TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

MÔN: TRÍ TUỆ NHÂN TẠO

Đề tài:

THIẾT KẾ VÀ CÀI ĐẶT CHƯƠNG TRÌNH TRÍ TUỆ NHÂN TẠO VỀ TRÒ CHƠI SOKOBAN

GVHD: *PGS. TS Hoàng Văn Dũng*

Mã HP: *ARIN330585_23_1_04CLC*

SVTH: *Nhóm 2*

MSSV	Họ và tên	Mức độ (%)
21110299	Nguyễn Phú Thành	100
21110175	Nguyễn Văn Hào	100
21110202	Bùi Quốc Khang	100
21110272	Phạm Hùng Phong	100

Tp. Hồ Chí Minh, tháng 12 năm 2023

DANH SÁCH THÀNH VIÊN THAM GIA ĐỒ ÁN

Mã học phần:

Đề tài: Thiết kế và cài đặt chương trình Trí tuệ nhân tạo về trò chơi Sokoban

---

STT	HỌ VÀ TÊN THÀNH VIÊN	MÃ SỐ SINH VIÊN	TỶ LỆ THAM GIA
1	Nguyễn Phú Thành	21110299	100%
2	Nguyễn Văn Hào	21110175	100%
3	Bùi Quốc Khang	21110202	100%
2	Phạm Hùng Phong	21110273	100%

Ghi chú:

Tỷ lệ %: Mức độ phần trăm hoàn thành của từng sinh viên tham gia.

Trưởng nhóm: Bùi Quốc Khang

Nhận xét của giáo viên

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh - Tháng 12 năm 2023

LỜI CẢM ƠN

Lời đầu tiên, nhóm em xin gửi lời cảm ơn chân thành nhất đến ***Thầy Hoàng Văn Dũng*** – giảng viên bộ môn ***Trí tuệ nhân tạo*** của chúng em. Trong quá trình học tập và tìm hiểu bộ môn, chúng em đã nhận được sự quan tâm giúp đỡ, hướng dẫn rất tận tình và tâm huyết từ Thầy. Thầy đã giúp chúng em tích lũy thêm nhiều kiến thức để có cái nhìn sâu sắc và hoàn thiện hơn trong lĩnh vực công nghệ thông tin. Để từ đó, ứng dụng những kiến thức mà Thầy truyền tải, nhóm em xin trình bày lại những gì mà mình đã học hỏi được thông qua việc thực hiện đề tài ***"Thiết kế và cài đặt chương trình Trí tuệ nhân tạo về trò chơi Sokoban"***.

Kiến thức là vô hạn và sự tiếp nhận kiến thức của bản thân mỗi người luôn tồn tại những hạn chế nhất định. Do đó, trong phạm vi khả năng của bản thân, nhóm em đã rất cố gắng để hoàn thành đề tài một cách tốt nhất. Tuy nhiên, chắc chắn không tránh khỏi những thiếu sót, nhóm chúng em rất mong nhận được sự cảm thông và những ý kiến đóng góp đến từ Thầy để đề tài của nhóm em được hoàn thiện hơn.

Một lần nữa, ***nhóm em xin chân thành cảm ơn Thầy*** đã tận tình hướng dẫn, chỉ bảo các thành viên nhóm em trong suốt quá trình học tập và thực hiện đồ án này.

Kính chúc Thầy sức khỏe, hạnh phúc thành công trên con đường sự nghiệp giảng dạy.

Trân trọng

Đại diện nhóm

Bùi Quốc Khang

MỤC LỤC

MỤC LỤC HÌNH ẢNH

MỤC LỤC BẢNG

LỜI MỞ ĐẦU 1

CHƯƠNG 1. MỞ ĐẦU 2

1.1 Phát biểu về bài toán: 2

1.2 Mục đích, yêu cầu thực hiện: 3

1.3 Đối tượng, phạm vi nghiên cứu: 3

1.3.1 Đối tượng nghiên cứu: 3

1.3.2 Phạm vi nghiên cứu: 4

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 5

2.1 Ngôn ngữ lập trình 5

2.1.1 Ngôn ngữ Python 5

2.1.2 Đặc điểm của ngôn ngữ 5

2.2 Môi trường lập trình 6

2.3 Xây dựng giao diện với Pygame 7

2.4 Các thuật toán áp dụng 8

2.4.1 Thuật toán Breadth First Search (BFS): 8

2.4.2 Thuật toán Depth First Search (DFS): 9

2.4.3 Thuật toán Interative Deepening Depth First Search (IDFS): 10

2.4.4 Thuật toán Limited Depth First Search (LDFS): 11

2.4.5 Thuật toán Greedy 12
2.4.6 Thuật toán UCS 13
2.4.7 Thuật toán A Star 15
2.4.8 Thuật toán Hill Climbing 16
2.4.9 Thuật toán Beam Search 17
CHƯƠNG 3. PHÂN TÍCH, THIẾT KẾ GIẢI PHÁP 19
3.1 Mô tả dữ liệu và thiết lập tham số 19
3.2 Các phương thức chính 21
CHƯƠNG 4. CÀI ĐẶT, ĐÁNH GIÁ KẾT QUẢ 26
4.1 Cài đặt chương trình 26
4.2 Phân tích kết quả 32
4.2.1 Phân tích map số 1 32
4.2.2 Phân tích map số 2 35
4.2.3 Phân tích tổng hợp 20 map 38
CHƯƠNG 5. KẾT LUẬN 41
5.1 Đánh giá kết quả thực hiện được 41
5.2 Định hướng phát triển 41
BẢNG PHÂN CÔNG CÔNG VIỆC 43
TÀI LIỆU THAM KHẢO 44

MỤC LỤC HÌNH ẢNH

Hình 1: Minh họa game Sokoban 2
Hình 2 Trang web chính thức của Python 5
Hình 3. Giao diện Spyder 7
Hình 4. Minh họa Pygame 8
Hình 5. Form đăng nhập 26
Hình 6. Giao diện sau khi đăng nhập 27
Hình 7. Giao diện khi nhấn Play 27
Hình 8. Giao diện chơi 1 player 28
Hình 9. Giao diện khi chọn một mức độ 28
Hình 10. Giao diện chơi 1 level 29
Hình 11. Giao diện sau khi chiến thắng map 30
Hình 12. Giao diện chơi 2 players 31
Hình 13. Giao diện chơi Map 1 32
Hình 14. Áp dụng các thuật toán vào Map 1 33
Hình 15. Biểu đồ so sánh số đỉnh duyệt và thời gian tìm kiếm 9 thuật toán vào Map 1 34
Hình 16. Biểu đồ so sánh số bước đi 9 thuật toán trong Map 1 34
Hình 17. Giao diện chơi game Map 2 35
Hình 18. Áp dụng thuật toán LDFS vào Map 2 36
Hình 19. Biểu đồ so sánh số đỉnh duyệt và thời gian tìm kiếm 9 thuật toán trong Map 2 37
Hình 20. Biểu đồ sánh số bước đi của 9 thuật toán trong Map 2 37
Hình 21. Số liệu thống kê các thông số của các thuật toán trên 20 map 38
Hình 22. Biểu đồ so sánh số đỉnh đã duyệt của 9 thuật toán trên 20 map 39
Hình 23. Biểu đồ so sánh thời gian tìm kiếm của 9 thuật toán trên 20 map 39
Hình 24. Biểu đồ so sánh số bước di chuyển của 9 thuật toán trên 20 map 40

