ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO ĐỀ TÀI ỨNG DỤNG GIẢI THUẬT BFS VÀ A* TRONG GAME SOKOBAN

Họ và tên:	Vũ Trung Kiên Bùi Thị Thùy Linh
Học phần:	Nhập môn trí tuệ nhân tạo
Giảng viên hướng dẫn:	GS.TS. Nguyễn Thế Toàn CN. Nguyễn Nhật Tùng

HÀ NỘI - 2025

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO ĐỀ TÀI ỨNG DỤNG GIẢI THUẬT BFS VÀ A* TRONG GAME SOKOBAN

Họ và tên:	Vũ Trung Kiên Bùi Thị Thùy Linh
Học phần:	Nhập môn trí tuệ nhân tạo
Giảng viên hướng dẫn:	GS.TS. Nguyễn Thế Toàn CN. Nguyễn Nhật Tùng

HÀ NỘI - 2025

TÓM TẮT

Trong khuôn khổ môn học Nhập môn trí tuệ nhân tạo, nhóm chúng em lựa chọn đề tài "Ứng dụng giải thuật BFS và A* trong trò chơi Sokoban" nhằm nghiên cứu và triển khai các thuật toán tìm kiếm thông minh trên không gian trạng thái. Trò chơi Sokoban là một bài toán điển hình trong lĩnh vực trí tuệ nhân tạo, yêu cầu người chơi tìm đường đẩy các hộp về đúng vị trí đích trong một bản đồ có nhiều chướng ngại vật. Bài toán này có đặc điểm là không gian trạng thái rộng, nhiều ràng buộc và dễ phát sinh các trường hợp bế tắc.

Trong đề tài, chúng em đã áp dụng thuật toán breadth-first search (BFS) để tìm lời giải tối ưu về số bước đi, và thuật toán A* kết hợp với các hàm lượng giá như manhattan và euclidean distance để tăng hiệu quả tìm kiếm. Bên cạnh đó, đề tài còn thực hiện các cải tiến như phát hiện hộp bị kẹt, tối ưu sinh trạng thái hợp lệ và rút gọn không gian trạng thái nhằm tăng tốc độ giải và tiết kiệm tài nguyên bộ nhớ.

Thông qua việc kết hợp giữa lý thuyết và thực hành, đề tài giúp sinh viên hiểu rõ cách ứng dụng các giải thuật trí tuệ nhân tạo trong môi trường trò chơi, đồng thời rèn luyện tư duy logic, kỹ năng phân tích và thiết kế giải pháp hiệu quả.

LỜI CẢM ƠN

Nhóm chúng em xin bày tỏ lòng biết ơn sâu sắc tới thầy GS.TS Nguyễn Thế Toàn – giảng viên phụ trách môn học Nhập môn trí tuệ nhân tạo, người đã tận tình hướng dẫn, hỗ trợ và truyền cảm hứng cho nhóm trong suốt quá trình thực hiện đề tài. Những định hướng, góp ý chuyên môn cùng sự khích lệ của thầy là động lực quan trọng giúp nhóm hoàn thành tốt môn học này.

Dù đã cố gắng nghiên cứu và thực hiện với tinh thần nghiêm túc, nhưng do kiến thức và kinh nghiệm thực tiễn còn hạn chế, đề tài chắc chắn vẫn còn những thiếu sót. Nhóm chúng em rất mong nhận được sự góp ý của thầy để tiếp tục hoàn thiện và nâng cao chất lượng công trình.

Một lần nữa, chúng em xin chân thành cảm ơn giảng viên GS.TS Nguyễn Thế Toàn. Kính chúc thầy luôn mạnh khỏe, hạnh phúc và thành công trong sự nghiệp giáo dục.

Trân trọng cảm ơn!

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1 Giới thiệu chung

1.1.1. Tính cấp thiết của đề tài

Trong bối cảnh trí tuệ nhân tạo ngày càng phát triển và được ứng dụng rộng rãi trong nhiều lĩnh vực như giao thông, y tế, sản xuất và đặc biệt là trò chơi, việc nghiên cứu và áp dụng các thuật toán tìm kiếm trong môi trường mô phỏng ngày càng trở nên quan trọng. Trò chơi Sokoban là một bài toán kinh điển trong lĩnh vực trí tuệ nhân tạo, không chỉ mang tính giải trí mà còn mang tính chất mô hình hóa những bài toán tìm đường có ràng buộc phức tạp trong không gian trạng thái lớn.

Việc lựa chọn Sokoban làm đối tượng nghiên cứu là hoàn toàn phù hợp trong việc giúp sinh viên tiếp cận, hiểu và vận dụng các giải thuật như breadth-first search (BFS) và A* vào bài toán thực tiễn. Đây là hai trong số những thuật toán nền tảng, có giá trị ứng dụng cao trong thiết kế hệ thống thông minh, đặc biệt là trong các bài toán điều hướng và tối ưu hóa.

Đề tài không chỉ giúp người học nắm vững kiến thức lý thuyết mà còn rèn luyện kỹ năng phân tích, thiết kế và hiện thực hóa các thuật toán trên môi trường giả lập. Đồng thời, quá trình triển khai đề tài còn cho thấy rõ vai trò của việc tối ưu không gian tìm kiếm và sử dụng heuristic hiệu quả, góp phần giải quyết bài toán nhanh hơn và sử dụng ít tài nguyên hơn.

Với những lý do nêu trên, việc nghiên cứu và ứng dụng các giải thuật tìm kiếm trong trò chơi Sokoban là một đề tài cần thiết và có ý nghĩa thực tiễn trong việc học tập và phát triển tư duy thuật toán của sinh viên ngành công nghệ thông tin, đặc biệt trong lĩnh vực trí tuệ nhân tạo

1.1.2. Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là quá trình giải quyết bài toán tìm đường trong trò chơi Sokoban bằng cách áp dụng các thuật toán tìm kiếm trong trí tuệ nhân tạo, cụ thể là thuật toán breadth-first search (BFS) và thuật toán A*.

Cụ thể, đề tài tập trung vào:

- Mô hình hóa không gian trạng thái trong Sokoban, bao gồm vị trí người chơi, hộp và đích.
- Phân tích cách hoạt động và hiệu quả của các giải thuật tìm kiếm trong việc tìm ra chuỗi hành động tối ưu để đưa tất cả các hộp đến đúng vị trí đích.
- Thiết kế và triển khai hàm heuristic phù hợp (như khoảng cách manhattan, khoảng cách euclidean) trong giải thuật A*.
- So sánh, đánh giá hiệu quả giữa BFS và A* trong các bản đồ Sokoban có độ phức tạp khác nhau.
- Tối ưu hóa quá trình tìm kiếm bằng cách loại bỏ các trạng thái không hợp lệ, phát hiện deadlock, và rút gọn không gian trạng thái.

Thông qua việc nghiên cứu các yếu tố trên, đề tài hướng tới việc làm rõ cách thức các thuật toán trí tuệ nhân tạo có thể được ứng dụng trong các môi trường giả lập có tính logic cao như trò chơi Sokoban.

1.1.3. Ý nghĩa thực tiễn và khoa học

Về mặt khoa học, đề tài góp phần cụ thể hóa việc ứng dụng các thuật toán tìm kiếm trong trí tuệ nhân tạo vào bài toán thực tế dưới dạng trò chơi. Việc phân tích và triển khai các thuật toán như breadth-first search (BFS) và A* giúp sinh viên hiểu sâu hơn về nguyên lý hoạt động của các thuật toán này, cũng như cách xây dựng hàm lượng giá (heuristic) để nâng cao hiệu quả tìm kiếm. Bên cạnh đó, đề tài còn giúp làm rõ các khái niệm quan trọng trong AI như không gian trạng thái, trạng thái mục tiêu, tối ưu hóa tìm kiếm và ràng buộc logic trong hành vi tác tử.

Về mặt thực tiễn, việc ứng dụng các giải thuật tìm kiếm vào trò chơi Sokoban không chỉ mang tính học thuật mà còn phản ánh các bài toán thực tế trong nhiều lĩnh vực như logistics, robot vận chuyển, điều hướng trong không gian hẹp, và thiết kế hệ thống tự động. Sokoban là mô hình đơn giản nhưng đầy thách thức, tương tự như các bài toán sắp xếp, điều khiển và ra quyết định trong môi trường có ràng buộc. Kết quả của đề tài không chỉ hỗ trợ quá trình học tập, mà còn có thể là nền tảng cho các nghiên cứu mở rộng sau này trong lĩnh vực tự động hóa và hệ thống thông minh.

