của sự kiện này cho biết nút chuột nào đã được nhấn.

2.3 Giới thiệu về các kỹ thuật sử dụng

Kỹ thuật Parallax Scrolling:

Mục đích khi sử dụng kỹ thuật này là để tạo hiệu ứng chiều sâu và tăng tính thẩm mỹ cho bản đồ, kỹ thuật parallax scrolling được sử dụng trong module **world.py**.

Cách thức hoạt động: Kỹ thuật này cho phép các lớp nền (background layers) khác nhau di chuyển với tốc độ khác nhau khi người chơi di chuyển, nhằm tạo ra cảm giác về không gian ba chiều và làm phong phú trải nghiệm hình ảnh.

Sprite Group (Nhóm Sprite):

Trong thư viện Pygame có cung cấp lớp `pygame.sprite.Group`, đây là một lớp rất phổ biến vì nó giúp quản lý và vẽ nhiều đối tượng (sprites) trên màn hình một cách

hiệu quả.

Ứng dụng trong dự dự án: Sprite Groups được sử dụng để quản lý các nhóm đối tượng như nhóm quái vật (`enemy_group`), nhóm phép thuật (`magic_group`) và nhóm vật phẩm (`item_group`).

Lợi ích khi sử dụng Sprite Groups là giúp đơn giản hóa việc cập nhật và vẽ hàng loạt đối tượng cùng một lúc, cũng như phát hiện va chạm giữa các đối tượng.

Object-Oriented Programming (OOP – Lập trình hướng đối tượng):

Dự án được thiết kế theo mô hình OOP, nơi các thành phần của trò chơi được biểu diễn dưới dạng các lớp (classes) và đối tượng (objects).

Các lớp quan trọng bao gồm:

Player (trong `entities/player.py`): Đại diện cho nhân vật người chơi.

Enemy (trong `entities/enemy.py`): Đại diện cho các loại quái vật.

Magic (trong `magic.py`): Đại diện cho các kỹ năng phép thuật của người chơi.

World (trong `world.py`): Quản lý bản đồ và các yếu tố của thế giới trò chơi.

Button (trong `button.py`): Đại diện cho các nút tương tác trong menu.

ItemBox (trong `load_map.py`): Đại diện cho các vật phẩm có thể thu thập.

Decoration, Exit (trong `entities/environment.py`): Đại diện cho các yếu tố trang trí và điểm thoát của bản đồ.

Bên cạnh đó, `Search_Algorithm.py`: là nơi chứa các thuật toán tìm kiếm đường đi cho quái vật.

Ưu điểm của OOP: Giúp tổ chức mã nguồn thành các đơn vị module nhỏ hơn, dễ quản lý, bảo trì và cũng như tăng tính tái sử dụng của mã.

2.4 Thuật toán AC-3 Search

Thuật toán AC-3 (Arc Consistency algorithm #3) là một thuật toán phổ biến và hiệu quả được sử dụng để thực thi tính nhất quán cung cho các Bài toán Thỏa mãn Ràng buộc [2]. Mục tiêu của AC-3 không phải là tìm ra một lời giải cụ thể cho CSP, mà là thu hẹp miền giá trị của các biến bằng cách loại bỏ những giá trị không thể tham gia vào bất kỳ lời giải nào, từ đó đơn giản hóa bài toán cho các thuật toán tìm kiếm lời giải sau này.

2.4.1. Trình bày ý tưởng

Ý tưởng trung tâm của AC-3 là kiểm tra tính nhất quán của các "cung" trong đồ thị ràng buộc. Một CSP bao gồm:

- Một tập các biến $X = \{X_1, X_2, \dots, X_n\}$.
- Với mỗi biến X_i , có một miền giá trị D_i chứa tất cả các giá trị mà X_i có thể nhận.
- Một tập các ràng buộc $C = \{C_1, C_2, \dots, C_k\}$, mỗi ràng buộc C_j liên quan đến một tập con các biến và xác định các tổ hợp giá trị hợp lệ cho các biến đó.

Một cung (X_i, X_j) được gọi là **nhất quán** nếu với mọi giá trị x trong miền D_i

của X_i , tồn tại một giá trị y trong miền D_j của X_j sao cho việc gán $X_i=x$ và $X_j=y$ là hợp lệ đối với tất cả các ràng buộc liên quan đến X_i và X_j .

Nếu một giá trị $x \in D_i$ không tìm thấy bất kỳ giá trị $y \in D_j$ nào để thỏa mãn ràng buộc, thì x được coi là không nhất quán và sẽ bị loại bỏ khỏi D_i . Việc loại bỏ một giá trị khỏi miền của một biến có thể làm cho các cung khác liên quan đến biến đó trở nên không nhất quán. Do đó, AC-3 sử dụng một hàng đợi (queue) để quản lý các cung cần được kiểm tra lại. Quá trình này lặp đi lặp lại cho đến khi không còn giá trị nào có thể bị loại bỏ khỏi bất kỳ miền nào, lúc này CSP được gọi là nhất quán cung.

2.4.2. Các bước thực hiện

Thuật toán AC-3 có thể được mô tả bằng mã giả như sau:

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise

inputs: *csp*, a binary CSP with components (X, D, C)

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue})$

if REVISE(*csp*, X_i, X_j) **then**

if size of $D_i = 0$ **then return false**

for each X_k in $X_i.\text{NEIGHBORS} - \{X_j\}$ **do**

 add(X_k, X_i) to *queue*

return true

function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i

revised \leftarrow false

for each x in D_i **do**

if no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j **then**

 delete x from D_i

revised \leftarrow true

return revised

Mã giả trên là mã giả AIMA theo sách "Trí tuệ nhân tạo: Phương pháp tiếp cận hiện đại" của Stuart Russell và Peter Norvig biên soạn [3].

2.4.3. Áp dụng vào bài tập 8 ô chữ

Bài toán 8 ô chữ (8-Puzzle) một bài toán nổi tiếng trong lĩnh vực trí tuệ nhân tạo và giải đồ. Nó đóng vai trò là một bài toán mô hình hấp dẫn với nhiều ứng dụng khác nhau và được sử dụng rộng rãi để khám phá tìm kiếm theo phương pháp heuristic và khám phá không gian trạng thái [3]. Việc áp dụng trực tiếp AC-3 để giải bài toán 8-Puzzle không phải là cách tiếp cận thông thường, vì 8-Puzzle không phải là một CSP ở dạng chuẩn mà AC-3 thường xử lý (như tô màu đồ thị). Tuy nhiên, có thể cố

gắng hình dung một cách trừu tượng, dù có thể không tự nhiên bằng các ví dụ CSP điển hình.

Cách tiếp cận CSP cho 8-Puzzle:

Hãy xem xét một cách định nghĩa khác, tập trung vào việc xác định vị trí cuối cùng của các ô số, thay vì tìm chuỗi hành động.

Biến: X_k (với $k=1, \dots, 8$) là biến đại diện cho vị trí của ô số k . Ô trống có thể được xem là ô số 9 hoặc xử lý riêng.

Miền giá trị: $D_k = \{P_1, P_2, \dots, P_9\}$ là tập hợp 9 vị trí trên bảng 3×3 .

Ràng buộc:

Ràng buộc duy nhất (AllDiff): Tất cả các ô số phải ở các vị trí khác nhau. X_i khác X_j với mọi i khác j . Đây là ràng buộc chính.

Ràng buộc trạng thái đích (nếu có): Nếu chúng ta đang kiểm tra tính nhất quán của một trạng thái hiện tại so với trạng thái đích, thì ràng buộc có thể là $X_k =$ vị trí đích của ô k .

Minh họa AC-3 với ràng buộc AllDiff (một phần):

Giả sử chúng ta có 3 biến (để đơn giản hóa ví dụ) là X_1, X_2, X_3 và 3 vị trí P_1, P_2, P_3 . Miền ban đầu: $D_1 = \{P_1, P_2, P_3\}$ $D_2 = \{P_1, P_2, P_3\}$ $D_3 = \{P_1, P_2, P_3\}$

Ràng buộc: X_1 khác X_2 , X_1 khác X_3 , X_2 khác X_3 .

Bước 1: Khởi tạo hàng đợi với các cung: (X_1, X_2) , (X_2, X_1) , (X_1, X_3) , (X_3, X_1) , (X_2, X_3) , (X_3, X_2) .

Giả sử có một bước trước đó đã làm thay đổi miền, ví dụ: Sau một số bước xử lý, giả sử miền của X_1 bị thu hẹp còn $D_1 = \{P_1\}$. Do D_1 thay đổi, các cung (X_2, X_1) và (X_3, X_1) được thêm lại vào hàng đợi (nếu chưa có hoặc đã được xử lý).

Xét cung (X_2, X_1) : Miền $D_2 = \{P_1, P_2, P_3\}$, $D_1 = \{P_1\}$. Ràng buộc: X_2 khác X_1 .
Hàm REVISE (csp, X_2, X_1):

Với $xval = P_1$ trong D_2 :

Có $yval \in D_1$ (chỉ có P_1) sao cho P_1 khác $yval$ không? Không, vì $P_1 = P_1$.