MỤC LỤC BẢNG

Bảng 1. So sánh số đỉnh, thời gian và số bước đi của các thuật toán trong Map 1 33
Bảng 2. So sánh số đỉnh, thời gian và số bước đi của các thuật toán trong Map 2 36
Bảng 3. Bảng phân công nhiệm vụ 43

LỜI MỞ ĐẦU

Trong thế giới game hiện đại, sự phát triển của công nghệ trí tuệ nhân tạo (AI) đang mở ra cánh cửa cho những trải nghiệm chơi game mới mẻ và hấp dẫn hơn bao giờ hết. Và một trong những ứng dụng tiêu biểu của công nghệ này là trong trò chơi Sokoban - một trò chơi logic thú vị, đòi hỏi người chơi phải sắp xếp và đẩy các hộp đến vị trí đích. Bằng cách sử dụng các thuật toán và kỹ thuật học máy, trí tuệ nhân tạo có thể tạo ra các chiến lược chơi game thông minh, thách thức người chơi và cung cấp trải nghiệm chơi game độc đáo.

Mặc dù trò chơi Sokoban có vẻ đơn giản, nhưng việc tạo ra một hệ thống trí tuệ nhân tạo hiệu quả để chơi trò chơi này vẫn đòi hỏi sự kết hợp hoàn hảo giữa khả năng lập kế hoạch, quản lý tài nguyên và kỹ năng đưa ra quyết định nhanh chóng. Đối với các nhà phát triển game, sự tích hợp của công nghệ trí tuệ nhân tạo trong Sokoban không chỉ là một thử thách mà còn là cơ hội để cung cấp trải nghiệm chơi game độc đáo và thú vị cho người chơi.

Thông qua việc nghiên cứu chi tiết về ứng dụng của trí tuệ nhân tạo trong trò chơi Sokoban, đề tài nghiên cứu này được thực hiện với hy vọng có thể hiểu rõ hơn về cách mà công nghệ này đóng góp vào sự phát triển của ngành công nghiệp game và cách nó ảnh hưởng đến trải nghiệm chơi game của người chơi.

CHƯƠNG 1. MỞ ĐẦU

1.1 Phát biểu về bài toán:

Game Sokoban là một trò chơi logic có tính năng giải đố và là một trong những trò chơi cổ điển và phổ biến trong thể loại này. Đề tài về game Sokoban thường bao gồm các khía cạnh sau:

- Luật chơi cơ bản: Trong Sokoban, người chơi điều khiển một nhân vật (thường là một người công nhân hoặc một nhân vật tương tự) để đẩy các hộp (hoặc các đối tượng khác) vào các ô mục tiêu trên một màn chơi, người chơi chỉ có thể đẩy hộp một cách trái phép, không thể kéo hoặc đẩy nhiều hộp cùng một lúc, mục tiêu của trò chơi là di chuyển tất cả các hộp đến vị trí đích trong số lần bước ít nhất có thể.

- Độ khó: Sokoban được thiết kế với nhiều cấp độ khó khác nhau, từ dễ đến rất khó

- Thể loại: Sokoban thuộc thể loại game puzzle và logic

- Ứng dụng: Sokoban không chỉ là một trò chơi giải trí, mà còn được sử dụng trong lĩnh vực nghiên cứu trí tuệ nhân tạo và tối ưu hóa

Hình 1: *Minh họa game Sokoban*

Trong quá trình giải bài toán, đề tài về game Sokoban tập trung vào việc nghiên cứu và phát triển các khía cạnh của trò chơi này, bao gồm cách thiết kế màn chơi, độ

khó, ứng dụng trong lĩnh vực giáo dục và nghiên cứu. Sokoban là một ví dụ điển hình về trò chơi logic mang tính thách thức cao và vẫn thu hút sự quan tâm của người chơi trên khắp thế giới.

1.2 Mục đích, yêu cầu thực hiện:

Đề tài ***Thiết kế và cài đặt chương trình Trí tuệ nhân tạo về trò chơi Sokoban*** đặt ra các vấn đề trọng tâm về mục tiêu mà sinh viên nghiên cứu cần thực hiện được bao gồm:

Vấn đề đầu tiên được đưa ra về yếu tố lý thuyết, sinh viên thực hiện cần nghiên cứu chuyên môn, hiểu được tổng quan và khả năng ứng dụng của thuật toán mà nhóm đã lựa chọn BFS, DFS, IDFS, LDFS, Greedy, UCS, A*, Beam Search và Hill Climbing. Song song với công nghệ được lựa chọn để sử dụng, nhóm sinh viên cần tìm hiểu thêm cách xây dựng giao diện đẹp, thân thiện với người dùng.

Vấn đề thứ hai mà đề tài đặt ra là ứng dụng các kiến thức đã tìm hiểu được vào xây dựng một sản phẩm cụ thể, ở đây là **Chương trình tìm kiếm lời giải**, sản phẩm hoàn thành phải là một chương trình, đáp ứng các yêu cầu cơ bản, phục vụ cho người dùng trong việc tìm lời giải cho bài toán và có thể so sánh thời gian giải của các thuật toán khác nhau như thế nào.

1.3 Đối tượng, phạm vi nghiên cứu:

1.3.1 Đối tượng nghiên cứu:

Các thuật toán:

+ Breadth First Search (BFS)
+ Depth First Search (DFS)
+ IDFS
+ LDFS
+ Greedy
+ UCS
+ A*