1.1.4. Phương pháp nghiên cứu

Đề tài được thực hiện theo hướng tiếp cận kết hợp giữa lý thuyết và thực hành, thông qua các bước nghiên cứu và triển khai như sau:

- Nghiên cứu tài liệu: Tìm hiểu các khái niệm cơ bản của trí tuệ nhân tạo, đặc biệt là các thuật toán tìm kiếm trong không gian trạng thái như breadth-first search (BFS) và A*. Đồng thời, nghiên cứu cấu trúc và quy tắc của trò chơi Sokoban để hiểu rõ bài toán cần giải quyết.
- Phân tích bài toán Sokoban: Mô hình hóa không gian trạng thái của trò chơi, xác định các thành phần chính như trạng thái ban đầu, trạng thái mục tiêu, tập hành động hợp lệ, điều kiện thắng và các ràng buộc logic (như hộp bị kẹt, di chuyển không hợp lệ...).
- Thiết kế và cài đặt thuật toán: Cài đặt hai thuật toán BFS và A* để giải quyết bài toán tìm đường trong Sokoban. Với A*, xây dựng các hàm lượng giá phù hợp như manhattan distance và euclidean distance để đánh giá hiệu quả heuristic.
- Tối ưu hóa không gian trạng thái: Thực hiện các kỹ thuật giảm thiểu trạng thái không cần thiết bằng cách phát hiện deadlock, tránh sinh trạng thái trùng lặp và loại bỏ các bước di chuyển không hợp lệ.
- Thử nghiệm và đánh giá: Thử nghiệm chương trình trên nhiều bản đồ Sokoban với độ phức tạp khác nhau. So sánh kết quả tìm được giữa hai thuật toán về số bước đi, thời gian xử lý và hiệu quả bộ nhớ.
- Rút ra kết luận: Phân tích ưu, nhược điểm của từng thuật toán trong từng tình huống, từ đó đề xuất hướng cải tiến và mở rộng cho các nghiên cứu sau.

1.2. Giới thiệu về Visual Code Studio

Visual Studio Code (VS Code) là một môi trường phát triển tích hợp nhẹ (lightweight IDE), mã nguồn mở và đa nền tảng, được phát triển bởi Microsoft và chính thức ra mắt vào năm 2015. Trong vòng chưa đầy một thập kỷ, VS Code đã trở thành một trong những trình soạn thảo mã nguồn phổ biến nhất thế giới, được hàng triệu lập trình viên và nhóm phát triển phần mềm tin dùng. Sự phổ biến của VS Code đến từ khả năng mở rộng linh hoạt, giao diện thân thiện, hiệu năng cao, và đặc biệt là tính năng hỗ trợ lập trình hiện đại nhưng không làm người dùng cảm thấy cồng kềnh như những IDE truyền thống.

Khác với các trình soạn thảo đơn thuần như Notepad++ hay Sublime Text, VS Code cung cấp một loạt các chức năng tương đương với một IDE đầy đủ như IntelliJ hay Visual Studio, nhưng có dung lượng nhẹ và khả năng hoạt động mượt mà trên cả các máy tính cấu hình trung bình. Điều này giúp VS Code đặc biệt phù hợp với môi trường giáo dục, nghiên cứu, hoặc phát triển các dự án vừa và nhỏ—đặc biệt là các đề tài sinh viên như *thiết kế trò chơi "Sokoban" sử dụng thuật toán BFS và A**.

Visual Studio Code

Hình 1: Ứng dụng Visual Studio Code

1.2.1. Ưu điểm

- Đa dụng và mở rộng linh hoạt: VS Code hỗ trợ nhiều ngôn ngữ và công nghệ thông qua hệ thống tiện ích mở rộng (extensions). Người dùng có thể lập trình web, game, AI, hay backend chỉ với một công cụ duy nhất.
- Nhẹ, miễn phí, dễ sử dụng: VS Code có dung lượng nhỏ, cài đặt nhanh chóng, hoàn toàn miễn phí và tương thích với hầu hết hệ điều hành phổ biến (Windows, macOS, Linux).
- Hỗ trợ lập trình thông minh: Với IntelliSense, tự động hoàn thành mã và gợi ý cú pháp giúp tăng tốc độ và độ chính xác khi lập trình.
- Tích hợp Git và terminal: Giúp lập trình viên làm việc hiệu quả hơn mà không cần rời khỏi trình soạn thảo.
- Cộng đồng mạnh mẽ: VS Code được cập nhật thường xuyên và có cộng đồng hỗ trợ đông đảo.

1.2.2. Nhược điểm

- Thiếu chuyên sâu: So với các IDE chuyên dụng (như PyCharm, Visual Studio, IntelliJ), VS Code không có nhiều công cụ tích hợp sâu phục vụ cho các dự án lớn.
- Phím tắt mặc định khó nhớ: Người dùng thường phải cấu hình lại các tổ hợp phím để phù hợp với thói quen cá nhân.
- Tiêu tốn tài nguyên: Là ứng dụng Electron nên VS Code tiêu thụ RAM và pin tương đối cao, đặc biệt khi chạy nhiều extension.

1.2.3. Tính năng kỹ thuật nổi bật của VS Code

- Hỗ trợ đa ngôn ngữ lập trình: VS Code hỗ trợ hàng chục ngôn ngữ lập trình như JavaScript, Python, C/C++, Java, Rust, Go... nhờ vào hệ thống extension phong phú từ cộng đồng và chính Microsoft cung cấp.
- Tự động hoàn thành mã và gợi ý thông minh (IntelliSense): Giúp lập trình viên tiết kiệm thời gian và giảm lỗi cú pháp nhờ khả năng gợi ý tên hàm, biến, kiểu dữ liệu, hoặc đoạn mã thông minh dựa trên ngữ cảnh
- Tích hợp Git và hệ thống quản lý phiên bản: Cho phép người dùng thực hiện commit, push, pull, merge và theo dõi lịch sử thay đổi mã nguồn trực tiếp trong giao diện làm việc, giúp đồng bộ và làm việc nhóm hiệu quả hơn.
- Debugging toàn diện: Cung cấp khả năng gỡ lỗi trực tiếp trong trình soạn thảo với breakpoint, call stack, watch variables và điều hướng dòng thực thi một cách trực quan.
- Terminal tích hợp: Cho phép mở và chạy terminal shell trực tiếp bên trong VS Code (bash, PowerShell, cmd...) mà không cần chuyển qua công cụ ngoài.
- Khả năng mở rộng và tùy biến cao: Với hàng chục ngàn extension có sẵn trên Marketplace, người dùng có thể cá nhân hóa từ theme giao diện đến công cụ hỗ trợ đặc thù cho từng loại dự án.

1.2.4. Lợi ích khi sử dụng VS Code trong phát triển game học thuật

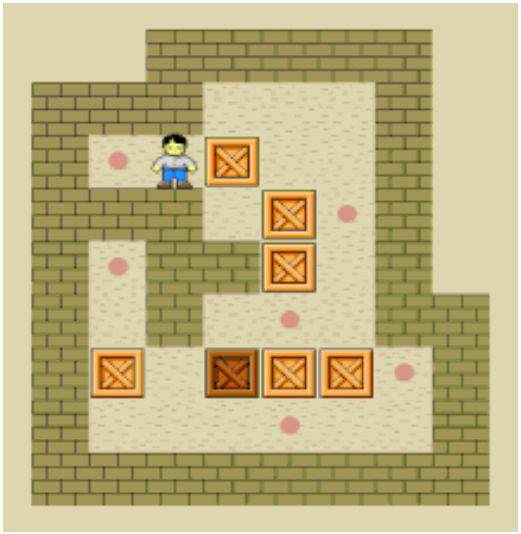
- Trong khuôn khổ đề tài nghiên cứu và phát triển trò chơi đơn giản như "Sokoban" sử dụng thuật toán tìm đường BFS và A*, VS Code là một công cụ hoàn toàn phù hợp với các yêu cầu về trực quan hóa thuật toán, thử nghiệm logic, xử lý sự kiện người dùng và kết nối giữa các module giao diện – xử lý – hiển thị.
- Lập trình linh hoạt: Có thể dùng VS Code để lập trình bằng Python, JavaScript, hoặc C++ tuỳ vào lựa chọn của nhóm thực hiện đề tài.