Vậy, P_1 bị loại khỏi D_2 .

D_2 trở thành $\{P_2, P_3\}$. revised = true.

Vì D_2 thay đổi, các cung liên quan đến X_2 như (X_1, X_2) (nếu X_1 chưa cố định) và (X_3, X_2) sẽ được thêm vào hàng đợi.

Xét cung (X_3, X_1) : Miền $D_3 = \{P_1, P_2, P_3\}$, $D_1 = \{P_1\}$. Ràng buộc: X_3 khác X_1 . Tương tự, P_1 sẽ bị loại khỏi D_3 . D_3 trở thành $\{P_2, P_3\}$. revised = true.

Tiếp tục quá trình: Bây giờ, giả sử hàng đợi có cung (X_2, X_3) . $D_2 = \{P_2, P_3\}$, $D_3 = \{P_2, P_3\}$. Hàm REVISE (csp, X_2, X_3):

Với $xval = P_2$ trong D_2 :

Có $yval \in D_3$ (là P_2 hoặc P_3) sao cho P_2 khác $yval$ không? Có, $yval = P_3$ thỏa mãn P_2 khác P_3 . Vậy P_2 trong D_2 được giữ lại.

Với $xval = P_3$ trong D_2 :

Có $yval \in D_3$ (là P_2 hoặc P_3) sao cho P_3 khác $yval$ không? Có, $yval = P_2$ thỏa mãn P_3 khác P_2 . Vậy P_3 trong D_2 được giữ lại. Không có gì thay đổi trong D_2 khi xét cung (X_2, X_3) . $revised = false$.

Quá trình này tiếp tục cho đến khi hàng đợi rỗng. Nếu không có miền nào trở thành rỗng, hệ thống là nhất quán cung.

Lưu ý quan trọng về 8-Puzzle và AC-3:

Như đã đề cập, 8-Puzzle là bài toán tìm đường đi (pathfinding problem) trong không gian trạng thái, nơi mỗi trạng thái là một cấu hình của bảng, và các hành động là di chuyển ô trống. Các thuật toán như A^* và BFS trực tiếp tìm kiếm một chuỗi các hành động này.

Việc mô hình hóa 8-Puzzle như một CSP để AC-3 áp dụng trực tiếp cho việc *tìm lời giải* là không điển hình. Ví dụ trên chỉ mang tính chất minh họa cách AC-3 hoạt động với các biến và ràng buộc, và cách nó loại bỏ các giá trị không nhất quán từ miền của biến. Trong thực tế, AC-3 thường được dùng như một bước tiền xử lý để giảm không gian tìm kiếm cho một thuật toán backtracking hoặc tìm kiếm khác giải quyết CSP.

PHẦN 3. PHÂN TÍCH VÀ THIẾT KẾ GIẢI PHÁP

3.1 Ý tưởng thuật toán

Ý tưởng thuật toán cốt lõi của dự án "The Last Knight" xoay quanh việc lựa chọn, triển khai và đánh giá các thuật toán được học trong môn Trí tuệ nhân tạo phù hợp để giải quyết các bài toán cụ thể trong quá trình phát triển một trò chơi phiêu lưu 2D. Trọng tâm chính là xây dựng hành vi di chuyển thông minh và hợp lý cho các loại quái vật, để chúng có thể tìm đường và tương tác hiệu quả với người chơi trong một môi trường bản đồ phức tạp, có vật cản.

Tiếp cận chính cho hành vi tìm đường của quái vật: Ưu tiên các thuật toán tìm kiếm đường đi tối ưu và hiệu quả. Đối với việc quái vật cần tìm đường đến vị trí người chơi, các thuật toán tìm kiếm đồ thị như A^* và BFS (*Breadth-First Search*) được xem xét là các thuật toán dễ áp dụng nhất và nhóm đã chọn sử dụng đầu tiên.

*Thuật toán A^** được lựa chọn vì khả năng tìm đường đi ngắn nhất (hoặc chi phí thấp nhất) một cách hiệu quả bằng cách sử dụng hàm heuristic để định hướng quá trình tìm kiếm. Trong trò chơi, điều này giúp quái vật di chuyển một cách thông minh và tự nhiên hơn, tránh việc khám phá các khu vực không cần thiết.

Thuật toán BFS đảm bảo tìm ra đường đi ngắn nhất về số bước, hữu ích trong việc kiểm tra tính kết nối hoặc trên các bản đồ nhỏ hơn nơi chi phí tính toán không quá lớn.

Đánh giá hiệu suất: Đối với các thuật toán được áp dụng trực tiếp (A^* , BFS), hiệu suất được đánh giá dựa trên các yếu tố như:

Thời gian tìm kiếm: Quái vật cần phản ứng nhanh với sự di chuyển của người chơi.

Số bước/chi phí của đường đi: Đường đi tìm được có hợp lý và hiệu quả không.

Tài nguyên sử dụng: Mức độ tiêu thụ bộ nhớ và CPU, đảm bảo game vẫn chạy mượt mà khi có nhiều quái vật cùng hoạt động.

Xem xét các thuật toán khác và lý do áp dụng/không áp dụng: Bên cạnh A^* và BFS , nhóm cũng đã tìm hiểu các thuật toán khác trong khuôn khổ môn học Trí tuệ nhân tạo để có cái nhìn toàn diện hơn và quyết định chọn các thuật toán sau:

Đầu tiên, Steepest-Ascent Hill Climbing (Leo đồi dốc nhất):

Khả năng áp dụng: Thuật toán này tìm kiếm cục bộ, cố gắng di chuyển đến trạng thái lân cận tốt nhất dựa trên hàm heuristic. Nó có thể được xem xét cho các hành vi AI rất đơn giản, ví dụ như quái vật chỉ di chuyển một bước theo hướng tốt nhất tức thời mà không cần lập kế hoạch đường đi dài.

Nguyên nhân không phải là giải pháp chính cho tìm đường phức tạp:

Dễ bị kẹt ở các điểm tối ưu cục bộ, ví dụ quái vật đi vào ngõ cụt và không thể tìm đường ra mặc dù có đường đi khác.

Không đảm bảo tìm được đường đi tối ưu toàn cục hoặc thậm chí là tìm được đường đi trong mọi trường hợp phức tạp.

Hướng xử lý/Đánh giá: Nếu áp dụng, hiệu suất có thể được đánh giá bằng tốc độ tìm đến người chơi nhưng chất lượng đường đi thường kém hơn A*/BFS. Trong dự án này, do yêu cầu về đường đi thông minh và tránh bị kẹt, Steepest-Ascent Hill Climbing không được chọn làm thuật toán tìm đường chính cho quái vật.

Thứ hai, And-Or Search (Tìm kiếm AND-OR):

Khả năng áp dụng: Thuật toán này mạnh cho các bài toán có thể phân rã thành các bài toán con với điều kiện AND và OR.

Nguyên nhân không phải là giải pháp chính cho tìm đường của quái vật đơn lẻ:

Bài toán tìm đường của một quái vật đến người chơi thường là một chuỗi các lựa chọn OR (đi hướng này hoặc hướng kia) mà không có cấu trúc AND phức tạp.

Việc áp dụng And-Or Search có thể trở nên phức tạp hơn cần thiết và nếu không được tối ưu cẩn thận cho bài toán tìm đường đơn thuần, nó có thể hoạt động kém hiệu quả hơn A* (ví dụ, hoạt động như DFS nếu chỉ có nút OR).

Hướng xử lý/Đánh giá: Nếu một hành vi AI phức tạp hơn được thiết kế (ví dụ: quái vật cần hoàn thành một chuỗi các mục tiêu phụ theo một logic AND/OR nhất định trước khi tấn công người chơi), thì And-Or Search có thể trở nên phù hợp. Tuy nhiên, với phạm vi hiện tại của "The Last Knight", nó không được triển khai cho hành vi di chuyển chính.

Thứ ba, thuật toán Backtracking:

Khả năng áp dụng: Thuật toán Backtracking có thể được áp dụng để giải quyết các bài toán thỏa mãn ràng buộc (Constraint Satisfaction Problems - CSPs) trong trò chơi. Ví dụ: tạo màn chơi theo ràng buộc (đảm bảo các yếu tố màn chơi như vị trí quái vật, vật phẩm, địa hình thỏa mãn các điều kiện thiết kế) hoặc lập kế hoạch hành động cho quái vật.

Nguyên nhân không phải là giải pháp chính cho tìm đường của quái vật:

Bài toán tìm đường đi của quái vật đòi hỏi phải đưa ra quyết định nhanh chóng trong thời gian thực. Thuật toán Backtracking, với bản chất thử và sai, có thể không đủ hiệu quả để đáp ứng yêu cầu này.

Ngoài ra, việc mô hình hóa bài toán tìm đường dưới dạng một CSP có thể phức tạp và không tự nhiên bằng các thuật toán tìm kiếm đường đi chuyên dụng như A* hay BFS.