+ Hill climbing

+ BeamSearch

1.3.2 Phạm vi nghiên cứu:

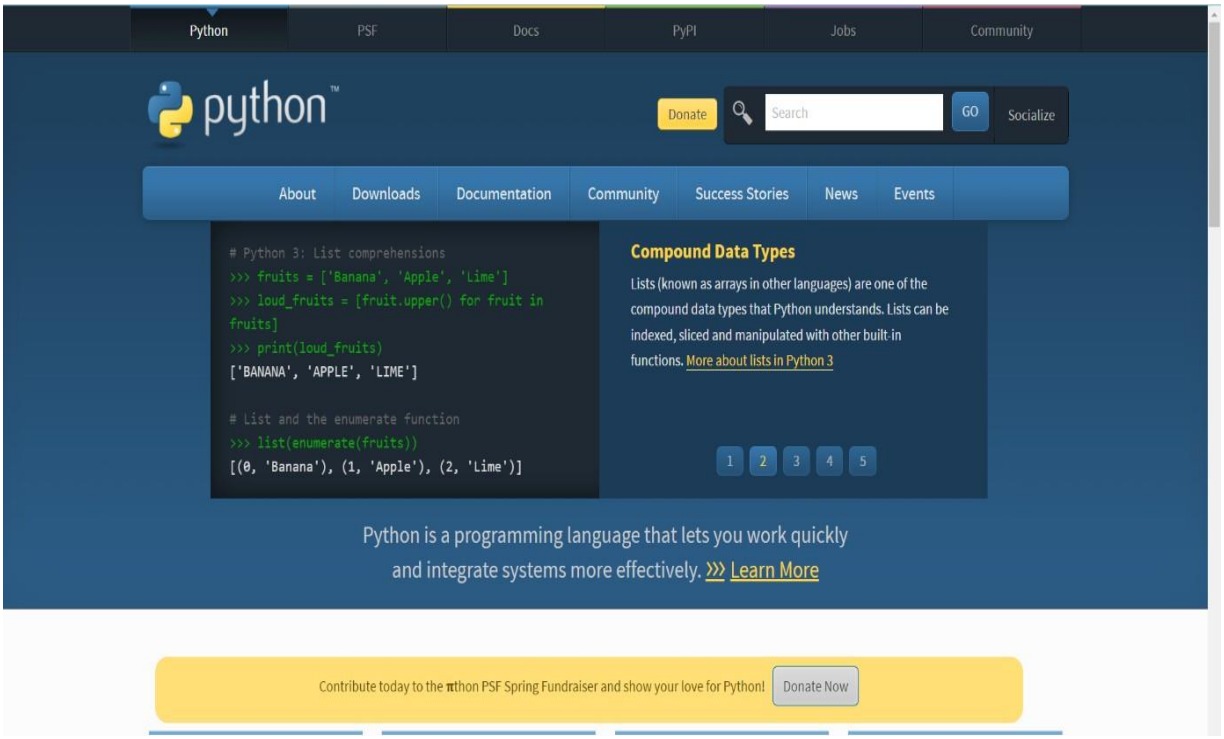
Cách tìm kiếm lời giải cho các thuật toán BFS, DFS, IDFS, Greedy, UCS, LDFS, A*, Beam Search và Hill Climbing cho bài toán Sokoban

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Ngôn ngữ lập trình

2.1.1 Ngôn ngữ Python

Python là một ngôn ngữ lập trình được sử dụng rộng rãi trong các ứng dụng web, phát triển phần mềm, khoa học dữ liệu và máy học (ML). Các nhà phát triển sử dụng Python vì nó hiệu quả, dễ học và có thể chạy trên nhiều nền tảng khác nhau. Phần mềm Python được tải xuống miễn phí, tích hợp tốt với tất cả các loại hệ thống và tăng tốc độ phát triển



Hình 2 *Trang web chính thức của Python*

2.1.2 Đặc điểm của ngôn ngữ

- Python được thông dịch: Python được trình thông dịch xử lý trong thời gian chạy. Bạn không cần phải biên dịch chương trình của mình trước khi thực hiện nó.

- Python có tính tương tác (Interactive): Tại một dấu nhắc Python (command

line) bạn có thể tương tác trực tiếp với trình thông dịch để viết chương trình Python.

- Python hỗ trợ kỹ thuật lập trình hướng đối tượng hoặc kỹ thuật lập trình đóng gói mã trong các đối tượng.

2.2 Môi trường lập trình

Spyder là một môi trường khoa học mã nguồn mở miễn phí được viết bằng Python, dành cho Python, được thiết kế bởi và dành cho các nhà khoa học, kỹ sư và nhà phân tích dữ liệu. Nó có sự kết hợp độc đáo giữa chức năng chỉnh sửa, phân tích, gỡ lỗi và lập hồ sơ nâng cao của một công cụ phát triển toàn diện với khả năng khám phá dữ liệu, thực thi tương tác, kiểm tra sâu và khả năng trực quan hóa của một gói khoa học..

**** Các ưu điểm của Spyder:***

- Môi trường tích hợp: Spyder kết hợp một loạt chức năng quan trọng như chỉnh sửa mã, phân tích số liệu, gỡ lỗi và trực quan hóa vào một giao diện duy nhất. Điều này giúp người dùng tiết kiệm thời gian và năng lực trong quá trình làm việc.

- Dành riêng cho Python: Spyder tập trung vào ngôn ngữ lập trình Python, tạo điều kiện thuận lợi cho việc phát triển và thử nghiệm mã nguồn Python một cách hiệu quả.

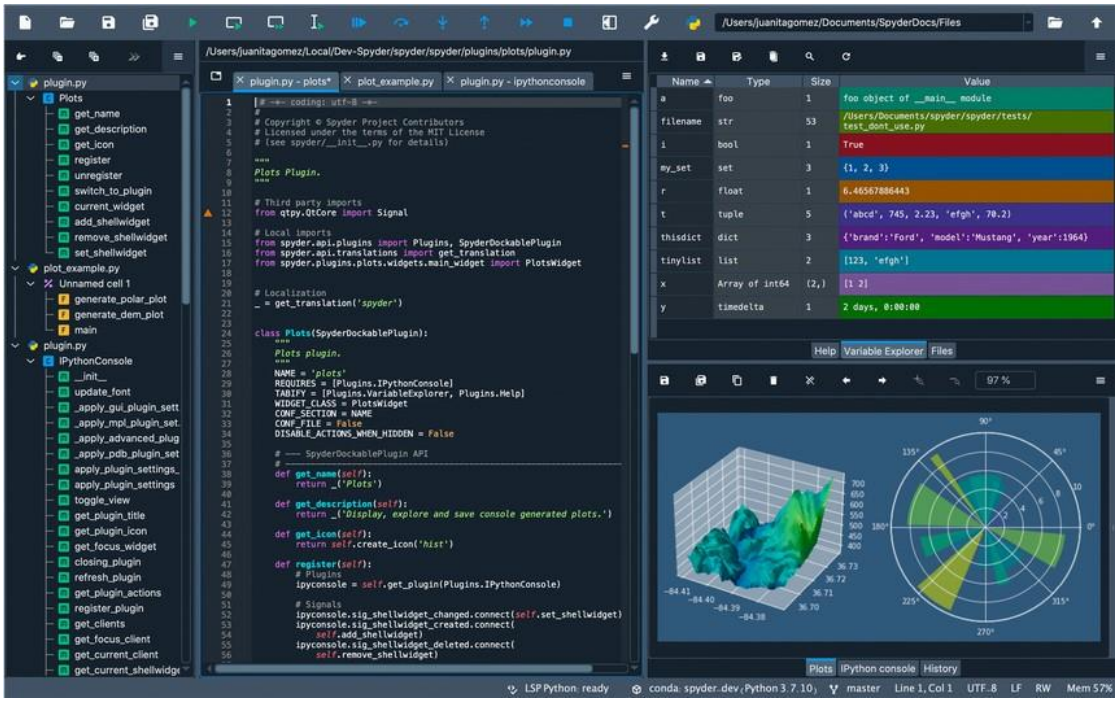
- Khám phá dữ liệu tiện lợi: Spyder cung cấp các công cụ mạnh mẽ cho việc khám phá dữ liệu, giúp người dùng nhanh chóng hiểu rõ thông tin và xu hướng bên trong dữ liệu.