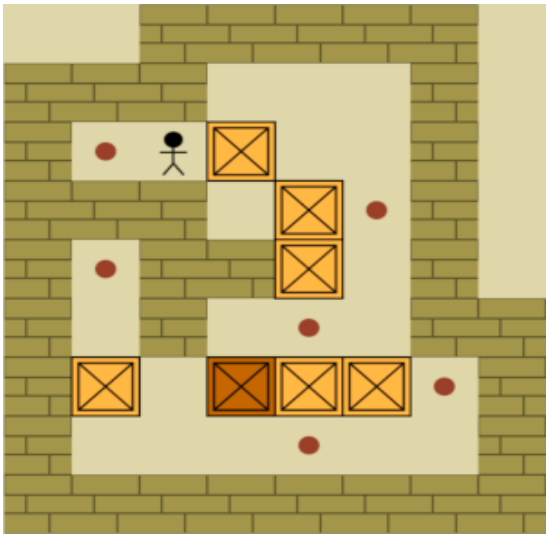
- Dễ tích hợp thư viện bên ngoài: Các thư viện như pygame, tkinter (Python) hay p5.js, canvas (JavaScript) đều có thể được cài đặt và sử dụng dễ dàng trong VS Code.
- Hỗ trợ teamwork và quản lý tiến độ: Khi kết hợp với GitHub, VS Code tạo điều kiện cho làm việc nhóm hiệu quả, đặc biệt trong môi trường học thuật nơi nhiều sinh viên cùng phát triển một sản phẩm.
- Trực quan hóa thuật toán: VS Code cho phép chạy thử nhanh và dễ dàng kiểm tra các thay đổi trong thuật toán A* mà không cần build toàn bộ chương trình như ở các IDE nặng nề hơn.
- Tính ứng dụng cao và phù hợp với xu hướng phát triển phần mềm hiện đại : Ngoài việc phù hợp cho các đề tài nhỏ, VS Code còn đang được sử dụng bởi các công ty công nghệ lớn trong các dự án thực tế, đặc biệt là trong lĩnh vực phát triển web, ứng dụng di động, AI, và game 2D đơn giản. Điều này chứng minh rằng việc học và làm việc trên VS Code không chỉ giúp hoàn thành tốt đề tài học thuật mà còn tạo nền tảng vững chắc cho các công việc kỹ thuật sau này.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Giới thiệu về trò chơi

Trò chơi Sokoban là một trò chơi logic cổ điển, xuất hiện đầu tiên vào những năm 1980 tại Nhật Bản. Trong trò chơi này, người chơi đóng vai một công nhân cố gắng di chuyển các hộp đồ đến các vị trí mục tiêu trên một bản đồ lươn lẹo và có rất ít không gian để làm việc. Trò chơi tưởng chừng đơn giản nhưng đòi hỏi sự tư duy chiến lược và lập kế hoạch cẩn thận. Người chơi phải sắp xếp và di chuyển các hộp đồ sao cho chúng đặt đúng vị trí mục tiêu mà không bị kẹt cùng với hộp hoặc tường.





Cách chơi: Trò chơi Sokoban là một trò chơi logic thú vị trong đó bạn phải di chuyển nhân vật để đẩy các hộp vào các vị trí đích trên một lưới. Dưới đây là cơ chế cơ bản để chơi trò chơi Sokoban:

- Mục tiêu: Mục tiêu của trò chơi Sokoban là đẩy tất cả các hộp (được ký hiệu là 'B') vào các ô đích (được ký hiệu là 'G') trên lưới.
- Thành phần chính:

1.Nhân vật (Player): Bạn điều khiển một nhân vật trên lưới, thường được ký hiệu bằng 'P'.

2.Hộp (Box): Hộp là đối tượng bạn cần đẩy vào các ô đích. Chúng thường được ký hiệu bằng 'B'.

3.Ô đích (Goal): Ô đích là nơi bạn cần đẩy các hộp tới. Chúng thường được ký hiệu b

2.2. Ứng dụng trí tuệ nhân tạo trong trò chơi Sokoban

Trí tuệ nhân tạo đã đánh dấu sự tiến bộ lớn trong việc tạo ra các trình điều khiển tự động cho trò chơi Sokoban. Thay vì chỉ dựa vào người chơi con người, chúng ta có thể sử dụng các thuật toán và hệ thống Trí tuệ nhân tạo để làm cho trò chơi này tự động giải quyết. Sự kết hợp giữa lập trình và Trí tuệ nhân tạo cho phép chúng ta tạo ra các "người máy" có khả năng tư duy logic và giải quyết các cấp độ trong trò chơi Sokoban một cách hiệu quả.

Ứng dụng Trí tuệ nhân tạo trong Sokoban có nhiều ứng dụng thú vị. Chúng có thể được sử dụng để:

- Tạo trình điều khiển tự động cho trò chơi Sokoban, giúp giải quyết các cấp độ mà người chơi có thể gặp khó khăn.
- Nghiên cứu và phát triển các thuật toán tối ưu để di chuyển hộp đồ và đạt được mục tiêu trong thời gian ngắn nhất.
- Hiểu rõ cách mà máy tính có thể áp dụng kiến thức trong quá trình học máy để cải thiện hiệu suất chơi Sokoban.

2.3 Một số khái niệm

2.3.1. Dạng trò chơi

Các trò chơi có dạng như sokoban,… là những trò chơi đòi hỏi trí tuệ và thường được thiếp lập như sau:

+ Bản đồ (Map): 1.Trò chơi thường diễn ra trên một bản đồ gồm các ô vuông. Mỗi ô có một chức năng cụ thể:

- Không gian trống (Floor): Được đại diện bởi các ô trống, mà người chơi và hộp có thể di chuyển vào.
- Tường (Wall): Được đại diện bởi các ô chứa tường, mà không thể di chuyển vào.
- Mục tiêu (Goal): Là các ô mà người chơi cần di chuyển hộp đồ tới. Mục tiêu thường được ký hiệu bằng các điểm.
- Hộp đồ (Box): Là các hộp cần di chuyển đến các ô là mục tiêu.

+ Người chơi (Player): Người chơi điều khiển một nhân vật (thường được biểu thị bằng một biểu tượng) để di chuyển các hộp đồ từ vị trí ban đầu đến các ô là mục tiêu.

+ Quy tắc di chuyển:

- Người chơi có thể di chuyển lên, xuống, trái, hoặc phải, nhưng chỉ được di chuyển vào các ô trống.
- Người chơi có thể đẩy hộp đồ bằng cách đứng bên cạnh hộp và tiến vào hộp, đẩy nó trong hướng tương ứng. Hộp không thể di chuyển qua các ô tường hoặc qua hộp khác.

+ Mục tiêu của trò chơi: Mục tiêu của trò chơi là di chuyển tất cả các hộp đồ từ vị trí ban đầu đến các ô là mục tiêu, đặt chính xác trên tất 4.cả các mục tiêu. Người chơi cần phải tìm cách sắp xếp các hộp để đạt được mục tiêu này.

Trò chơi Sokoban có nhiều cấp độ khó khăn khác nhau và thường đòi hỏi người chơi phải suy nghĩ chiến lược, vận dụng logic và lập kế hoạch cẩn thận để giải quyết các câu đố. Các thuật toán Trí tuệ nhân tạo cũng có thể được sử dụng để tạo trình điều khiển tự động để giải quyết trò chơi Sokoban hoặc để phát triển các chiến lược thông minh cho người chơi.

2.3.2 Cây trò chơi

Cây trò chơi trong Sokoban, còn gọi là "cây tìm kiếm trò chơi," là một phần quan trọng của việc áp dụng Trí Tuệ Nhân Tạo vào trò chơi Sokoban. Cây trò chơi đại diện cho toàn bộ không gian trạng thái của trò chơi và tất cả các biểu diễn có thể có của nó. Đây là một khía cạnh quan trọng khi bạn cố gắng giải quyết một trò chơi Sokoban bằng cách sử dụng các thuật toán tìm kiếm.

Dưới đây là một số khái niệm và thông tin liên quan đến cây trò chơi trong Sokoban:

1. Trạng thái (State) : Mỗi nút trên cây trò chơi đại diện cho một trạng thái cụ thể của trò chơi. Trạng thái này bao gồm vị trí của người chơi, vị trí của các hộp đồ, vị trí của các mục tiêu và cấu trúc bản đồ.
2. Nút (Node): Mỗi nút trên cây trò chơi là một trạng thái cụ thể của trò chơi tại một thời điểm cụ thể. Cây trò chơi bắt đầu với một nút gốc, đại diện cho trạng thái ban đầu của trò chơi.
3. Cây Trò Chơi (Game Tree): Cây trò chơi là một biểu đồ cây có cấu trúc, trong đó mỗi nút biểu thị một trạng thái của trò chơi, và các cạnh biểu thị các hành

động mà người chơi có thể thực hiện để chuyển từ một trạng thái sang một trạng thái khác.