Hướng xử lý/Đánh giá:

Trong các trường hợp mà tính tối ưu của lời giải quan trọng hơn tốc độ tìm kiếm (ví dụ: tạo màn chơi), Backtracking có thể là một lựa chọn phù hợp.

Hiệu suất của Backtracking phụ thuộc nhiều vào cách các ràng buộc được định nghĩa và thứ tự các biến được xét.

Thứ tư, thuật toán Q-Learning:

Khả năng áp dụng: Thuật toán Q-Learning (một thuật toán học tăng cường) có thể được áp dụng để huấn luyện hành vi của quái vật. Ví dụ: quái vật có thể học các chiến lược chiến đấu hiệu quả hoặc học cách di chuyển tối ưu trong các môi trường phức tạp.

Nguyên nhân không phải là giải pháp chính cho tìm đường của quái vật:

Q-Learning đòi hỏi một lượng lớn dữ liệu (tức là tương tác với môi trường) để học được các giá trị Q tối ưu. Trong môi trường trò chơi thời gian thực, việc thu thập đủ dữ liệu có thể tốn thời gian và tài nguyên.

Ngoài ra, việc xác định hàm phần thưởng phù hợp cho bài toán tìm đường (ví dụ: phần thưởng khi đến gần người chơi, hình phạt khi va vào tường) có thể đòi hỏi thử nghiệm và điều chỉnh kỹ lưỡng.

Hướng xử lý/Đánh giá:

Nếu Q-Learning được sử dụng để huấn luyện hành vi của quái vật, cần đánh giá khả năng học hỏi và thích nghi của chúng.

Các yếu tố như tốc độ học (learning rate), hệ số chiết khấu (discount factor) và hàm phần thưởng có ảnh hưởng lớn đến hiệu suất của Q-Learning.

Cần xem xét sự cân bằng giữa khả năng khám phá (exploration) và khai thác (exploitation) của quái vật trong quá trình học.

Tổng kết ý tưởng thuật toán:

Dự án sẽ tập trung triển khai, tối ưu hóa và chọn A* làm thuật toán chính cho việc tìm đường của quái vật, do tính hiệu quả và khả năng tìm được đường đi thông minh. BFS sẽ được sử dụng để so sánh, đối chiếu hoặc cho các trường hợp đơn giản hơn. Các thuật toán khác như Steepest-Ascent Hill Climbing, And-Or Search, Backtracking và Q-Learning được nghiên cứu để hiểu rõ hơn về các phương pháp giải quyết vấn đề trong AI, phân tích ưu nhược điểm và xác định lý do chúng có thể không phải là lựa chọn hàng đầu cho bài toán tìm đường cụ thể trong "The Last Knight", nhưng có thể hữu ích cho các khía cạnh khác hoặc các phiên bản mở rộng của trò chơi trong tương lai. Đánh giá sẽ tập trung vào chất lượng đường đi, thời gian xử lý và tác động đến trải nghiệm người chơi.

3.2 Chi tiết về các thuật toán chính đã sử dụng

Các thuật toán tìm kiếm đường đi *A**, *BFS*, *Steepest Ascent Hill Climbing*, *And Or Search*, *Backtracking* và *Q-Learning* được sử dụng để điều khiển hành vi di chuyển của quái vật được triển khai trong module **Search_Algorithm.py**.

Thuật toán A (A Star):*

Bản chất: Là một thuật toán tìm kiếm có thông tin (informed search algorithm), nghĩa là nó sử dụng thông tin bổ sung (heuristic) về bài toán để tìm ra lời giải một cách hiệu quả hơn.

Hàm Heuristic: A* sử dụng một hàm heuristic để ước tính chi phí tức khoảng cách từ vị trí của quái vật đang đứng hiện tại trên bản đồ đến vị trí của người chơi. Hàm heuristic này cung cấp một "ước đoán thông minh" về chi phí còn lại, giúp A* ưu tiên các đường đi có khả năng dẫn đến mục tiêu nhanh hơn.

Trong trò chơi này, hàm heuristic sử dụng là khoảng cách Manhattan (tổng của sự khác biệt tuyệt đối giữa tọa độ x và tọa độ y) giữa quái vật và người chơi.

Cơ chế hoạt động: Thuật toán A* sử dụng một hàng đợi ưu tiên (priority queue) để lưu trữ các vị trí cần xét, với ưu tiên được xác định bởi tổng chi phí đã đi qua và chi phí ước tính đến mục tiêu ($f = g + h$, trong đó g là chi phí đã đi qua và h là chi phí ước tính).

Ưu điểm: A* thường được ưu tiên vì nó có thể tìm ra đường đi ngắn nhất (hoặc đường đi có chi phí thấp nhất) một cách hiệu quả hơn so với BFS, đặc biệt là trong các bản đồ lớn và phức tạp, nơi có nhiều ngóc ngách và đường đi khác nhau.

Thuật toán BFS (Breadth-First Search - Tìm kiếm theo chiều rộng):

Bản chất: Là một thuật toán tìm kiếm không có thông tin (uninformed search algorithm), nghĩa là nó không sử dụng bất kỳ thông tin bổ sung nào về bài toán ngoài thông tin có sẵn trong đồ thị (bản đồ).

Cơ chế hoạt động: BFS duyệt qua tất cả các vị trí theo thứ tự khoảng cách từ vị trí bắt đầu. Nó bắt đầu từ vị trí của quái vật và khám phá tất cả các vị trí lân cận có thể đi được, sau đó khám phá tất cả các vị trí lân cận của các vị trí đó, và cứ tiếp tục như vậy cho đến khi tìm thấy mục tiêu (người chơi).

Ưu điểm: BFS đảm bảo tìm ra đường đi ngắn nhất (nếu có).

Nhược điểm: có thể tốn kém hơn về mặt tài nguyên bộ nhớ so với A*, đặc biệt là trong các bản đồ có nhiều nhánh và đường đi khác nhau, vì nó cần lưu trữ tất cả các vị trí đã được khám phá.

Thuật toán Steepest Ascent Hill Climbing (Leo đồi dốc nhất):

Bản chất: Là một thuật toán tìm kiếm cục bộ (local search algorithm). Nó cố gắng cải thiện giải pháp hiện tại bằng cách thực hiện các bước đi đến trạng thái lân cận tốt nhất một cách lặp đi lặp lại. Nó không khám phá toàn bộ không gian trạng thái mà tập trung vào việc "leo lên" từ vị trí hiện tại.

Hàm Heuristic: Tương tự như A*, Steepest Ascent Hill Climbing sử dụng một hàm heuristic (ví dụ: khoảng cách Manhattan) để đánh giá "chất lượng" (thường là ước tính khoảng cách đến mục tiêu) của các trạng thái lân cận.

Cơ chế hoạt động: Bắt đầu từ vị trí hiện tại của quái vật. Tiếp đó là xem xét tất

cả các vị trí lân cận có thể di chuyển tới (được tạo bởi `generate_new_states`). Chọn vị trí lân cận có khoảng cách Manhattan nhỏ nhất đến người chơi và phải tốt hơn vị trí hiện tại. Nếu tìm thấy một vị trí như vậy, quái vật di chuyển đến đó và lặp lại quá trình. Thuật toán dừng lại khi không có vị trí lân cận nào tốt hơn vị trí hiện tại (đạt local maximum) hoặc khi đã đến được vị trí người chơi, hoặc sau một số lượt lặp tối đa.

Ưu điểm: Thường đơn giản để triển khai. Ít tốn bộ nhớ hơn so với A* hay BFS vì nó chỉ cần lưu giữ thông tin về trạng thái hiện tại và các lân cận tức thời. Có thể phản ứng nhanh và tìm ra một hướng di chuyển hợp lý một cách nhanh chóng trong nhiều trường hợp.

Nhược điểm: Rất dễ bị kẹt ở các điểm tối ưu cục bộ (ví dụ: quái vật đi vào một ngõ cụt mà mọi hướng đi tiếp theo đều có vẻ tệ hơn theo heuristic, mặc dù có một đường vòng tốt hơn ở xa hơn). Không đảm bảo tìm ra đường đi ngắn nhất hoặc tối ưu toàn cục. Kết quả phụ thuộc nhiều vào trạng thái bắt đầu và địa hình của bản đồ.

Thuật toán And-Or Search (Tìm kiếm AND-OR):

Bản chất: Là một thuật toán tìm kiếm được thiết kế cho các bài toán có thể được phân rã thành các bài toán con, trong đó một số vấn đề con yêu cầu tất cả các thành phần phải được giải quyết (điều kiện AND), trong khi các vấn đề con khác chỉ cần một trong số các lựa chọn được giải quyết (điều kiện OR).