- Tương tác và thực thi dễ dàng: Môi trường tương tác của Spyder cho phép người dùng thực thi từng phần mã và kiểm tra kết quả ngay lập tức, giúp tối ưu quá trình phát triển và chỉnh sửa mã.

- Gỡ lỗi thông minh: Spyder cung cấp các công cụ gỡ lỗi tiện lợi giúp người dùng xác định và sửa lỗi một cách nhanh chóng, giúp tăng hiệu suất làm việc.

- Trực quan hóa đẹp mắt: Với khả năng trực quan hóa dữ liệu mạnh mẽ, Spyder giúp người dùng tạo ra các biểu đồ và đồ thị hấp dẫn để trình bày kết quả nghiên cứu một cách rõ ràng và dễ hiểu.

- Phù hợp cho mọi người: Dù bạn là nhà khoa học, kỹ sư hay nhà phân tích dữ liệu, Spyder đều cung cấp môi trường làm việc linh hoạt và hiệu quả cho tất cả các dự án và nhiệm vụ.



***Hình 3**. Giao diện Spyder*

2.3 Xây dựng giao diện với Pygame

Pygame là một bộ mô-đun Python đa nền tảng được thiết kế để viết trò chơi điện tử. Nó bao gồm đồ họa máy tính và thư viện âm thanh được thiết kế để sử dụng với ngôn ngữ lập trình Python

Pygame sử dụng thư viện Simple DirectMedia Layer (SDL), với mục đích cho phép phát triển trò chơi máy tính trong thời gian thực mà không cần cơ chế bậc thấp của ngôn ngữ lập trình C và các dẫn xuất của nó.

Các tính năng khác mà SDL không có bao gồm toán học vectơ, phát hiện va chạm, quản lý đồ họa 2D, hỗ trợ MIDI, camera, thao tác mảng pixel, chuyển đổi, lọc, hỗ trợ phông chữ freetype nâng cao và vẽ.

***Hình 4**. Minh họa Pygame*

Trong đề tài này, nhóm chúng em sử dụng Pygame để xây dựng giao diện và mô phỏng chuyển động thuật toán.

Đầu mỗi file, khai báo thư viện pygame qua câu lệnh **'import pygame'**.

2.4 Các thuật toán áp dụng

2.4.1 Thuật toán Breadth First Search (BFS):

Breadth First Search (BFS) là một trong những thuật toán tìm kiếm phổ biến trong đồ thị. Thuật toán tìm đường đi từ đỉnh xuất phát đến đỉnh kết thúc bằng cách duyệt theo chiều rộng.

Tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: cho trước một đỉnh của đồ thị; thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn

nhất có thể.

Thuật toán BFS thực hiện duyệt các đỉnh từ đỉnh xuất phát và duyệt tiếp tới các đỉnh kề nó, tiếp tục cho đến khi không còn đỉnh nào để đi. Bên cạnh đó trong quá trình duyệt, ta lưu lại các đỉnh đã đi qua để từ đó truy vết tìm ra đường đi ngắn nhất.

Ứng dụng: Thuật toán tìm kiếm theo chiều rộng được dùng để giải nhiều bài toán trong lý thuyết đồ thị, chẳng hạn như:

+ Tìm tất cả các đỉnh trong một thành phần liên thông

+ Tìm đường đi ngắn nhất giữa hai đỉnh u và v (với chiều dài đường đi tính bằng số cung)

+ Kiểm tra xem một đồ thị có là đồ thị hai phía Tìm các thành phần liên thông

- *Ưu điểm:*

✓ Xét duyệt tất cả các đỉnh để trả về kết quả.

✓ Nếu số đỉnh là hữu hạn, thuật toán chắc chắn tìm ra kết quả.

- *Nhược điểm:*

✓ Mang tính chất vét cạn, không nên áp dụng nếu duyệt số đỉnh quá lớn.

✓ Mang tính chất mù quáng, duyệt tất cả đỉnh, không chú ý đến thông tin trong các đỉnh để duyệt hiệu quả, dẫn đến duyệt qua các đỉnh không cần thiết.

2.4.2 Thuật toán Depth First Search (DFS):

Giải thuật tìm kiếm theo chiều sâu (Depth First Search – viết tắt là DFS), còn được gọi là giải thuật tìm kiếm ưu tiên chiều sâu, là giải thuật duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị và sử dụng stack (ngăn xếp) để ghi nhớ đỉnh liền kề để bắt đầu việc tìm kiếm khi không gặp được đỉnh liền kề trong bất kỳ vòng lặp nào.

Thuật toán bắt đầu từ nút gốc (trên cùng) của cây và đi xa nhất có thể xuống một nhánh (đường dẫn) nhất định, sau đó quay lại cho đến khi tìm thấy đường dẫn chưa được khám phá và sau đó khám phá nó. Thuật toán thực hiện điều này cho đến khi toàn bộ đồ thị được khám phá. Nhiều vấn đề trong khoa học máy tính có thể được giải quyết dưới dạng đồ thị.

Ứng dụng: Tìm kiếm theo chiều sâu thường được sử dụng làm chương trình con trong các thuật toán phức tạp hơn.

+ Thuật toán so khớp, Hopcroft–Karp, sử dụng DFS như một phần của thuật toán để giúp tìm kết quả khớp trong biểu đồ

+ Tìm đường đi ngắn nhất giữa hai đỉnh u và v (với chiều dài đường đi tính bằng số cung)

+ Thuật toán duyệt cây, còn được gọi là tìm kiếm cây, có ứng dụng trong bài toán người bán hàng du lịch và thuật toán Ford-Fulkerso

Ưu điểm:

- ✓ Xét duyệt tất cả các đỉnh để trả về kết quả.
- ✓ Nếu số đỉnh là hữu hạn, thuật toán chắc chắn tìm ra kết quả.

- *Nhược điểm:*

- ✓ Mang tính chất vét cạn, không nên áp dụng nếu duyệt số đỉnh quá lớn.
- ✓ Mang tính chất mù quáng, duyệt tất cả đỉnh, không chú ý đến thông tin trong các đỉnh để duyệt hiệu quả, dẫn đến duyệt qua các đỉnh không cần thiết.

2.4.3 Thuật toán Interative Deepening Depth First Search (IDFS):

Trong trí tuệ nhân tạo hay các lý thuyết đồ thị, thuật toán tìm kiếm có giới hạn độ sâu (IDFS) hay là một thuật toán phát triển các nút chưa xét các theo chiều sâu nhưng có giới hạn mức để tránh đi vào những con đường không mang lại kết quả tốt như trong thuật toán tìm kiếm sâu dần.