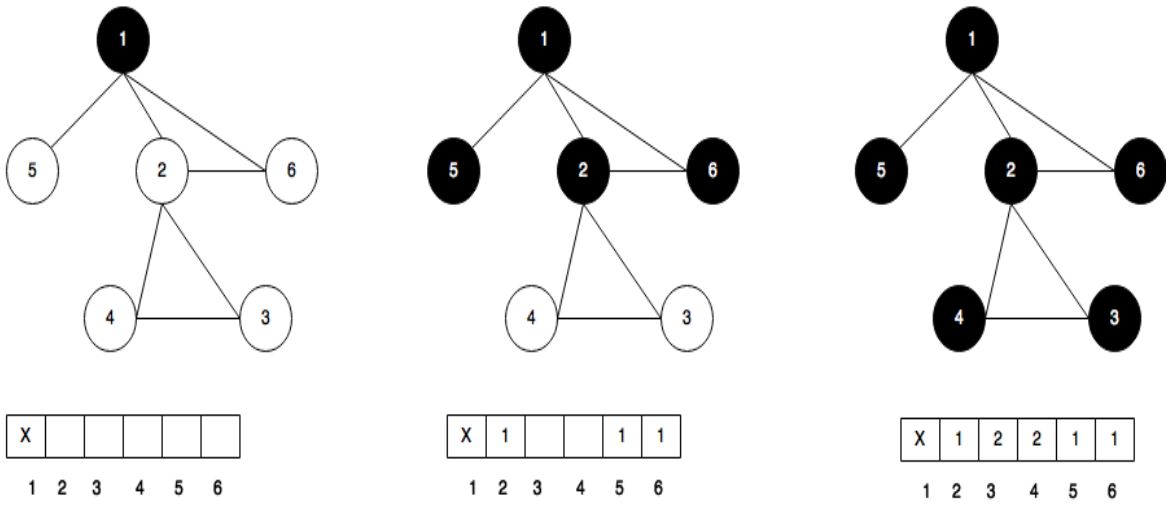
4. Trạng Thái Bắt Đầu (Initial State): Trạng thái ban đầu của trò chơi, thường chứa thông tin về vị trí ban đầu của người chơi, các hộp đồ, và các mục tiêu.
5. Trạng Thái Kết Thúc (Goal State): Trạng thái mà bạn cố gắng đạt được trong trò chơi Sokoban, trong đó tất cả các hộp đồ đã được di chuyển đến các mục tiêu.
6. Hành Động (Action): Các hành động mà người chơi có thể thực hiện để thay đổi trạng thái của trò chơi, bao gồm di chuyển lên, xuống, trái hoặc phải, hoặc đẩy hộp đồ.
7. Thuật Toán Tìm Kiếm (Search Algorithm): Trong cây trò chơi, thuật toán tìm kiếm được sử dụng để duyệt qua các nút trạng thái để tìm ra một dãy hành động dẫn đến trạng thái kết thúc. Thuật toán tìm kiếm có thể làm cho việc giải quyết trò chơi Sokoban tự động trở nên hiệu quả.
8. Cắt Tự Nhiên (Pruning): Cắt tự nhiên là quá trình loại bỏ các nút trong cây trò chơi mà không cần thiết để giảm bớt độ phức tạp của thuật toán tìm kiếm.

Cây trò chơi trong Sokoban là một phần quan trọng của việc áp dụng trí tuệ nhân tạo để giải quyết trò chơi này. Nó giúp bạn biểu diễn và tìm hiểu toàn bộ không gian trạng thái của trò chơi và tìm ra cách tối ưu để đạt được trạng thái kết thúc.

2.4. Các giải thuật tìm kiếm

2.4.1. Giải thuật BFS

Thuật toán Breadth-First Search (BFS) là một thuật toán tìm kiếm không có trọng số được sử dụng để duyệt và tìm kiếm trạng thái hoặc nút trong một đồ thị hoặc cây. BFS bắt đầu từ nút gốc (nút ban đầu) và duyệt toàn bộ đồ thị theo chiều rộng, tức là duyệt qua tất cả các nút cùng cấp trước khi di chuyển xuống cấp tiếp theo. Thuật toán này thường được sử dụng để tìm đường đi ngắn nhất giữa hai nút trong đồ thị hoặc cây không có trọng số.



Dưới đây là mô tả cụ thể của thuật toán BFS:

1. Khởi tạo: Bắt đầu từ nút gốc (nút ban đầu) và đặt nó vào hàng đợi (queue).
2. Duyệt và mở rộng: Lặp qua các bước sau đây cho đến khi hàng đợi trống: a. Lấy nút đầu tiên ra khỏi hàng đợi. b. Kiểm tra xem nút này có phải là mục tiêu (nút kết thúc) hay không. Nếu có, quá trình kết thúc và bạn đã tìm thấy lời giải (đường đi) nếu mục tiêu là mục đích của tìm kiếm. c. Nếu nút không phải là mục tiêu, mở rộng nút này bằng cách thêm tất cả các nút con của nó (các nút kề) vào hàng đợi (queue).
3. Lặp lại: Quay lại bước 2 để tiếp tục duyệt và mở rộng các nút khác trong hàng đợi.
4. Kết thúc: Khi hàng đợi trống, nếu không tìm thấy mục tiêu, thuật toán đã hoàn thành và thông báo rằng không có đường đi nào tới mục tiêu.

BFS đảm bảo tìm đường đi ngắn nhất trong đồ thị vô hướng không có trọng số. Nó cũng thường được sử dụng trong các bài toán tìm kiếm đường đi, kiểm tra kết nối giữa các đối tượng, và phân tích cấu trúc đồ thị.

Một lưu ý quan trọng là BFS tiêu tốn nhiều bộ nhớ khi tìm kiếm trong các đồ thị lớn với số lượng lớn nút con.

*** Trong game Sokoban, chiến lược tìm kiếm BFS như sau:**

BFS là một thuật toán tìm kiếm không có trọng số, nó duyệt qua tất cả các trạng thái cùng mức trước khi di chuyển xuống cấp tiếp theo. Trong Sokoban, nó có thể được sử dụng để tìm một đường đi từ vị trí ban đầu đến mục tiêu. Tuy nhiên, BFS không xem xét chi phí của việc đi lại nên nó không đảm bảo tìm đường đi ngắn nhất.

Để sử dụng BFS trong Sokoban, bạn bắt đầu từ vị trí ban đầu của người chơi và duyệt qua tất cả các trạng thái con có thể đạt được từ trạng thái hiện tại. Bạn tiếp tục duyệt qua các trạng thái mới được tạo ra cho đến khi bạn tìm thấy mục tiêu hoặc bạn đã duyệt qua tất cả các trạng thái mà không tìm thấy lời giải.

2.4.2. Giải thuật A*

Thuật toán A* là một thuật toán tìm kiếm thông minh (intelligent search) được sử dụng để tìm đường đi tối ưu từ một trạng thái ban đầu đến trạng thái kết thúc trong một không gian trạng thái có cấu trúc. Thuật toán này thường được áp dụng trong các bài toán tìm kiếm đường đi trong trò chơi, lập trình robot tự lái, quá trình quyết định trong trí tuệ nhân tạo, và nhiều lĩnh vực khác. A* sử dụng một hàm heuristic để ước tính chi phí tối ưu từ trạng thái hiện tại đến trạng thái kết thúc, và nó duyệt qua các trạng thái tiềm năng theo thứ tự ước tính chi phí thấp nhất đầu tiên.

Dưới đây là mô tả cụ thể về cách thuật toán A* hoạt động:

1. Khởi tạo: Bắt đầu từ trạng thái ban đầu, tạo một danh sách có trạng thái ban đầu và gán chi phí gốc (g) cho nó là 0.
2. Danh Sách Mở (Open List): Tạo một danh sách mở để lưu trữ các trạng thái chưa được duyệt. Đưa trạng thái ban đầu vào danh sách này.
3. Danh Sách Đóng (Closed List): Tạo một danh sách đóng để lưu trữ các trạng thái đã được duyệt.
4. Lặp cho đến khi danh sách mở trống:

- Chọn trạng thái (nút) có ước tính chi phí thấp nhất (f) từ danh sách mở.
- Di chuyển trạng thái này từ danh sách mở sang danh sách đóng.
- Kiểm tra nếu trạng thái này là trạng thái kết thúc. Nếu có, dừng thuật toán và trả về đường đi tối ưu.
- Duyệt qua các trạng thái con (nếu có) của trạng thái hiện tại:
    - Tính toán chi phí g từ trạng thái ban đầu đến trạng thái con thông qua trạng thái hiện tại.
    - Tính toán chi phí ước tính h từ trạng thái con đến trạng thái kết thúc bằng cách sử dụng hàm heuristic.
    - Tính toán chi phí tổng f = g + h.