Cơ chế hoạt động: Thuật toán xây dựng một cây hoặc đồ thị tìm kiếm bao gồm các nút AND và nút OR. Một nút OR được coi là có thể giải quyết nếu ít nhất một trong các nút con của nó có thể giải quyết được. Trong ngữ cảnh di chuyển của quái vật, việc tạo ra các trạng thái kế tiếp từ `generate_new_states` thường tạo ra các lựa chọn OR (quái vật có thể đi hướng này hoặc hướng kia). Một nút AND được coi là có thể giải quyết nếu tất cả các nút con của nó đều có thể giải quyết được (ít phổ biến hơn trong bài toán tìm đường đơn giản của quái vật trừ khi có các mục tiêu phụ phức tạp). Trong triển khai cho việc tìm đường đơn giản (nơi `generate_new_states` chủ yếu cung cấp các lựa chọn OR), thuật toán And-Or Search thường hoạt động theo kiểu đệ quy, khám phá các đường đi. Nếu trạng thái hiện tại là mục tiêu, nhánh đó thành công. Nếu không, nó tạo ra các trạng thái kế tiếp và thử từng trạng thái đó. Nếu bất kỳ trạng thái kế tiếp nào dẫn đến giải pháp, thì trạng thái hiện tại cũng được coi là dẫn đến giải pháp. Với cấu trúc này, nó có thể tương tự như một thuật toán Tìm kiếm theo chiều sâu (DFS).

Ưu điểm: Rất mạnh mẽ cho các bài toán có cấu trúc phân rã AND-OR rõ ràng, ví dụ như trong lập kế hoạch phức tạp hoặc các hệ chuyên gia.

Nhược điểm: Nếu bài toán không có cấu trúc AND rõ ràng và chủ yếu là các lựa chọn OR (như trong tìm đường đơn giản từ `generate_new_states`), thuật toán có thể không mang lại lợi thế đáng kể so với các thuật toán tìm kiếm khác và có thể hoạt

động kém hiệu quả hơn A* (ví dụ, không đảm bảo đường đi ngắn nhất nếu không được tối ưu hóa). Có thể phức tạp hơn trong việc triển khai và gỡ lỗi so với các thuật toán tìm kiếm đường đi truyền thống nếu không thực sự cần đến khả năng xử lý AND.

Thuật toán Backtracking:

Bản chất: Backtracking là một kỹ thuật giải thuật được sử dụng để giải quyết các bài toán thỏa mãn ràng buộc (Constraint Satisfaction Problems - CSPs) bằng cách xây dựng từng bước các ứng cử viên cho lời giải, và loại bỏ các ứng cử viên "không thể" ngay khi xác định được rằng chúng không thể nào hoàn thành thành một lời giải hợp lệ.

Cơ chế hoạt động:

Thuật toán bắt đầu bằng việc chọn một biến và gán cho nó một giá trị khả thi đầu tiên.

Sau đó, nó kiểm tra xem việc gán giá trị này có thỏa mãn tất cả các ràng buộc liên quan đến biến đó không.

Nếu có một ràng buộc không được thỏa mãn, thuật toán quay lui (backtrack), tức là loại bỏ giá trị vừa gán và thử một giá trị khác. Nếu không có giá trị nào thỏa mãn, thuật toán quay lui về biến trước đó và thử một giá trị khác cho biến đó.

Quá trình này tiếp tục cho đến khi tìm được một lời giải hợp lệ (tất cả các biến đều được gán giá trị thỏa mãn tất cả các ràng buộc) hoặc đã thử hết tất cả các khả năng mà không tìm thấy lời giải nào.

Ưu điểm: Tìm được tất cả các nghiệm và có thể hiệu quả trong các bài toán có không gian tìm kiếm nhỏ.

Nhược điểm: Có thể rất chậm trong các bài toán có không gian tìm kiếm lớn.

Thuật toán Q-Learning:

Bản chất: Q-Learning là một thuật toán học tăng cường (Reinforcement Learning) không cần mô hình để học một chiến lược hành động tối ưu.

Cơ chế hoạt động:

Thuật toán học một hàm Q, trong đó $Q(s, a)$ đại diện cho "chất lượng" của việc thực hiện hành động a trong trạng thái s.

Ban đầu, các giá trị Q được khởi tạo (thường là 0 hoặc ngẫu nhiên).

Agent (quái vật) tương tác với môi trường, thực hiện các hành động và nhận phần thưởng/hình phạt.

Các giá trị Q được cập nhật dựa trên kinh nghiệm, theo công thức: $Q(s, a) = Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$

s: trạng thái hiện tại, a: hành động vừa thực hiện, s': trạng thái tiếp theo, a': hành động tiếp theo, α : tốc độ học, γ : hệ số chiết khấu, r: phần thưởng.

Ưu điểm: Không cần mô hình môi trường trước và có thể học các chiến lược

phức tạp.

Nhược điểm: Đòi hỏi nhiều dữ liệu (tương tác với môi trường) để học. Việc lựa chọn các tham số (α , γ) và hàm phần thưởng có ảnh hưởng lớn đến hiệu suất.

Các hàm phụ tìm kiếm đường đi được viết ở module **Search_Algorithm.py** bao gồm:

Hàm `get_start()`: Xác định vị trí bắt đầu của quái vật trên bản đồ dựa trên hình chữ nhật bao quanh quái vật (`enemy.rect`). Hàm này có thể chuyển đổi tọa độ pixel thành tọa độ ô (tile) để phù hợp với biểu diễn bản đồ.

Hàm `get_goal()`: Xác định vị trí mục tiêu (người chơi) trên bản đồ dựa trên hình chữ nhật bao quanh người chơi (`player.rect`). Tương tự như `get_start()`, hàm này có thể chuyển đổi tọa độ pixel thành tọa độ ô (tile).

Hàm `generate_new_states()`: Tạo ra các trạng thái mới (các vị trí có thể di chuyển tới) từ một trạng thái hiện tại (vị trí hiện tại của quái vật), giới hạn bởi các vật cản (các khối bản đồ đã thiết lập) và biên giới bản đồ. Hàm này có thể trả về một danh sách hoặc tập hợp các trạng thái mới.

Hàm `manhattan()`: Tính toán khoảng cách Manhattan (một loại hàm heuristic) giữa hai vị trí trên bản đồ.

Việc lựa chọn giữa A*, BFS, Steepest Ascent Hill Climbing, And-Or Search, Backtracking và Q-Learning có thể phụ thuộc vào yêu cầu cụ thể của hành vi quái vật mong muốn và sự cân bằng giữa hiệu suất, độ phức tạp và "sự thông minh" của đường đi. Trong dự án, A* thường được ưu tiên cho hành vi tìm đường chính xác và hiệu quả đến mục tiêu. BFS có thể hữu ích khi cần đảm bảo đường đi ngắn nhất về số ô mà không quá chú trọng đến chi phí tính toán trên bản đồ nhỏ. Steepest Ascent Hill Climbing có thể tạo ra hành vi di chuyển tức thời, phản ứng nhanh nhưng không đảm bảo tối ưu và có thể bị kẹt. And-Or Search, Backtracking và Q-Learning trong bối cảnh hiện tại của việc tìm đường đi của quái vật là không phù hợp và hiệu quả.

PHẦN 4. THỰC NGHIỆM, ĐÁNH GIÁ, PHÂN TÍCH KẾT QUẢ

4.1 Môi trường thử nghiệm

Các thử nghiệm trong trò chơi được chạy trên thiết bị máy tính cá nhân có cấu hình chi tiết như sau:

Hệ điều hành: Windows 11.

Bộ xử lý (CPU): Intel Core i7 11800H có tốc độ xung nhịp cơ bản là 2.3GHz, và có thể ép xung lên đến 4.6GHz, có 8 nhân và 16 luồng.

Bộ nhớ RAM: gồm 2 thanh Ram 8GB DDR4, có tốc độ bus là 3200MHz.

Card đồ họa (GPU): NVIDIA GeForce RTX 3050 Laptop GPU với 4GB VRAM GDDR6.

Màn hình: Độ phân giải là 1920x1080 với tần số quét màn hình là 60Hz.

Môi trường phần mềm và thư viện:

Ngôn ngữ lập trình: Sử dụng Python 3.12.10.

Thư viện Pygame: Sử dụng Pygame phiên bản 2.6.1.

Môi trường phát triển tích hợp (IDE): Visual Studio Code.

Tham số thiết lập của trò chơi khi thử nghiệm:

Để đảm bảo tính nhất quán và khả năng so sánh giữa các lần thử nghiệm, các tham số sau của trò chơi đã được thiết lập cố định (hoặc thay đổi một cách có kiểm soát):

Độ phân giải màn hình trò chơi: Chạy với độ phân giải gốc của thiết bị cùng với kích thước như thiết lập ở module **setting.py** là rộng 1300 pixels, cao 650 pixels.

Chất lượng đồ họa: Mặc định.

Âm lượng nhạc nền và hiệu ứng âm thanh: Đặt ở mức 100% hoặc tắt hoàn toàn để tập trung vào hiệu suất.

Các bản đồ/màn chơi cụ thể được sử dụng để thử nghiệm: Thử nghiệm hiệu suất được thực hiện tại Màn 2.

Số lượng đối tượng (quái vật, vật phẩm) trên màn hình: Kiểm tra hiệu suất với 2 quái bay cùng chỗ hoặc 2 quái vật loại Boss xuất hiện đồng thời hoặc 3 quái nhỏ trở lên.