IDFS kết hợp tính hiệu quả về không gian của tìm kiếm theo chiều sâu và tìm kiếm nhanh của tìm kiếm theo chiều rộng (đối với các nút gần gốc hơn). Thuật toán được đưa ra để khắc phục điểm yêu của thuật toán tìm kiếm giới hạn độ sâu LDFS . Đó là khi mà tất cả các lời giải nằm ở độ sâu lớn hơn giới hạn độ sâu l thì giải thuật LDFS sẽ thất bại.

Giải thuật tìm kiếm sâu dần sẽ :

- Áp dụng giải thuật DLS đối với đường đi có độ dài <= 1

- Nếu thất bại, tiếp tục áp dụng giải thuật DFS đối với đường đi có độ dài

<= 2 … cứ như vậy đến khi tìm được lời giải hoặc khi toàn bộ cây đã được xét mà không tìm được lời giải.

Ứng dụng: Tìm kiếm theo chiều sâu có giới hạn thường được áp dụng trong các bài toán đa dạng như:

+ Kiểm tra tính phẳng của đồ thị: IDFS có thể được sử dụng để kiểm tra tính phẳng của đồ thị.

+ Tìm LCA (Lowest Common Ancestor): IDFS có thể được sử dụng để tìm LCA của hai đỉnh trong cây.

+ Quy hoạch động trên cây: IDFS có thể được sử dụng để giải các bài toán quy hoạch động trên cây.

- *Ưu điểm:*

✓ Nó tổ chức các lợi ích của thuật toán tìm kiếm BFS và DFS về mặt hiệu quả tìm kiếm và bộ nhớ nhanh.

- *Nhược điểm:*

✓ Hạn chế chính của IDS là nó lặp lại tất cả các công việc của giai đoạn trước

2.4.4 Thuật toán Limited Depth First Search (LDFS):

Thuật toán Limited Depth-First Search (LDFS) hoạt động tương tự như Depth-First Search (DFS), nhưng với một hạn chế độ sâu cụ thể. Nó là một biến thể của DFS trong đó việc duyệt trạng thái sẽ dừng lại khi đạt đến mức độ sâu được xác định trước (giới hạn sâu).

LDFS cũng như IDFS, kết hợp tính hiệu quả về không gian của tìm kiếm theo chiều sâu và tìm kiếm nhanh của tìm kiếm theo chiều rộng (đối với các nút gần gốc hơn). Tuy nhiên, Lưu ý rằng Limited Depth-First Search sẽ không duyệt các trạng thái có độ

sâu lớn hơn giới hạn xác định trước. Điều này có thể dẫn đến kết quả không tìm thấy nếu trạng thái mục tiêu không nằm trong phạm vi của độ sâu được cho.

Ứng dụng: LDFS thường được sử dụng trong việc tìm kiếm trong không gian trạng thái khi người dùng có thông tin về độ sâu tối đa mà họ muốn tìm kiếm.

+ Kiểm tra tính phẳng của đồ thị: LDFS có thể được sử dụng để kiểm tra tính phẳng của đồ thị.

+ Tìm LCA (Lowest Common Ancestor): LDFS có thể được sử dụng để tìm LCA của hai đỉnh trong cây.

+ Quy hoạch động trên cây: LDFS có thể được sử dụng để giải các bài toán quy hoạch động trên cây.

- *Ưu điểm:*

✓ Nó là bộ nhớ hiệu quả, sử dụng không gian tuyến tính O (bxL)

- *Nhược điểm:*

✓ Chưa hoàn thành nếu giải pháp nằm dưới giới hạn L(d <l), vì nó không thể tìm thấy Solution.

✓ Nó có thể không tìm thấy tối ưu nếu có nhiều hơn Solution.

✓ Nó không hiệu quả về thời gian vì phải mất O (b ^ L).

✓ Nó có thể gây ra các vòng lặp nếu tìm kiếm cây được sử dụng trên biểu đồ.

2.4.5 Thuật toán Greedy

Thuật toán Greedy (tham lam) là một thuật toán tìm kiếm trong trí tuệ nhân tạo và tối ưu hóa, thường được sử dụng trong các bài toán quyết định đa giai đoạn. Trong thuật toán này, quyết định được đưa ra dựa trên quy tắc chọn lựa tốt nhất ở từng bước đến khi đạt được kết quả cuối cùng.

Lựa chọn của giải thuật tham lam có thể phụ thuộc vào lựa chọn trước đó. Việc quyết định sớm và thay đổi hướng đi của giải thuật cùng với việc không bao giờ xét lại các quyết định cũ sẽ dẫn đến kết quả là giải thuật này không tối ưu để tìm giải pháp toàn cục.

Ứng dụng: Tìm kiếm tham lam thường được áp dụng trong các bài toán đa dạng như:

+ Lập kế hoạch và phân bổ tài nguyên: Thuật toán tham lam có thể được sử dụng để lên lịch công việc hoặc phân bổ tài nguyên một cách hiệu quả.

+ Vấn đề thay đổi tiền xu: Thuật toán tham lam có thể được sử dụng để thực hiện thay đổi cho một số tiền nhất định với số lượng xu tối thiểu, bằng cách luôn chọn đồng xu có giá trị cao nhất nhỏ hơn số tiền còn lại cần thay đổi.

+ Mã hóa Huffman: Thuật toán tham lam có thể được sử dụng để tạo mã không có tiền tố để nén dữ liệu, bằng cách xây dựng cây nhị phân theo cách có tính đến tần số của mỗi ký tự.

- *Ưu điểm:*

✓ Nó là Đơn giản và dễ triển khai.

✓ Thường cho ra kết quả nhanh chóng.

✓ Tiêu tốn ít tài nguyên so với các thuật toán phức tạp hơn.

- *Nhược điểm:*

✓ IDFS có thể mất thời gian nếu không gặp được trạng thái mục tiêu khi giới hạn độ sâu tăng lên, nhất là trong trường hợp không gian tìm kiếm lớn và cây tìm kiếm rất sâu.

✓ Nó có thể không tìm thấy tối ưu nếu có nhiều hơn Solution.

✓ Nó không hiệu quả về thời gian vì phải mất O (b ^ L).

✓ Nó có thể gây ra các vòng lặp nếu tìm kiếm cây được sử dụng trên biểu đồ.

2.4.6 Thuật toán UCS

Thuật toán UCS (Uniform Cost Search) là một thuật toán tìm kiếm đường đi ngắn nhất trong đồ thị, sử dụng chi phí để di chuyển từ một đỉnh đến đỉnh khác. UCS tập trung vào việc tìm kiếm đường đi có chi phí tổng cực tiểu từ nút ban đầu đến mục tiêu.