- Nếu trạng thái con không nằm trong danh sách mở hoặc chi phí f từ trạng thái ban đầu đến trạng thái con thấp hơn chi phí đã biết, cập nhật chi phí và thêm trạng thái con vào danh sách mở.

5. Nếu danh sách mở trống mà không tìm thấy đường đi đến trạng thái kết thúc, thì không có đường đi tới trạng thái đó.

Thuật toán A* thường được sử dụng trong trò chơi Sokoban và nhiều ứng dụng khác để tìm đường đi tối ưu và đảm bảo hiệu suất tối ưu. Hàm heuristic trong thuật toán A* là một yếu tố quan trọng, và nó cần được thiết kế sao cho ước tính chi phí từ trạng thái hiện tại đến trạng thái kết thúc là thực tế và không quá lạc hậu.

*** Chiến lược tìm kiếm A* trong Sokoban:**

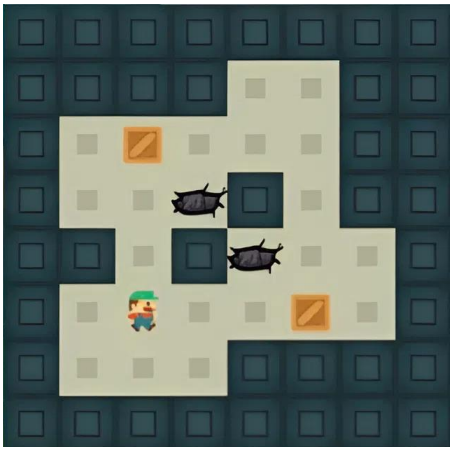
- A* Search là một chiến lược tìm kiếm thông minh sử dụng hàm heuristic để ước tính chi phí từ trạng thái hiện tại đến trạng thái kết thúc.
- A* Search duyệt qua các trạng thái con theo thứ tự ước tính chi phí tổng thấp nhất (f = g + h), với g là chi phí đã biết và h là ước tính chi phí từ trạng thái con đến trạng thái kết thúc.
- Hàm Heuristic Euclidean (Khoảng cách Euclide): Hàm heuristic Euclidean tính khoảng cách theo đường thẳng giữa hai điểm trên bản đồ. Trong Sokoban, nó có thể được sử dụng để đo khoảng cách từ vị trí hiện tại của hộp đến mục tiêu gần nhất. Hàm này đơn giản và cho kết quả tốt nếu không có rào cản trên đường đi.
- Hàm Heuristic Manhattan (Khoảng cách Manhattan): Hàm heuristic Manhattan tính khoảng cách bằng cách cộng tất cả các khoảng cách theo chiều ngang và chiều dọc giữa hai điểm. Trong Sokoban, nó có thể được sử dụng để đo khoảng cách từ vị trí hiện tại của hộp đến mục tiêu gần nhất. Hàm này đưa ra kết quả chính xác trong môi trường có các hộp và rào cản.
- A* sẽ tìm kiếm theo chiều rộng, ưu tiên các trạng thái có tổng chi phí nhỏ nhất. Khi nó tìm thấy một đường đi tiềm năng, nó sẽ tiếp tục mở rộng và kiểm tra các trạng thái con cho đến khi nó tìm thấy một đường đi hoặc xác định rằng không có đường đi nào tồn tại.

Kết quả là, A* kết hợp với hàm heuristic Euclidean hoặc Manhattan giúp tìm ra đường đi tối ưu trong trò chơi Sokoban một cách hiệu quả và nhanh chóng bằng cách định rõ mục tiêu và ước tính khoảng cách còn lại đến mục tiêu.

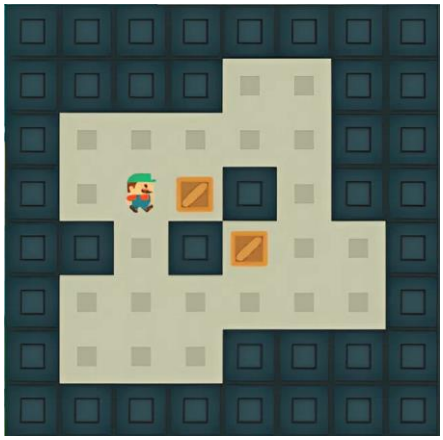
CHƯƠNG 3: THIẾT KẾ PHÁT TRIỂN TRÒ CHƠI

3.1 Tổng quan

Để mang đến một trò chơi thú vị đơn giản nhưng cũng không kém phần thử thách nhóm em đã tạo ra 30 màn chơi khác nhau.



(map ban đầu)



(map hoàn thành)

Đây chỉ là một màn đơn giản của trò chơi chúng ta có thể thử các map còn lại để có thể hiểu và đánh giá rõ hơn về các thuật toán tìm kiếm không gian.

3.2. Mô hình hóa các map trong trò chơi

Để lập mô hình trò chơi này, chúng em có một ký hiệu tiêu chuẩn được sử dụng để phân biệt độc lập và rõ ràng giữa tất cả các đối tượng trong trò chơi. Mỗi cấp độ được đưa ra dưới dạng biểu diễn duy nhất. Dữ liệu đầu vào để lập mô hình bao gồm các ký tự sau:

- 'P' cho vị trí của người chơi.
- '#' cho các tường rào hoặc vị trí không thể đi qua.
- '.' cho các vị trí mục tiêu.
- 'B' hoặc 'Box' cho hộp cần đẩy.
- ' ' (khoảng trắng) cho vị trí trống.

3.3. Hệ thống trò chơi

3.3.1 Các thư viện được áp dụng

Các thư viện chính được sử dụng trong đề tài là :numpy, pygame, colorama

- Thư viện NumPy (Numerical Python) là một thư viện mạnh mẽ và phổ biến trong Python dùng cho tính toán số học và xử lý dữ liệu đa chiều. NumPy cho phép bạn làm việc với mảng và ma trận nhanh chóng và hiệu quả, giúp thực hiện các phép toán số học, thống kê, và xử lý dữ liệu trong Python dễ dàng hơn, NumPy là một thư viện quan trọng và mạnh mẽ trong Python, thường được sử dụng trong nhiều ứng dụng khoa học dữ liệu, máy học, và xử lý ảnh, bao gồm cả việc xử lý dữ liệu trong trò chơi như Sokoban.
- Thư viện NumPy (Numerical Python) thường không được sử dụng trực tiếp để giải trò chơi Sokoban, mà thay vào đó, nó có thể được sử dụng để quản lý trạng thái và dữ liệu trong quá trình giải quyết Sokoban bằng thuật toán BFS hoặc A*.

(thư viện numpy biểu diễn trạng thái ban đầu và thực hiện di chuyển của người chơi.)

```
import numpy as np

# Biểu diễn trạng thái của màn chơi Sokoban bằng NumPy array
initial_state = np.array([
    ['#', '#', '#', '#', '#', '#'],
    ['#', ' ', ' ', ' ', ' ', '#'],
    ['#', ' ', '$', ' ', '@', '#'],
    ['#', ' ', ' ', ' ', ' ', '#'],
    ['#', '#', '#', '#', '#', '#']
])

# Dùng ma trận NumPy để thực hiện các phép toán
# Ví dụ: Di chuyển người chơi
new_state = np.copy(initial_state)  # Tạo bản sao của trạng thái ban đầu
player_position = np.argwhere(new_state == '@')[0]  # Tìm vị trí của người c
new_position = player_position + np.array([0, 1])  # Di chuyển người chơi sa
new_state[player_position[0], player_position[1]] = ' '  # Đánh dấu ô trống
new_state[new_position[0], new_position[1]] = '@'  # Đánh dấu vị trí mới của
```

- Thư viện Pygame là một thư viện phát triển trò chơi và đồ họa trong ngôn ngữ lập trình Python. Pygame cung cấp một cơ sở hạ tầng để tạo ra các ứng dụng đa phương

tiện, đặc biệt là trò chơi. Nó hỗ trợ đồ họa, âm thanh, nhập liệu và cung cấp các công cụ để xây dựng các trò chơi trên nền tảng đa nền tảng.