Các tính năng cụ thể được bật/tắt khi thử nghiệm:

Đầu tiên, phím F1 dùng để hiển thị thông tin cơ bản bao gồm:



Hình hiển thị thông tin khi nhấn phím F1

Có thể thấy được tên thuật toán đang áp dụng (Algo), thời gian tìm thấy đường đi (Time), số trạng thái mà thuật toán khám phá được hay chi phí tìm kiếm (Nodes) và độ dài đường đi (PathLen).

Thứ hai, phím F2 để vẽ đường đi của quái vật tìm đến nhân vật, nếu nhân vật di chuyển thì đường đi mới sẽ được vẽ tiếp sau khi đã đi hết đường đi cũ.



Hình hiển thị đường đi của quái vật khi nhấn phím F2

Thứ ba, phím F3 dùng để hiển thị tầm nhìn của quái (vision).



Hình hiển thị tầm nhìn của quái vật

Thứ tư, phím F4 dùng để đổi thuật toán tìm kiếm cho quái vật.



Hình ảnh trước khi bấm F4



Hình ảnh sau khi bấm F4

Quy trình thực hiện thử nghiệm:

Thử nghiệm chức năng:

Thứ nhất, cách nhân vật di chuyển(chạy, nhảy, đứng yên) có đúng với hoạt ảnh như đã tạo không.

Thứ hai, khi nhặt vật phẩm có gia tăng trạng thái của nhân vật và đồng thời có cập nhật lên màn hình trạng thái nhân vật khi nhặt được vật phẩm không.

Thứ ba, các đòn đánh thường và kỹ năng có chạy đúng hoạt ảnh, đúng hiệu quả như đã thiết lập và có gây sát thương lên quái không.

Thứ tư, các vật cản trong bản đồ có cản trở nhân vật đúng như đã thiết lập không.

Thử nghiệm hiệu suất:

Thứ nhất, chơi liên tục màn 2 và treo máy trong 10 phút để xem nhiệt độ máy và độ ổn định của trò chơi trong thời gian dài.

Thứ hai, thử nghiệm tình huống có nhiều hiệu ứng hình ảnh và nhiều quái vật tập hợp cùng một chỗ.

Thử nghiệm AI của quái vật:

Thứ nhất, quái vật có chạy đúng hành vi như đã thiết lập, có gây sát thương lên người chơi, nhận sát thương, biến mất khi hết thanh máu không.

Thứ hai, quan sát hành động tìm kiếm đường đi của quái vật khi người chơi ở các vị trí khác nhau trong tầm nhìn ở màn đó.

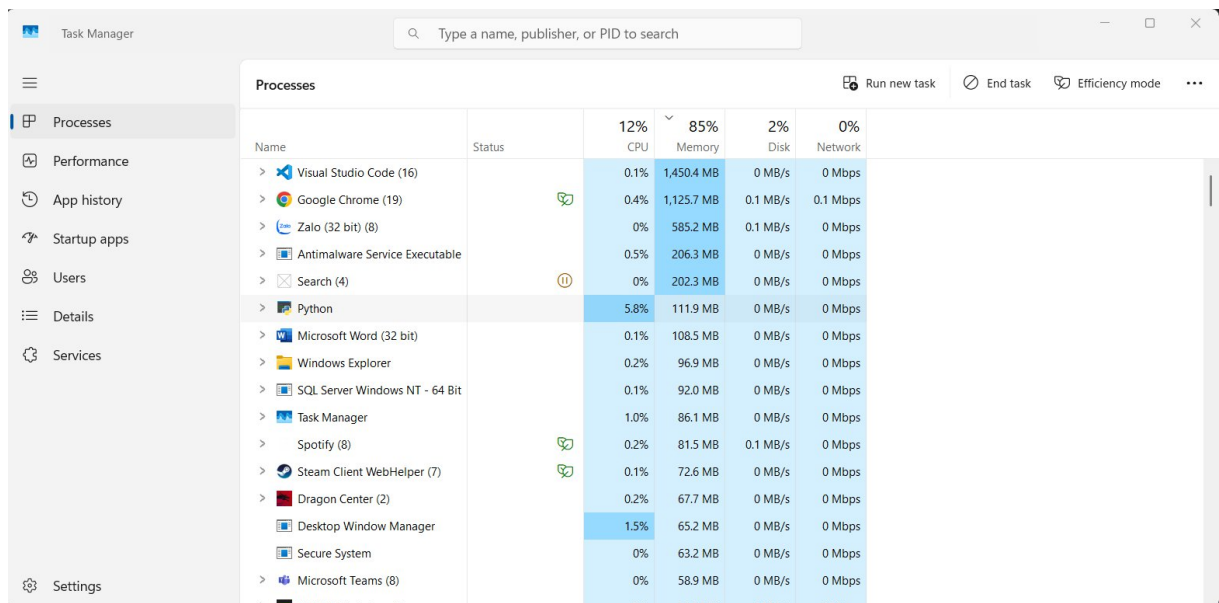
Công cụ hỗ trợ đo lường và ghi nhận kết quả:

Phân tích bộ nhớ/CPU:

Sử dụng Task Manager của Windows để xem tình trạng sử dụng tài nguyên thiết bị khi khởi chạy trò chơi.

Bằng những thử nghiệm đã tiến hành như trên, thì nhóm đã đưa ra được kết quả thử nghiệm bao gồm hiệu suất, chức năng, lỗi đã khắc phục, đánh giá ưu và nhược điểm.

4.2 Kết quả thử nghiệm



Name	Status	12% CPU	85% Memory	2% Disk	0% Network
Visual Studio Code (16)		0.1%	1,450.4 MB	0 MB/s	0 Mbps
Google Chrome (19)		0.4%	1,125.7 MB	0.1 MB/s	0.1 Mbps
Zalo (32 bit) (8)		0%	585.2 MB	0.1 MB/s	0 Mbps
Antimalware Service Executable		0.5%	206.3 MB	0 MB/s	0 Mbps
Search (4)		0%	202.3 MB	0 MB/s	0 Mbps
Python		5.8%	111.9 MB	0 MB/s	0 Mbps
Microsoft Word (32 bit)		0.1%	108.5 MB	0 MB/s	0 Mbps
Windows Explorer		0.2%	96.9 MB	0 MB/s	0 Mbps
SQL Server Windows NT - 64 Bit		0.1%	92.0 MB	0 MB/s	0 Mbps
Task Manager		1.0%	86.1 MB	0 MB/s	0 Mbps
Spotify (8)		0.2%	81.5 MB	0.1 MB/s	0 Mbps
Steam Client WebHelper (7)		0.1%	72.6 MB	0 MB/s	0 Mbps
Dragon Center (2)		0.2%	67.7 MB	0 MB/s	0 Mbps
Desktop Window Manager		1.5%	65.2 MB	0 MB/s	0 Mbps
Secure System		0%	63.2 MB	0 MB/s	0 Mbps
Microsoft Teams (8)		0%	58.9 MB	0 MB/s	0 Mbps
ABR/DA Container (6)		0%	56.3 MB	0 MB/s	0 Mbps

Hình ảnh tình trạng sử dụng tài nguyên thiết bị khi khởi chạy trò chơi

Thông qua Task Manager, có thể thấy được khi khởi chạy trò chơi cùng với Visual Studio Code thì tài nguyên Ram cần cung cấp để chạy trò chơi là khoảng 1.5GB, cũng tương đối nặng.



Hình ảnh nhiệt độ máy trước khi khởi chạy trò chơi

Trước khi khởi chạy trò chơi, nhiệt độ máy khá mát khi đang ở mức 49 độ C.



Hình ảnh nhiệt độ máy sau khi chơi được 10 phút

Tuy nhiên khi khởi động lên và chơi được khoảng 10 phút thì nhiệt độ máy đã tăng lên gần 20 độ C. Đây là một mức nhiệt độ khi chạy các tác vụ trung bình ở máy tính.