Tìm kiếm chi phí thống nhất là một thuật toán được sử dụng để di chuyển xung

quanh không gian tìm kiếm có trọng số theo hướng để đi từ nút bắt đầu đến một trong các nút kết thúc với chi phí tích lũy tối thiểu. Tìm kiếm này là một thuật toán tìm kiếm không chính xác vì nó hoạt động theo cách mạnh mẽ, tức là nó không xem xét trạng thái của nút hoặc không gian tìm kiếm. Nó được sử dụng để tìm đường dẫn có chi phí tích lũy thấp nhất trong biểu đồ có trọng số trong đó các nút được mở rộng theo chi phí truyền tải của chúng từ nút gốc. Điều này được thực hiện bằng cách sử dụng hàng đợi ưu tiên trong đó ưu tiên của nó là chi phí thấp hơn..

Ứng dụng: Thuật toán UCS (Uniform Cost Search) có rất nhiều ứng dụng trong các lĩnh vực khác nhau:

+ Hệ thống định tuyến mạng: UCS được sử dụng để tìm kiếm đường đi ngắn nhất hoặc đường đi tối ưu giữa các nút trong mạng, giúp định tuyến dữ liệu từ nguồn đến đích một cách hiệu quả.

+ Hệ thống thông tin địa lý (GIS): UCS được áp dụng trong hệ thống thông tin địa lý để tìm đường đi ngắn nhất giữa hai địa điểm, tối ưu hóa đường đi dựa trên chi phí hoặc thời gian di chuyển.

+ Quản lý tài nguyên và lập lịch: Trong quản lý tài nguyên và lập lịch công việc, UCS có thể được sử dụng để lập lịch công việc sao cho đạt được kết quả tối ưu dựa trên chi phí hoặc thời gian hoàn thành.

- *Ưu điểm:*

✓ Tìm kiếm đường đi ngắn nhất.

✓ Đảm bảo tìm ra đường đi tối ưu dựa trên chi phí.

- *Nhược điểm:*

✓ Đòi hỏi lưu trữ nhiều thông tin về chi phí và các trạng thái, đặc biệt trong các đồ thị lớn, điều này có thể tốn tài nguyên bộ nhớ lớn.

✓ Có thể gặp vấn đề với hiệu suất khi chi phí giữa các trạng thái rất khác nhau.

✓ Không cần quan tâm đến số lượng các bước liên quan đến tìm kiếm và chỉ quan tâm đến chi phí đường dẫn.

2.4.7 Thuật toán A Star

Thuật toán A* là giải thuật tìm kiếm trong đồ thị, tìm đường đi từ một đỉnh hiện tại đến đỉnh đích có sử dụng hàm để ước lượng khoảng cách hay còn gọi là hàm Heuristic.

Thuật toán A* lưu giữ một tập hợp các lời giải chưa được hoàn chỉnh, có nghĩa là những đường đi qua đồ thị cho trước, bắt đầu từ nút xuất phát. Để tính được đầy đủ thông tin đánh giá heuristic cho mỗi trạng thái trong không gian tìm kiếm của bài toán, một hàm đánh giá heuristic f(n) được định nghĩa đó là tổng của hai yếu tố:

$$f(n) = g(n) + h(n)$$

Trong đó:

+ g(n) là chi phí tính từ trạng thái xuất phát đến trạng thái hiện tại đang xét đến.

+ h(n) là hàm heuristic ước lượng chi phí đi từ trạng thái hiện tại tới đích.

Hàm f(n) có giá trị càng thấp thì trạng thái đó càng có độ ưu tiên, triển vọng cao.

Ứng dụng: Thuật toán A* (A-star) là một trong những thuật toán quan trọng và linh hoạt được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau vì khả năng tìm kiếm đường đi ngắn nhất một cách hiệu quả.

+ Trò chơi và trí tuệ nhân tạo: Trong trò chơi, A* được sử dụng để tìm đường đi ngắn nhất hoặc tối ưu cho các đơn vị hoặc người chơi.

+ Hệ thống định tuyến và điều khiển di chuyển: Trong hệ thống định tuyến mạng, A* được sử dụng để tìm đường đi ngắn nhất giữa các nút trong mạng.

- *Ưu điểm:*

✓ Tìm kiếm Tìm kiếm đường đi ngắn nhất.

✓ Tự động điều chỉnh đường đi để hướng tới mục tiêu thông qua hàm heuristic, giúp tối ưu hóa thời gian tìm kiếm.

- *Nhược điểm:*

✓ A* rất linh động nhưng vẫn gặp một khuyết điểm cơ bản – giống như chiến

lược tìm kiếm chiều rộng – đó là tốn khá nhiều bộ nhớ để lưu lại những trạng thái đã đi qua.

- ✓ Hiệu suất phụ thuộc rất lớn vào chất lượng của hàm heuristic, nếu hàm này không chính xác, thuật toán có thể không đưa ra lời giải tối ưu

2.4.8 Thuật toán Hill Climbing

Thuật toán Hill Climbing (leo đồi) là một thuật toán tìm kiếm cục bộ (local search) trong trí tuệ nhân tạo, được sử dụng để tìm giải pháp gần nhất hoặc tối ưu nhất trong không gian tìm kiếm bằng cách di chuyển tới các trạng thái kế tiếp mà có giá trị đánh giá tốt hơn so với trạng thái hiện tại. Thuật toán này tập trung vào việc tối ưu hóa một hàm mục tiêu mà không quan tâm đến việc có thể đạt được một giải pháp toàn cục hay không.

Giải thuật Hill Climbing dễ dàng tìm thấy một giải pháp tốt cục bộ (local optimum) nhưng khó tìm thấy giải pháp tối ưu (global optinum) trong tất cả các giải pháp được đưa ra (search space). Hill Climbing phù hợp để giải các bài toán "convex" (dịch tạm: lồi) như là tìm kiếm đơn giản trong lập trình tuyến tính, tìm kiếm nhị phân.

Ứng dụng: Thuật toán Hill Climbing được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau vì khả năng tìm kiếm đường đi ngắn nhất một cách hiệu quả.

+ Tối ưu hóa và tìm kiếm cục bộ: Hill Climbing được sử dụng trong các bài toán tối ưu hóa để tìm kiếm giải pháp tốt nhất trong không gian tìm kiếm.

+ Tối ưu hóa trong mô phỏng và trực quan hóa: Trong mô phỏng và trực quan hóa, Hill Climbing có thể được sử dụng để tối ưu hóa hiệu suất mô phỏng hoặc thiết kế các hệ thống trực quan.

Ưu điểm:

- ✓ Phương pháp tìm kiếm leo đồi chú trọng tìm hướng đi dễ dẫn đến trạng thái đích nhất. Cách đó được đưa ra nhằm làm giảm công sức tìm kiếm..

- *Nhược điểm:*

- ✓ Cực trị địa phương: nút đang xét tốt hơn các nút lân cận, nhưng đó không phải

là phương án tốt nhất trong toàn thể, ví vậy có thể phải quay lui về nút trước để đi theo hướng khác. Giải pháp này đòi hỏi ghi nhớ lại nhiều đường đi.

- ✓ Cao nguyên: Các giá trị của các phương án như nhau, không xác định được ngay hướng nào là tốt hơn trong vùng lân cận.