Sử dụng Pygame để tạo một màn chơi Sokoban :

1. Cài đặt và khởi tạo Pygame

```
pip install pygame
```

2. Tạo màn hình và biểu diễn trạng thái:

```
//                                    //
//========================//
'''
pygame.init()
pygame.font.init()
screen = pygame.display.set_mode((640, 640))
pygame.display.set_caption('Sokoban')
clock = pygame.time.Clock()
BACKGROUND = (0, 0, 0)
WHITE = (255, 255, 255)
'''
```

```
def renderMap(board):
    width = len(board[0])
    height = len(board)
    indent = (640 - width * 32) / 2.0
    for i in range(height):
        for j in range(width):
            screen.blit(space, (j * 32 + indent, i * 32 + 250))
            if board[i][j] == '#':
                screen.blit(wall, (j * 32 + indent, i * 32 + 250))
            if board[i][j] == '$':
                screen.blit(box, (j * 32 + indent, i * 32 + 250))
            if board[i][j] == '%':
                screen.blit(point, (j * 32 + indent, i * 32 + 250))
            if board[i][j] == '@':
                screen.blit(player, (j * 32 + indent, i * 32 + 250))
```

3. Vẽ trạng thái màn hình chơi

```
''' HÀM SOKOBAN '''
start_time = 0
end_time = 0
steps = 0
initial_memory = 0
def sokoban():
    running = True
    global sceneState
    global loading
    global algorithm
    global list_board
    global mapNumber
    stateLength = 0
    currentState = 0
    found = True
    while running:
        screen.blit(init_background, (0, 0))
        if sceneState == "init":
            initGame(maps[mapNumber])
        if sceneState == "executing":
            list_check_point = check_points[mapNumber]
            # Bắt đầu đếm thời gian
            start_time = time.time()
            if algorithm == "Euclidean Distance Heuristic":
                list_board = astar1.AStar_Search1(maps[mapNumber], list_check_point)
            elif algorithm == "Manhattan Distance Heuristic":
                list_board = astar.AStar_Search(maps[mapNumber], list_check_point)
            else:
                list_board = bfs.BFS_search(maps[mapNumber], list_check_point)
            # Dừng đếm thời gian
            end_time = time.time()
            if len(list_board) > 0:
                sceneState = "playing"
                stateLength = len(list_board[0])
                currentState = 0
                elapsed_time = end_time - start_time
                print(f"  Map: Level {mapNumber + 1} ")
                #  thời gian giải thuật
                print(f"  Thời gian: {elapsed_time} seconds")
            else:
                sceneState = "end"
                found = False
        if sceneState == "loading":
            loadingGame()
            sceneState = "executing"
```

4. Xử lý sự kiện người chơi

```
    if sceneState == "end":
        if found:
            foundGame(list_board[0][stateLength - 1])
        else:
            notfoundGame()
    if sceneState == "playing":
        clock.tick(2)
        renderMap(list_board[0][currentState])
        currentState = currentState + 1
        if currentState == stateLength:
            sceneState = "end"
            found = True
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT and sceneState == "init":
                if mapNumber < len(maps) - 1:
                    mapNumber = mapNumber + 1
            if event.key == pygame.K_LEFT and sceneState == "init":
                if mapNumber > 0:
                    mapNumber = mapNumber - 1
            if event.key == pygame.K_RETURN:
                if sceneState == "init":
                    sceneState = "loading"
                if sceneState == "end":
                    sceneState = "init"
            if event.key == pygame.K_SPACE and sceneState == "init":
                if algorithm == "Euclidean Distance Heuristic":
                    algorithm = "Manhattan Distance Heuristic"
                elif algorithm == "Manhattan Distance Heuristic":
                    algorithm = "BFS"
                else:
                    algorithm = "Euclidean Distance Heuristic"
    pygame.display.flip()
pygame.quit()
```

- Thư viện Colorama là một thư viện Python nhằm đơn giản hóa việc làm cho văn bản trên màn hình console trở nên màu sắc và nổi bật. Colorama cho phép bạn thêm màu sắc, in đậm, nghiêng, gạch chân và thay đổi màu nền của văn bản trên dòng lệnh hoặc console.
- Colorama được sử dụng để định nghĩa màu sắc cho các yếu tố trong trạng thái Sokoban (tường, hộp, người chơi, mục tiêu, ô trống) và sau đó in trạng thái với màu sắc lên màn hình console.
- Điều này có thể làm cho trò chơi trở nên thú vị hơn và dễ đọc hơn khi hiển thị thông tin trạng thái cho người chơi.

3.4. Áp dụng thuật toán

3.4.1. Thuật toán BFS

Cách thức thực hiện BFS:

Bước 1: Khởi tạo hàng đợi và đánh dấu đỉnh gốc là đã thăm.

Bước 2: Lặp cho đến khi hàng đợi trống:

- Lấy đỉnh hiện tại từ đầu hàng đợi.
- Kiểm tra xem đỉnh hiện tại có phải là đích cần tìm hay không. Nếu đúng, thuật toán kết thúc và trả về kết quả.
- Nếu không, duyệt qua tất cả các đỉnh kề của đỉnh hiện tại chưa được thăm.
- Đánh dấu các đỉnh kề và đưa chúng vào cuối hàng đợi.

Bước 3: Nếu hàng đợi trống và không tìm thấy đỉnh đích, thuật toán kết thúc và trả về kết quả không tìm thấy.

- Thuật toán BFS có thể được sử dụng để tìm kiếm đường đi ngắn nhất từ vị trí ban đầu của người chơi đến trạng thái kết thúc, tức là tìm kiếm đường đi tối ưu để đẩy các hộp vào các ô đích.
- Bằng cách duyệt qua các trạng thái được sinh ra từ BFS, ta có thể xác định đường đi tối ưu và số bước di chuyển ít nhất để hoàn thành mỗi cấp độ của trò chơi.

Kiểm tra tính khả thi:

- Trước khi thực hiện một bước di chuyển, thuật toán BFS có thể được áp dụng để kiểm tra tính khả thi của hướng di chuyển đó.

- Bằng cách tạo một trạng thái mới và kiểm tra xem trạng thái đó có hợp lệ hay không (ví dụ: không đi vào tường hoặc đẩy hộp ra khỏi bản đồ), ta có thể xác định các bước di chuyển hợp lệ mà người chơi có thể thực hiện.

Tạo cấp độ mới:

- BFS có thể được sử dụng để tạo ra các cấp độ mới trong trò chơi Sokoban.
- Bằng cách sử dụng BFS từ trạng thái ban đầu, ta có thể sinh ra các trạng thái mới bằng cách thực hiện các bước di chuyển từng bước và đánh dấu các trạng thái đã được thăm.
- Điều này cho phép chúng ta tạo ra các cấp độ mới có độ khó tăng dần cho người chơi.

3.4.2. Thuật toán A* (A-star):

A* là một thuật toán tìm kiếm đường đi trong đồ thị hoặc lưới dựa trên kết hợp giữa tìm kiếm theo chiều rộng (BFS) và tìm kiếm theo độ ưu tiên (best-first search). Thuật toán A* sử dụng một hàm heuristic để ước lượng chi phí từ điểm hiện tại đến điểm đích, kết hợp với chi phí thực tế đã đi qua để đưa ra quyết định tìm kiếm. A* duyệt qua các ô trong không gian tìm kiếm dựa trên giá trị heuristic và cost gần nhất, tìm kiếm đường đi ngắn nhất từ điểm bắt đầu đến điểm đích.

+ Hàm Euclide:

- Hàm Euclide là một hàm heuristic được sử dụng trong thuật toán A* để ước lượng chi phí từ điểm hiện tại đến điểm đích.
- Hàm Euclide tính khoảng cách Euclide (khoảng cách theo đường thẳng) giữa hai điểm trên mặt phẳng.
- Công thức tính khoảng cách Euclide giữa hai điểm (x1, y1) và (x2, y2) là

$$h = sqrt((x2 - x1)^2 + (y2 - y1)^2)$$

Trong trò chơi Sokoban, vị trí hiện tại của người chơi là điểm bắt đầu và vị trí của hộp cần di chuyển đến là điểm mục tiêu. Hàm Euclide ước lượng chi phí còn lại từ vị trí hiện tại đến vị trí mục tiêu dựa trên khoảng cách Euclide, giúp thuật toán A* tìm kiếm đường đi tối ưu.

+ Hàm Manhattan:

- Manhattan cũng là một hàm heuristic được sử dụng trong thuật toán A* để ước lượng chi phí từ điểm hiện tại đến điểm đích.