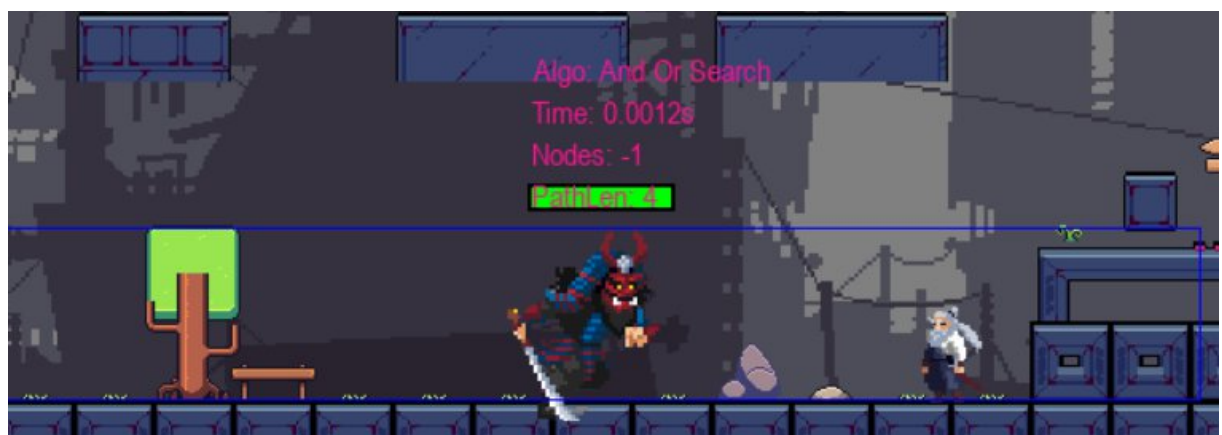
*Đường đi và không gian tìm kiếm của A**



Đường đi và không gian tìm kiếm của BFS



Đường đi và không gian tìm kiếm của Steepest-Ascent Hill Climbing



Đường đi và không gian tìm kiếm của And Or Search



Đường đi và không gian tìm kiếm của Backtracking



Đường đi và không gian tìm kiếm của Q-Learning

Hiệu suất:

Trò chơi chạy mượt mà với tốc độ khung hình ổn định ở trên cấu hình máy chạy thử nghiệm, ngay cả khi có nhiều quái vật và hiệu ứng hình ảnh trên màn hình. Điều này cho thấy rằng mã nguồn đã được tối ưu hóa tốt và có thể xử lý được tải nặng.

Thời gian tải bản đồ từ các tệp CSV có độ trễ khoảng 2 giây và có chút gián đoạn nhỏ. Việc sử dụng các tệp CSV cho phép dễ dàng tạo và chỉnh sửa bản đồ mà không ảnh hưởng đến hiệu suất tổng thể.

Thuật toán tìm kiếm BFS, A* hoạt động hiệu quả, giúp quái vật di chuyển một cách thông minh và tiết kiệm tài nguyên hệ thống. Quái vật có thể tìm đường đi đến người chơi một cách nhanh chóng và tránh các vật cản tốt.

Thuật toán Steepest Ascent Hill Climbing vẫn còn gặp một số lỗi đối với việc cho quái bay tìm đến người chơi bởi vì quái bay có tầm nhìn lớn tức không gian trạng thái nhiều hơn so với các loại quái vật còn lại.

Thuật toán And Or Search thì bị giới hạn không gian tìm kiếm, hiệu suất không tốt, gây giật lag khi đối diện với quái vật dạng bay.

Thuật toán Backtracking khi gặp quái vật dạng bay thì có vẽ ra đường đi tuy nhiên cũng gặp tình trạng giật lag, đặc biệt khi tiếp cận ở phía dưới hoặc phía trên.

Thuật toán Q-Learning khiến cho quái vật tự đánh mặc dù không thấy nhân vật trong tầm nhìn và khi nhân vật tiến vào tầm nhìn thì quái vật cũng không tìm thấy đường đi đến nhân vật nhưng lại gây mất máu nhân vật.

Chức năng:

Cơ chế di chuyển nhân vật phản hồi tốt và cho phép người chơi di chuyển, nhảy và thực hiện các hành động khác một cách linh hoạt. Các hành động như dịch chuyển ngắn và đỡ đòn hoạt động chính xác.

Khả năng chiến đấu với các đòn đánh thường và kỹ năng đặc biệt đều hoạt động đúng như đã thiết lập.

Quái vật có hành vi di chuyển và tấn công đa dạng đối với từng loại quái vật. Về phần trí tuệ nhân tạo của quái vật hoạt động hiệu quả trong việc tìm đường đi ngắn nhất đến người chơi khi người chơi bước vào tầm nhìn của quái vật.

Vật phẩm được thu thập và ảnh hưởng đến thuộc tính nhân vật một cách rõ ràng.

Menu điều hướng trực quan và dễ sử dụng, cho phép người chơi dễ dàng điều chỉnh các thiết lập trước khi vào trò chơi.

Hiệu ứng delay trên các nút menu cung cấp phản hồi trực quan và cải thiện trải nghiệm người dùng, giúp người chơi biết rằng hành động của họ đã được ghi nhận.

Lỗi đã khắc phục:

Trong quá trình phát triển, một số lỗi nhỏ đã được phát hiện và khắc phục, bao gồm:

Lỗi đồ họa khi hiển thị một số hình ảnh, đặc biệt là khi thay đổi kích thước. Điều này được giải quyết bằng cách tối ưu hóa việc tải và xử lý hình ảnh, cũng như sử dụng các hàm Pygame một cách chính xác.

Lỗi xử lý va chạm gây ra sự thiếu chính xác của trò chơi.

Vấn đề về hiệu suất khi có quá nhiều quái vật trên màn hình và sử dụng hàm tìm kiếm đường đi hiệu quả không phù hợp cho quái vật, dẫn đến việc trò chơi bị giật lag. Điều này được giải quyết bằng cách tối ưu hóa thuật toán tìm kiếm và quản lý đối tượng, giảm thiểu số lượng đối tượng cần xử lý cùng một lúc.

Đánh giá

Ưu điểm:

Đồ họa 2D đẹp mắt và phù hợp với thể loại phiêu lưu, tạo ra một thế giới trò chơi hấp dẫn và lôi cuốn. Các hình ảnh được thiết kế chi tiết và có phong cách nghệ thuật nhất quán.

Âm thanh sống động và tạo không khí tốt cho trò chơi, tăng cường trải nghiệm người chơi. Nhạc nền và hiệu ứng âm thanh được lựa chọn và sử dụng một cách cẩn thận.

Cơ chế chiến đấu đa dạng và thú vị, với các đòn đánh thường và kỹ năng đặc biệt. Người chơi có thể sử dụng các kỹ năng khác nhau để đối phó với các loại quái vật khác nhau.

Hệ thống menu được thiết kế trực quan, giúp người chơi dễ dàng điều hướng và tùy chỉnh trò chơi dễ dàng.

Thuật toán tìm kiếm BFS và A* là hoạt động hiệu quả khá tốt đối với trò chơi này, giúp quái vật di chuyển thông minh và tiết kiệm tài nguyên hệ thống. Điều này đặc biệt quan trọng trong các bản đồ lớn và phức tạp, nơi có nhiều đường đi và vật cản.

Cấu trúc mã nguồn rõ ràng và dễ bảo trì theo các nguyên tắc lập trình hướng đối tượng. Điều này giúp cho việc phát triển, sửa lỗi và mở rộng trò chơi trong tương lai trở nên dễ dàng hơn.

Nhược điểm:

Mặc dù đã chọn một thuật toán trong mỗi nhóm (tổng 6 cái) nhưng vẫn không thể áp dụng hết vào trò chơi được vì hiệu suất và độ phù hợp.

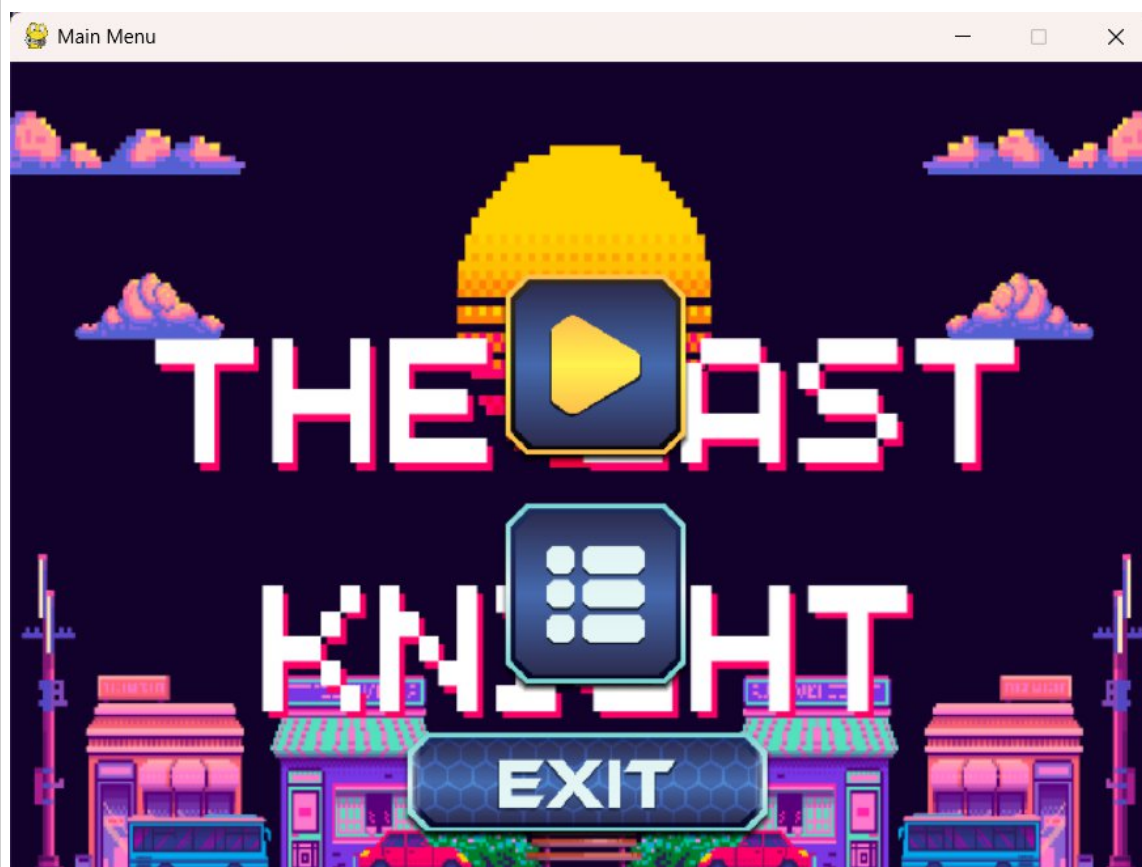
Số lượng màn chơi và loại quái vật có thể được mở rộng để tăng tính đa dạng và thời lượng chơi. Điều này sẽ cung cấp cho người chơi nhiều nội dung mới để khám phá và chinh phục.