2.4.9 Thuật toán Beam Search

Thuật toán Local beam search là một thuật toán tìm kiếm rất hữu ích trong việcgiải quyết các bài toán tối ưu. Nó cho phép tìm kiếm các giải pháp tốt nhất trong một tập hợp các giải pháp có thể. Vì vậy, việc tìm hiểu và nghiên cứu thuật toán này là rất quan trọng và có ý nghĩa trong việc nghiên cứu và giải quyết các bài toántối ưu. Ngoài ra, Local beam search cũng là một trong những thuật toán được sửdụng phổ biến trong lĩnh vực trí tuệ nhân tạo và học máy. Vì vậy, việc nắm vững kiến thức về thuật toán này cũng là một bước quan trọng để có thể phát triển các ứng dụng trí tuệ nhân tạo và học máy trong tương lai

Nó thuộc nhóm các thuật toán tìm kiếm địa phương và thường được sử dụng khi không thể áp dụng các phương pháp tìm kiếmtoàn cục, như thuật toán tìm kiếm theo chiều sâu hay tìm kiếm theo chiều rộng.Thuật toán Local beam search thường được sử dụng trong các bài toán tối ưu hóa đa dạng như tối ưu hóa màu sắc đồ thị, tối ưu hóa lịch trình đi lại hay tối ưuhóa các hàm số phi tuyến. Tuy nhiên, cũng có những hạn chế của thuật toán này, ví dụ như khó khăn trong việc tránh các điểm cực đại địa phương

Ứng dụng: Thuật toán Local beam search có nhiều ứng dụng trong các lĩnh vực khác nhau, bao gồm:

+ Tối ưu hóa: Local beam search có thể được sử dụng để tìm kiếm giải pháp tối ưu cho các bài toán tối ưu hóa trong các lĩnh vực như quản lý sản xuất, quản lý dự án, kế hoạch hóa năng lượng, kế hoạch hóa nguồn lực, quản lý chuỗi cung ứng, v.v.

+ Trò chơi: Local beam search có thể được sử dụng để tìm kiếm giải pháp tối ưu cho các trò chơi như trò chơi cờ vua, trò chơi đi cờ, v.v.

+ Tìm kiếm đường đi: Local beam search có thể được sử dụng để tìm kiếm đường đi tối ưu cho các bài toán như tìm kiếm đường đi ngắn nhất trong các mạng lưới, tìm kiếm đường đi tối ưu cho robot di động, v.v.

+ Học máy: Local beam search có thể được sử dụng để giải quyết các bài toán trong lĩnh vực học máy, chẳng hạn như tối ưu hóa các hàm mất mát, tìm kiếm các siêu tham số tối ưu cho các mô hình học máy, v.v.

+ Xử lý ngôn ngữ tự nhiên: Local beam search có thể được sử dụng để giải quyết các bài toán trong lĩnh vực xử lý ngôn ngữ tự nhiên, chẳng hạn như tìm kiếm các câu trả lời phù hợp cho các câu hỏi, tìm kiếm các từ đồng nghĩa hoặc đối nghịch trong văn bản, v.v.

Ưu điểm:

- ✓ Tốc độ nhanh
- ✓ Dễ dàng cài đặt
- ✓ Dễ dàng mở rộng
- ✓ Không yêu cầu bộ nhớ lớn

- *Nhược điểm:*

- ✓ Dễ bị rơi vào cực tiểu địa phương
- ✓ Yêu cầu lựa chọn thủ công của tham số
- ✓ Không đảm bảo tìm được giải pháp tối ưu toàn cục

CHƯƠNG 3. PHÂN TÍCH, THIẾT KẾ GIẢI PHÁP

3.1 Mô tả dữ liệu và thiết lập tham số

Có một class có tên là State dùng để chứa giá trị khởi tạo ban đầu, hàm heuristic và hàm truy vết đường đi

```
class state:
    def __init__(self, board, state_parent, cost, list_check_point, algorithm):
        self.board = board
        self.state_parent = state_parent
        self.cost = cost
        self.depth = cost
        self.heuristic = 0
        self.algorithm = algorithm
        self.check_points = deepcopy(list_check_point)

    ''' FUNCTION TO BACKTRACK TO THE FIRST IF THE CURRENT STATE IS GOAL '''
    def get_line(self):
        path = []
        current_state = self
        while current_state is not None:
            path.append(current_state.board)
            current_state = current_state.state_parent
        path.reverse()
        return path

    def h(self):
        list_boxes = find_boxes_position(self.board)
        if self.heuristic == 0: # Convert the lists to numpy arrays
            array_boxes = np.array(list_boxes)
            array_check_points = np.array(self.check_points) # Calculate the
            absolute sum of the differences
            self.heuristic = np.sum(np.abs(array_boxes - array_check_points))
        return self.heuristic

    def depth(self):
        return self.depth

    def g(self):
        return self.cost

    def __lt__(self, other):
        if self.algorithm == "ASTAR":
            return (self.h() + self.g()) < (other.h() + other.g())
        elif self.algorithm == "GREEDY" or self.algorithm == "BEAM"
            return (self.h()) < (other.h())
        elif self.algorithm == "UCS":
            return (self.g()) < (other.g())
        else:
            return False
```

- Phần __init__:

+ Đầu vào (Input):

- board: Đại diện cho bảng trạng thái hiện tại của trò chơi.
- state_parent: Trạng thái cha, trạng thái trước đó của trò chơi.
- cost: Chi phí (khoảng cách, số bước di chuyển) từ trạng thái ban đầu đến trạng thái hiện tại.
- list_check_point: Danh sách các điểm kiểm tra trên bảng.
- algorithm: Thuật toán được sử dụng để tìm kiếm (ví dụ: A*, Greedy, UCS, etc.).

+ Công việc (Work):

- Khởi tạo một trạng thái mới với các thông tin như board, state_parent, cost, depth (độ sâu ban đầu là cost), heuristic (ban đầu là 0), algorithm, và sao chép danh sách list_check_point.

- Phần get_line:

+ Công việc:

- Trả về một danh sách các trạng thái từ trạng thái hiện tại đến trạng thái đầu tiên. Điều này được thực hiện bằng cách backtracking từ trạng thái hiện tại đến trạng thái cha, trạng thái cha của trạng thái cha, và tiếp tục cho đến khi đến trạng thái đầu tiên.