- Hàm Manhattan tính khoảng cách Manhattan (tổng giá trị tuyệt đối) giữa hai điểm trên mặt phẳng.
- Công thức tính khoảng cách Manhattan giữa hai điểm (x1, y1) và (x2, y2) là

$$h = |x2 - x1| + |y2 - y1|.$$

- Trong trò chơi Sokoban, vị trí hiện tại của người chơi là điểm bắt đầu và vị trí của hộp cần di chuyển đến là điểm mục tiêu.
- Hàm Manhattan ước lượng chi phí còn lại từ vị trí hiện tại đến vị trí mục tiêu dựa trên khoảng cách Manhattan, giúp thuật toán A* tìm kiếm đường đi tối ưu.

CHƯƠNG 4: KIỂM THỬ VÀ ĐÁNH GIÁ

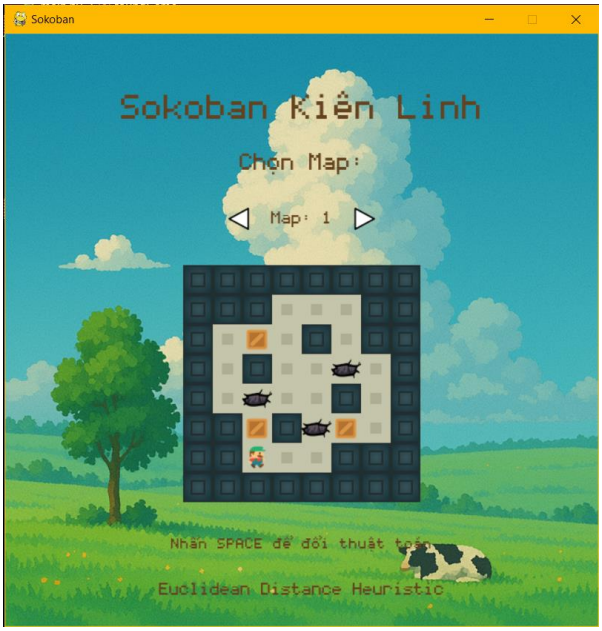
4.1 Các thư mục code

Thư mục Assets: Chứa các file tài nguyên chính của game như hình ảnh, font chữ,logo game,...
Thư mục Testcases: Chứa các file .txt với mỗi file là một map của game,trong từng file.txt chứa các ký hiệu được chỉ định là một đối tượng trong một map.
Thư mục Checkpoints: Chứa các file .txt với mỗi file là chứa tọa độ x,y của các điểm cần phải đẩy thùng tương ứng với số thứ tự mỗi file trong thư mục Testcases.
Thư mục Sources: chứa các file code python chính cho game.

- File astar : triển khai thuật toán tìm kiếm A* với hàm Manhattan Heuristic.
- File astar1 : triển khai thuật toán tìm kiếm A* với hàm Euclidean Heuristic.
- File bfs : Thư mục "bfs" triển khai thuật toán tìm kiếm theo chiều rộng (BFS) để giải quyết tìm đường đi trong trò chơi Sokoban.
- File main: Thư mục “main” chứa các logic chính của game ví dụ như cách render giao diện, xử lý tuần tự các trạng thái như lúc khởi tạo,bắt đầu,…..
- File support_function: Thư mục “support_function” chứa một tập hợp các hàm hỗ trợ cho việc giải bài toán trong trò chơi Sokoban bằng Python. Ở đây file này hỗ trợ các hàm phụ cho thuật toán BFS và thuật toán A* chứa Mahattan Heuristic.
- File support_function1: Thư mục “support_function1” chứa một tập hợp các hàm hỗ trợ cho việc giải bài toán trong trò chơi Sokoban bằng Python. Ở đây file này hỗ trợ các hàm phụ cho thuật toán A* chứa Mahattan Heuristic.

4.2. Cách chạy trò chơi

1. Để vào trò chơi chạy trực tiếp chương trình.
2. Khi vào ta sẽ thấy giao diện chính của trò chơi:



3. Để thay đổi thuật toán tìm kiếm ta ấn vào SPACE. (có 3 phương thức để ta lựa chọn)
4. Lựa chọn map tùy ý với 30 map. (ấn nút sang phải sang trái để thay đổi)
5. Sau cùng ấn ENTER để trò chơi tự tìm lời giải.

4.3 Kết quả

Các thuật toán được thực hiện bằng cách sử dụng hai đặc điểm quan trọng. Một là thời gian chờ và cái còn lại là độ sâu tối đa. Các thuật toán được thực hiện để dừng khi đạt đến độ sâu được xác định trước nhất định được gọi là độ sâu tối đa và cũng bị chấm dứt khi vượt quá thời gian tối đa, hết thời gian chờ. Điều này là cần thiết để ngăn chặn các thuật toán tìm kiếm trong không gian tìm kiếm vô hạn.

Chúng em đã xem xét hai phương pháp phỏng đoán cho Thuật toán A*. Phương pháp phỏng đoán đầu tiên được triển khai bằng cách xem xét khoảng cách tối thiểu được tính bằng khoảng cách Euclide giữa mục tiêu và các hộp. Heuristic thứ hai được tính toán bằng cách tính khoảng cách tối thiểu với khoảng cách Manhattan giữa chúng.

- Cả hai hàm heuristic Euclidean và Manhattan đều có thể cung cấp một ước lượng về khoảng cách từ trạng thái hiện tại đến trạng thái đích. Tuy nhiên, hiệu suất tìm kiếm có thể khác nhau vì cách mà hai hàm đánh giá khoảng cách.
- Cả hai hàm Euclidean và Manhattan đều có thể cung cấp lời giải chính xác trong trò chơi Sokoban. Tuy nhiên, độ chính xác của lời giải có thể khác nhau và phụ thuộc vào cấu trúc của bản đồ và các trạng thái trong trò chơi.
- Sử dụng hàm Euclidean có thể yêu cầu tính toán khoảng cách theo đường thẳng giữa các trạng thái, trong khi hàm Manhattan đánh giá khoảng cách theo đường chéo hoặc theo số bước ngang, dọc. Việc tính toán theo đường thẳng có thể tốn nhiều thời gian và tài nguyên tính toán hơn so với tính toán theo đường chéo hoặc số bước. Do đó, sử dụng hàm Manhattan có thể có hiệu quả thời gian và tài nguyên tốt hơn so với hàm Euclidean.

*** Đối với BFS:**

- Độ phức tạp thời gian và không gian: BFS có độ phức tạp thời gian và không gian lớn. Trong trò chơi Sokoban với bản đồ lớn và nhiều hộp, số lượng trạng thái có thể phát sinh rất lớn, dẫn đến việc tạo ra một hàng đợi rất lớn và yêu cầu nhiều bộ nhớ. Điều này có thể làm cho việc tìm kiếm trở nên chậm và không khả thi trong các trường hợp phức tạp.

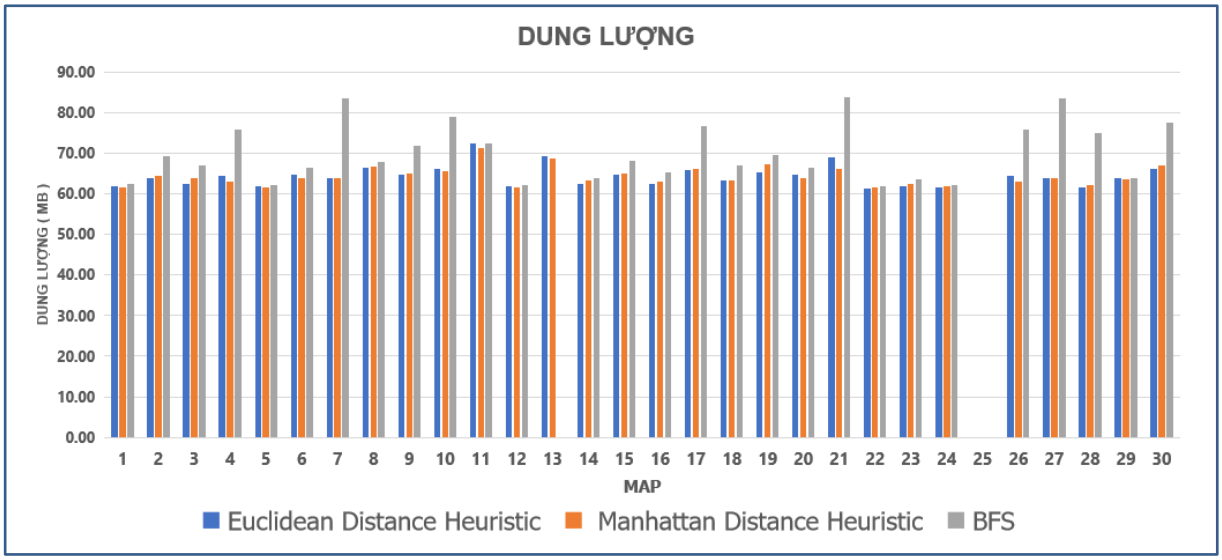
- Không tối ưu hóa: BFS duyệt qua tất cả các trạng thái có thể từ trạng thái ban đầu, ngay cả khi chúng không đưa đến trạng thái đích. Điều này có nghĩa là BFS không có khả năng ưu tiên tìm kiếm các trạng thái tiềm năng tốt nhất hoặc tìm kiếm theo hướng nhanh nhất đến trạng thái đích. Trong trò chơi Sokoban, có thể tồn tại những cấu trúc bản đồ phức tạp và các trường hợp mà BFS không tối ưu.