Tính năng AI của quái vật có thể được cải thiện để tạo ra những thử thách đa dạng và khó đoán hơn cho người chơi. Ví dụ, quái vật có thể sử dụng các chiến thuật khác nhau, phối hợp với nhau để tấn công, hoặc có các hành động đặc biệt.

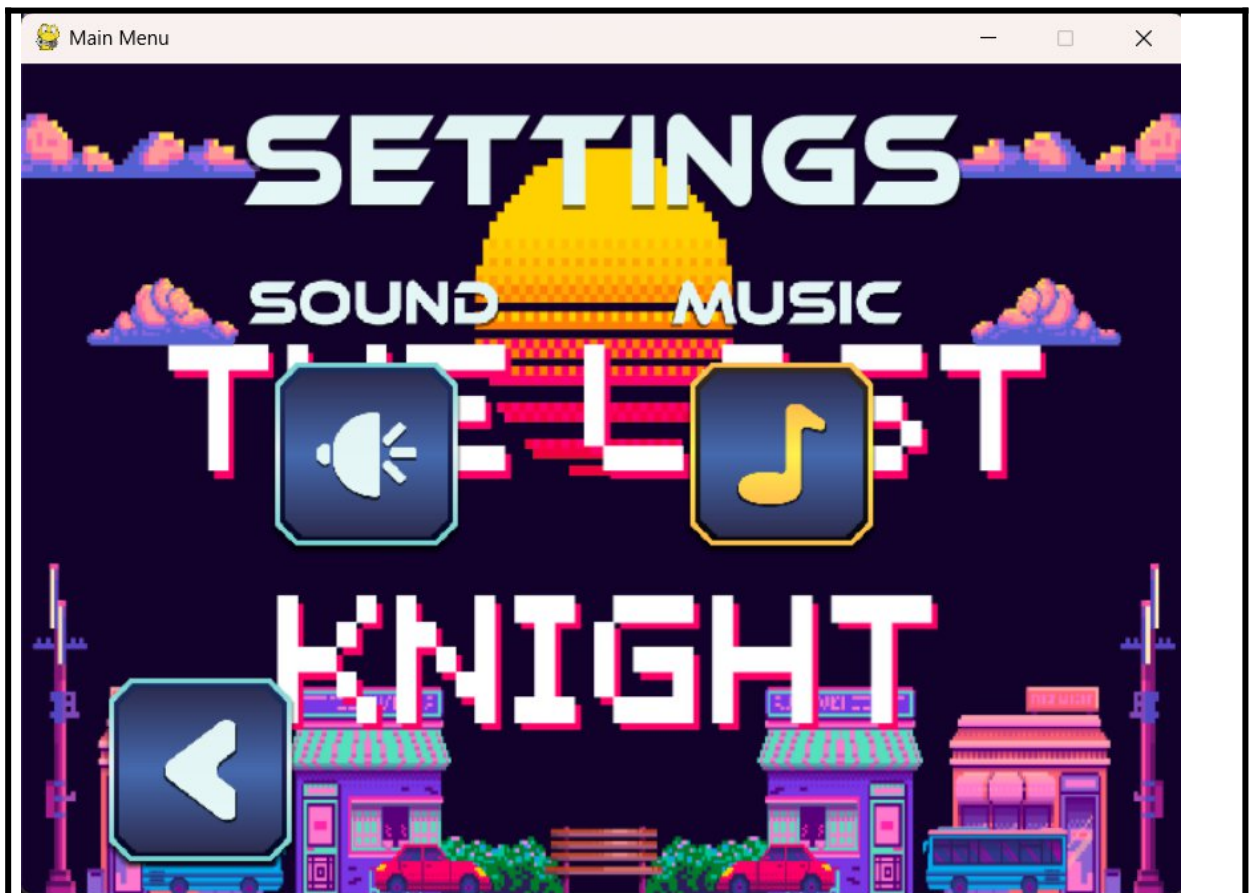
4.3 Kết quả giao diện và hướng dẫn sử dụng trò chơi

Ở phần này, nhóm sẽ trình bày các giao diện chính của trò chơi “The Last Knight” và cung cấp hướng dẫn chi tiết để người dùng có thể cài đặt và khởi chạy trò chơi trên máy tính cá nhân.

Giao diện trò chơi:



Màn hình chính (Menu game)



Màn hình cài đặt (Menu setting)

Giao diện trong trò chơi:



Màn hình Game over



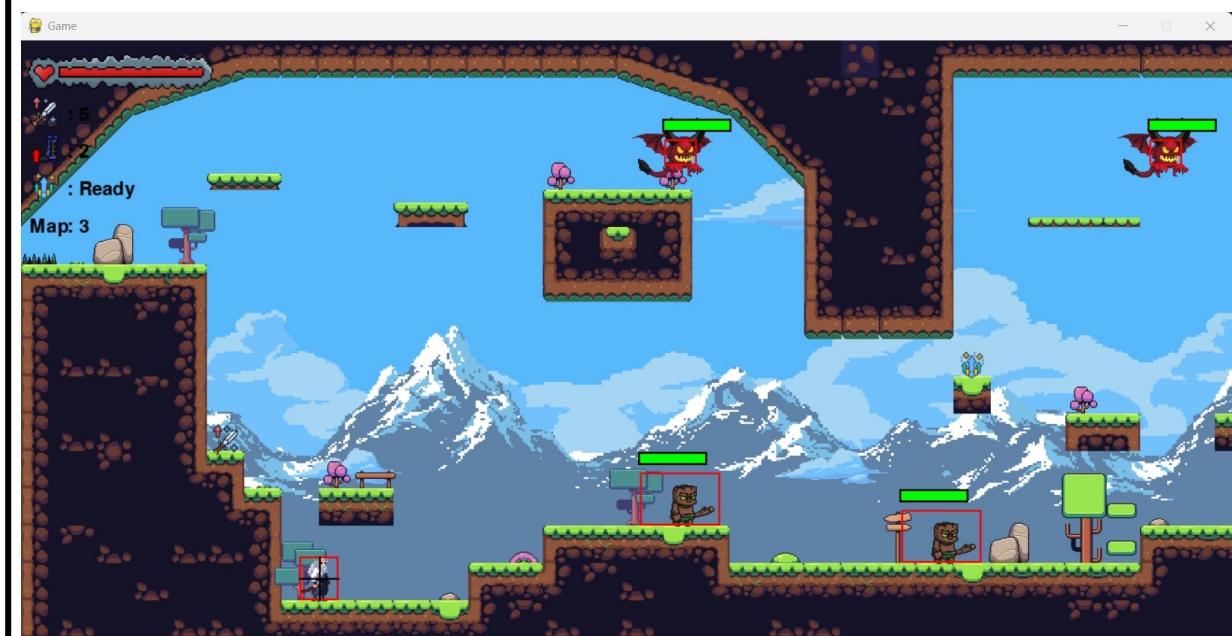
Màn hình qua màn



Giao diện xuất hiện đầu tiên của màn 1



Giao diện xuất hiện đầu tiên của màn 2



Giao diện xuất hiện đầu tiên của màn 3

Hướng dẫn thực thi trò chơi:

Ở phần này, nhóm sẽ cung cấp các bước cần thiết để người dùng có thể chạy được trò chơi trên máy của mình.

Yêu cầu hệ thống: Cần một thiết bị có cấu hình Windows 10 trở lên, có phiên bản python khuyến nghị là 3.12.10, có phiên bản pygame khuyến nghị là 2.6.1. Phần cứng thì cần tối thiểu 8GB RAM, có card đồ họa cơ bản, CPU Core không quá lỗi thời

Chuẩn bị môi trường:

Cài đặt Python: Hướng dẫn tải Python từ trang chủ (python.org). Ngoài ra, cần lưu ý quan trọng khi cài đặt.

Cài đặt Pygame: Hướng dẫn cài đặt Pygame bằng pip: Mở Command Prompt/Terminal và gõ lệnh ***pip install pygame***.