- Không xử lý được tình huống quay lui: BFS không xử lý tốt các tình huống quay lui trong trò chơi Sokoban. Trong một số trường hợp, để đẩy một hộp vào ô đích, người chơi có thể cần đẩy hộp qua một vị trí cần phải quay lại sau đó. BFS không tự động xử lý được tình huống này mà cần phải sử dụng các phương pháp khác như backtracking để giải quyết.

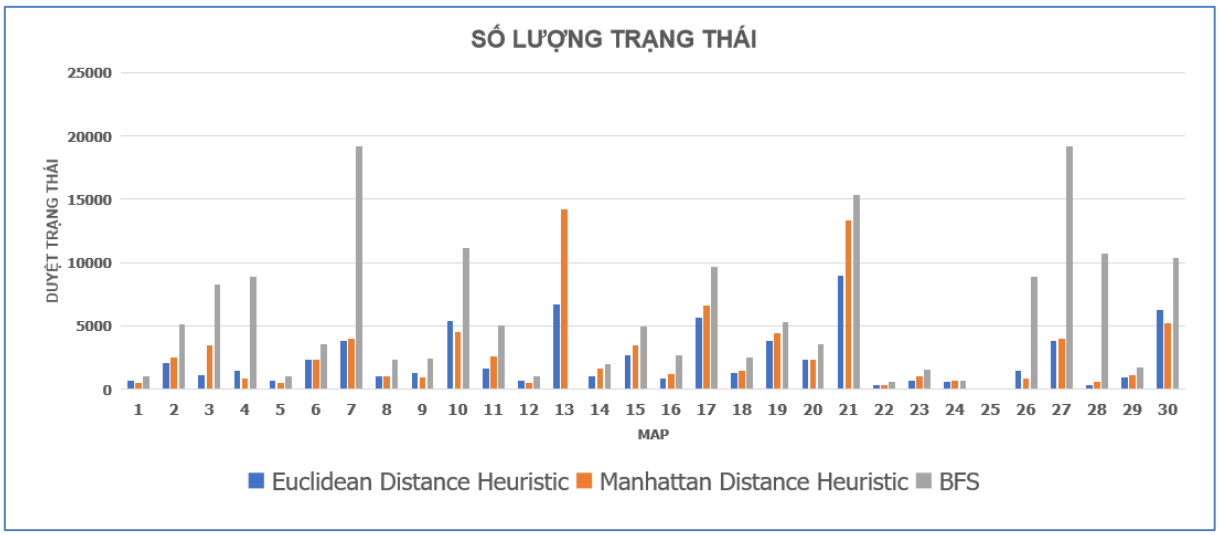
- Không sử dụng thông tin heuristic: BFS không sử dụng thông tin heuristic (đánh giá ước lượng) để ưu tiên tìm kiếm các trạng thái tiềm năng tốt nhất hoặc tìm kiếm theo hướng nhanh nhất đến trạng thái đích. Trong trò chơi Sokoban, có thể sử dụng các thông tin heuristic như khoảng cách Manhattan giữa các hộp

và ô đích để tăng hiệu suất tìm kiếm. Tuy nhiên, BFS không có khả năng tự động áp dụng các heuristic này.

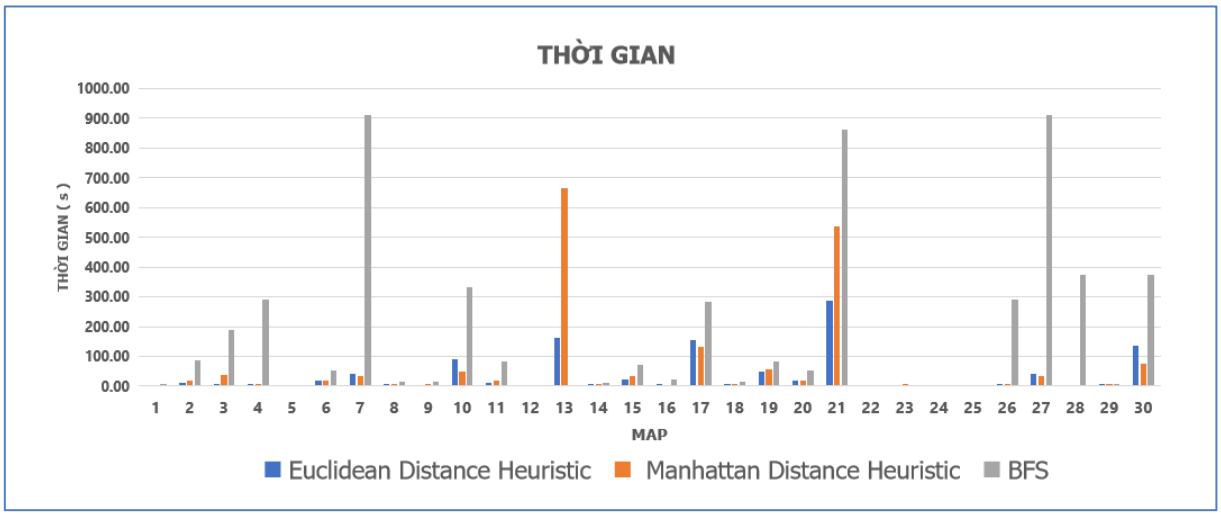
* Biểu đồ so sánh dung lượng khi chạy trên 30 map



* Biểu đồ về số lượng trạng thái



* Biểu đồ về thời gian



Nhận xét: BFS hiệu quả hơn so với A* trong vấn đề này. Cụ thể hơn:

Về thời gian chạy, thuật toán A* chậm hơn so với BFS. Lý do là thuật toán bên trong khi chúng ta thực hiện. Đầu tiên, A* cần tính toán chi phí bất cứ khi nào nó tạo ra một nút mới. Giai đoạn này sử dụng để khớp khoảng cách giữa các hộp và mục tiêu và tìm tổng tối thiểu của chi phí này và nó lãng phí trong khi BFS không cần lưu trữ giá trị này. Hơn nữa, cấu trúc dữ liệu mà chúng tôi sử dụng để lưu trữ danh sách các nút được sắp xếp theo thứ tự chi phí của nó là . Mọi thao tác như chèn xóa sẽ khiến chúng ta phải trả giá trong khi nút trong BFS sẽ được lưu trữ trong một danh sách cơ bản và các thao tác này chỉ khiến chúng ta phải trả giá.

Về không gian, nút trung bình được tạo ra trong thuật toán A* chỉ được so sánh với thuật toán BFS. Vì BFS sẽ quét các nút trong toàn bộ cấp độ hiện tại cho đến khi tìm thấy trạng thái cuối cùng hoặc chuyển sang cấp độ tiếp theo và lặp lại trong khi A* sẽ chọn nút có chi phí tối thiểu và đi về phía nút đó và mở rộng nhiều nút hơn ở cấp độ con và tiếp tục so sánh chi phí và chọn nút tiếp theo cho đến khi tìm thấy trạng thái cuối cùng, vì vậy nó không bị hạn chế, chúng ta phải quét tất cả các nút trong một cấp độ.

Tài liệu tham khảo

1. Giáo trình Artificial Intelligence – Stuart Russell, Peter Norvig 1994

Slide Trí tuệ nhân tạo –GV. Ngô Văn Linh.

2. http://pavel.klavik.cz/projekty/solver/solver.pdf

3. Taylor, Joshua, and Ian Parberry. "Procedural generation of Sokoban levels." Proceedings of the International North American Conference on Intelligent Games and Simulation. 2011. (https://ianparberry.com/pubs/GAMEON-NA_METH_03.pdf)