Cài đặt Pycaw: Tương tự ta gõ lệnh ***pip install pycaw***.

Tải trò chơi này bằng cách truy cập link github sau:

<https://github.com/DangTranAnhQuan/-n-cu-i-k-Tr-Tu-Nh-n-T-o>

Sau đó, mở Command Prompt, gõ lệnh **cd <link dẫn tới chỗ muốn tải>** rồi nhấn enter. Tiếp đó, gõ lệnh **git clone <link github>** rồi đợi hệ thống tự tải trò chơi từ github về địa chỉ vừa trở tới ở máy của mình. Khi tải xong thì mở folder vừa tải bằng Python và bấm run file **PLAY.py** để chơi trò chơi.

PHẦN 5. KẾT LUẬN

5.1 Tóm tắt

Dự án đã xây dựng thành công một trò chơi cơ bản về thể loại phiêu lưu với đồ họa giao diện là 2D bằng thư viện Pygame, qua đó thể hiện được khả năng ứng dụng các kiến thức đã học trong môn Trí tuệ nhân tạo và các kỹ năng lập trình vào thực tế. Trò chơi có các chức năng cốt lõi như di chuyển nhân vật, chiến đấu với quái vật, thu thập vật phẩm và điều hướng menu. Việc sử dụng các thuật toán tìm kiếm và thiết kế hướng đối tượng đã giúp ứng dụng được các kiến thức đã học trong môn học này đồng thời cũng tối ưu hóa hiệu suất khi áp dụng đúng và hiệu quả thuật toán tìm kiếm và khả năng bảo trì của mã nguồn nhờ tách rõ các module có chức năng riêng biệt. Tuy nhiên, vẫn còn một số hạn chế và hướng phát triển trong tương lai để trò chơi trở nên hoàn thiện hơn và mang lại trải nghiệm tốt hơn cho người chơi.

5.2 Bài học kinh nghiệm

Trong quá trình thực hiện dự án đã mang lại nhiều bài học quý báu:

Đầu tiên, không thể không kể đến tầm quan trọng trong việc lập kế hoạch và thiết kế chi tiết trước khi bắt đầu viết mã, đây là một phần nhóm chuẩn bị và tìm hiểu lâu nhất vì đây là một phần tiên quyết và cốt lõi, cũng như rất quan trọng để đảm bảo dự án đi đúng hướng và đạt được mục tiêu. Điều này bao gồm việc xác định rõ ràng các yêu cầu, thiết kế trò chơi, thiết kế chi tiết cho từng chức năng, cũng như sử dụng một thuật toán tối ưu nhất trong mỗi nhóm thuật toán.

Thứ hai, phải hiểu sâu sắc về thư viện Pygame, vì đây là một thư viện mạnh mẽ và linh hoạt trong ngôn ngữ python, thư viện cung cấp nhiều công cụ để phát triển trò chơi, nhưng trong quá trình xây dựng dự án này, nhóm cũng gặp rất nhiều khó khăn vì số lượng module của thư viện nhiều và việc hiểu rõ các module và chức năng của nó là cần thiết để sử dụng hiệu quả vào trò chơi. Do đó, việc tìm hiểu tài liệu liên quan đến dự án và ví dụ, cách giải quyết khi gặp một số lỗi cơ bản hay còn gọi là khả năng debug là rất quan trọng để tận dụng tối đa khả năng của Pygame.

Thứ ba, về vấn đề quản lý bộ nhớ và tối ưu hóa hiệu suất của dự án cũng là một trong những yếu tố then chốt cần được chú ý trong quá trình phát triển trò chơi, đặc biệt là khi có nhiều đối tượng và hiệu ứng hình ảnh trên màn hình bởi vì khi chạy ở thiết bị dùng để kiểm thử dự án có thể không gặp tình trạng giật hay tràn bộ nhớ nhưng khi đổi thiết bị khác có cấu hình thấp hơn thì tình trạng này có thể diễn ra thường xuyên, gây ảnh hưởng đến trải nghiệm của người chơi. Bên cạnh đó, nhóm cũng áp dụng các kỹ thuật đã được học trong môn học này như sử dụng Sprite Groups để phân loại các nhóm, đồng thời tối ưu hóa thuật toán tìm kiếm, khi nào nên dùng thuật toán tìm kiếm này, khi nào không nên dùng thuật toán này, và đồng thời giảm thiểu việc tạo và hủy đối tượng liên tục giúp cải thiện hiệu suất đáng kể.

Thứ tư, lập trình hướng đối tượng (OOP): OOP giúp tổ chức mã nguồn một cách khoa học và hiệu quả, tăng tính module hóa, dễ dàng trong việc bảo trì, nâng cấp. Việc chia trò chơi thành các lớp và đối tượng riêng biệt giúp quản lý các thành phần khác nhau của trò chơi một cách độc lập và dễ dàng thay đổi hoặc thêm mới các chức năng.

Cuối cùng là kỹ năng làm việc nhóm giữa các thành viên, việc phân chia số lượng công việc hiệu quả, dựa trên kỹ năng của từng thành viên cùng với các yếu tố như kỹ năng giao tiếp, phối hợp và giải quyết xung đột để đạt được mục tiêu chung. Việc chia sẻ công việc, thảo luận thiết kế và phối hợp trong quá trình phát triển là rất quan trọng để đảm bảo dự án thành công.

5.3 Hướng phát triển

Bên cạnh những bài học kinh nghiệm được đúc kết khi xây dựng dự án thì nhóm cũng có tìm hiểu các hướng phát triển của dự án nhưng chưa đủ khả năng và thời gian để thực hiện, đó là:

Mở rộng nội dung:

Thêm nhiều màn chơi mới với các thử thách và bối cảnh đa dạng hơn. Giới thiệu thêm các loại quái vật mới với hành vi và kỹ năng độc đáo. Bổ sung thêm các loại vật phẩm và kỹ năng mới cho nhân vật để tăng tính đa dạng trong lối chơi và kéo dài thời gian trải nghiệm. Điều này sẽ thu hút người chơi mới và giúp giữ chân những người chơi cũ, đồng thời cung cấp cho họ nhiều nội dung mới để khám phá và chinh phục trò chơi này.

Phát triển cốt truyện:

Xây dựng một cốt truyện hấp dẫn, có chiều sâu. Thêm vào các nhiệm vụ phụ để làm phong phú thêm thế giới trò chơi và tạo động lực cho người chơi khám phá. Cốt truyện có thể được truyền tải thông qua các đoạn hội thoại giữa nhân vật, các đoạn cắt cảnh hoặc các chi tiết ẩn ý trong yếu tố môi trường trò chơi. Đây cũng là phân đòi hỏi đến khả năng sáng tạo về viết lách và xây dựng kịch bản, một kỹ năng mà nhóm nhận thấy cần trau dồi thêm.

Cải thiện AI của quái vật:

Triển khai các thuật toán AI phức tạp hơn để quái vật trở nên thông minh hơn, tạo ra những thử thách đa dạng và khó đoán hơn cho người chơi. Quái vật có thể sử dụng các chiến thuật khác nhau, biết cách phối hợp với nhau để tấn công, hoặc sở hữu các hành động đặc biệt bất ngờ.

Thêm chế độ chơi nhiều người chơi:

Cho phép nhiều người chơi cùng nhau tham gia khám phá thế giới và hợp tác chiến đấu hoặc thậm chí là cạnh tranh với nhau. Điều này sẽ tăng đáng kể tính tương tác và giá trị chơi lại của trò chơi.

Tối ưu hóa hiệu suất và đồ họa:

Tiếp tục rà soát và cải thiện hiệu suất, đồ họa của trò chơi để mang lại trải nghiệm tốt nhất cho người chơi trên nhiều loại cấu hình thiết bị khác nhau. Điều này có thể bao gồm việc sử dụng các kỹ thuật tối ưu hóa đồ họa tiên tiến hơn, giảm thiểu việc sử dụng tài nguyên bộ nhớ và CPU cũng như hỗ trợ các độ phân giải màn hình khác nhau.

Tài liệu tham khảo

[1] Pygame Developers. Pygame Documentation. Truy cập ngày 6 tháng 5, 2025. Truy cập ở: <https://www.pygame.org/docs/>

[2] "Giải bài toán ràng buộc Arc và AC3," *Toolify.ai*. Truy cập ngày 6 tháng 5, 2025. Truy cập ở: <https://www.toolify.ai/vi/ai-news-vn/gii-bi-ton-rng-buc-arc-v-ac3-3085896>

[3] "8 Puzzle Problem in AI," *AlmaBetter Bytes*. Truy cập ngày 6 tháng 5, 2025. Truy cập ở:

<https://www.almabetter.com/bytes/tutorials/artificial-intelligence/8-puzzle-problem-in-ai>

[4] Eshan Pandey, "AC-3.md," *aima-pseudocode GitHub Repository*. Truy cập ngày 8 tháng 5, 2025. Truy cập ở:

<https://github.com/aimacode/aima-pseudocode/blob/master/md/AC-3.md>