- Phần h:

+ Công việc:

- Tính toán hàm heuristic (h) được sử dụng trong các thuật toán như Greedy, A*.
- Nếu giá trị của heuristic là 0, hàm này sẽ tính toán giá trị của heuristic dựa trên vị trí của các hộp (boxes) trên bảng và các điểm kiểm tra (check points).
- Hàm này sử dụng sự khác biệt tuyệt đối (absolute differences) giữa vị trí của các hộp

3.2 Các phương thức chính

```
''' FIND THE PLAYER'S CURRENT POSITION IN A BOARD '''
def find_position_player(board):
    array = np.array(board)
    x, y = np.where(array == '@')
    if len(x) > 0 and len(y) > 0:
      return (x[0], y[0])
    else:
      return (-1, -1) # error board


''' COMPARE 2 BOARDS '''
def compare_matrix(board_A, board_B):
    array_A = np.array(board_A)
    array_B = np.array(board_B)
    return np.array_equal(array_A, array_B)


''' CHECK WHETHER THE BOARD IS GOAL OR NOT '''
def check_win(board, list_check_point):
    try:
        return all(board[p[0]][p[1]] == '$' for p in list_check_point)
    except:
        return None


''' CHECK WHETHER A SINGLE BOX IS ON A CHECKPOINT '''
def is_box_on_check_point(box, list_check_point):
    return box in list_check_point


''' CHECK WHETHER A SIGNLE BOX IS STUCK IN THE CORNER '''
def check_in_corner(board, x, y, list_check_point):
    '''return true if board[x][y] in corner'''

    if (
        (board[x-1][y-1] == '#' and board[x-1][y] == '#' and
         board[x][y-1]    == '#') or
        (board[x+1][y-1] == '#' and board[x+1][y] == '#' and
         board[x][y-1] == '#') or
        (board[x-1][y+1] == '#' and board[x-1][y] == '#' and
         board[x][y+1] == '#') or
        (board[x+1][y+1] == '#' and board[x+1][y] == '#' and
         board[x][y+1] == '#')
    ) and not is_box_on_check_point((x, y), list_check_point):
        return True
    return False
```

- Hàm find_position_player(board)

+ Mô tả: Hàm này tìm vị trí hiện tại của người chơi trên bảng (board) của trò chơi.

+ Công việc:

- Chuyển đổi bảng thành một mảng NumPy (array).
- Sử dụng np.where để tìm tất cả các vị trí của ký tự '@' trên bảng.
- Nếu có ít nhất một vị trí được tìm thấy (len(x) > 0 và len(y) > 0), hàm sẽ trả về vị trí đầu tiên của người chơi (x[0], y[0]). Nếu không tìm thấy vị trí của người chơi trên bảng, hàm trả về (-1, -1) để chỉ ra lỗi trong bảng.

- Hàm compare_matrix(board_A, board_B)

+ Mô tả: So sánh hai ma trận (boards) board_A và board_B.

+ Công việc:

- Chuyển đổi các bảng thành các mảng NumPy (array_A và array_B).
- Sử dụng np.array_equal để kiểm tra xem hai ma trận có bằng nhau hay không.
- Trả về True nếu hai ma trận giống nhau và False nếu không.

- Hàm check_win(board, list_check_point)

+ Mô tả: Kiểm tra xem bảng có đạt được trạng thái kết thúc (mục tiêu) hay không, dựa trên danh sách các điểm kiểm tra (check points).

+ Công việc:

- Sử dụng all để kiểm tra xem tất cả các vị trí trong list_check_point có phải là các ô chứa ký tự '$' trên bảng hay không.
- Trả về True nếu tất cả các ô kiểm tra đều chứa hộp '$', ngược lại trả về False.

- Hàm is_box_on_check_point(box, list_check_point)

+ Mô tả: Kiểm tra xem một hộp cụ thể có nằm trên một điểm kiểm tra hay không.

+ Công việc:

- Sử dụng toán tử in để kiểm tra xem box có trong list_check_point hay không.
- Trả về True nếu hộp nằm trên một điểm kiểm tra và False nếu không.

- Hàm check_in_corner(board, x, y, list_check_point)

+ Mô tả: Kiểm tra xem một hộp cụ thể có bị kẹt ở góc của bảng hay không.

+ Công việc:

- Kiểm tra xem ô (x, y) trên bảng có bị kẹt ở một góc nào đó không. Nếu ô đó bị kẹt ở góc và không nằm trên một điểm kiểm tra thì trả về True. Ngược lại trả về False.

```
''' CHECK WHETHER THE BOARD ALREADY EXISTED IN THE TRAVERSED LIST'''
def is_board_exist(board, list_state):
    '''return true if has same board in list'''
    for state in list_state:
        if compare_matrix(state.board, board):
            return True
    return False


''' CHECK WHETHER ALL BOXES ARE STUCK '''
def is_all_boxes_stuck(board, list_check_point):
    box_positions = find_boxes_position(board)
    result = True
    for box_position in box_positions:
        if is_box_on_check_point(box_position, list_check_point):
            return False
        if is_box_can_be_moved(board, box_position):
            result = False
    return result

''' CHECK WHETHER AT LEAST ONE BOX IS STUCK IN THE CORNER'''
def is_board_can_not_win(board, list_check_point):
    '''return true if box in corner of wall -> can't win'''
    array = np.array(board)
    x, y = np.where(array == '$')

    for i in range(len(x)):
        if check_in_corner(board, x[i], y[i], list_check_point):
            return True
    return False


''' FIND ALL BOXES' POSITIONS '''
def find_boxes_position(board):
    array = np.array(board)
    x, y = np.where(array == '$')
    xy = np.transpose([x, y])
    result = xy.tolist()
    return result
```

- Hàm is_board_exist(board, list_state)

+ Mô tả: Kiểm tra xem một bảng cụ thể đã tồn tại trong danh sách các trạng thái đã duyệt qua hay chưa.

+ Công việc:

- Duyệt qua từng trạng thái trong danh sách list_state.
- Sử dụng hàm compare_matrix để so sánh bảng của trạng thái hiện tại với bảng trong từng trạng thái trong danh sách.
- Trả về True nếu bảng đã tồn tại trong danh sách và False nếu không.

- Hàm is_all_boxes_stuck(board, list_check_point)

+ Mô tả: Kiểm tra xem tất cả các hộp có bị kẹt hay không.

+ Công việc:

- Lấy danh sách các vị trí của các hộp trên bảng sử dụng hàm find_boxes_position.
- Khởi tạo biến result là True.
- Duyệt qua từng vị trí của các hộp:
- Nếu một hộp đang nằm trên một điểm kiểm tra (list_check_point), hàm sẽ trả về False.
- Nếu một hộp có thể di chuyển được (is_box_can_be_moved trả về True), result sẽ được gán lại là False.
- Trả về giá trị của result sau khi duyệt qua tất cả các hộp. Nếu tất cả các hộp đều bị kẹt, result sẽ là True.

- Hàm is_board_can_not_win(board, list_check_point)

+ Mô tả: Kiểm tra xem có ít nhất một hộp bị kẹt ở góc không thể thắng hay không.

+ Công việc:

- Lấy vị trí của tất cả các hộp ($) trên bảng.
- Duyệt qua từng vị trí của các hộp:
- Nếu một hộp bị kẹt ở góc (sử dụng hàm check_in_corner), và không nằm trên một điểm kiểm tra, hàm sẽ trả về True.
- Nếu không có hộp nào bị kẹt ở góc, hoặc nếu tất cả các hộp đều nằm trên điểm kiểm

tra, hàm sẽ trả về False.

- Hàm find_boxes_position(board)

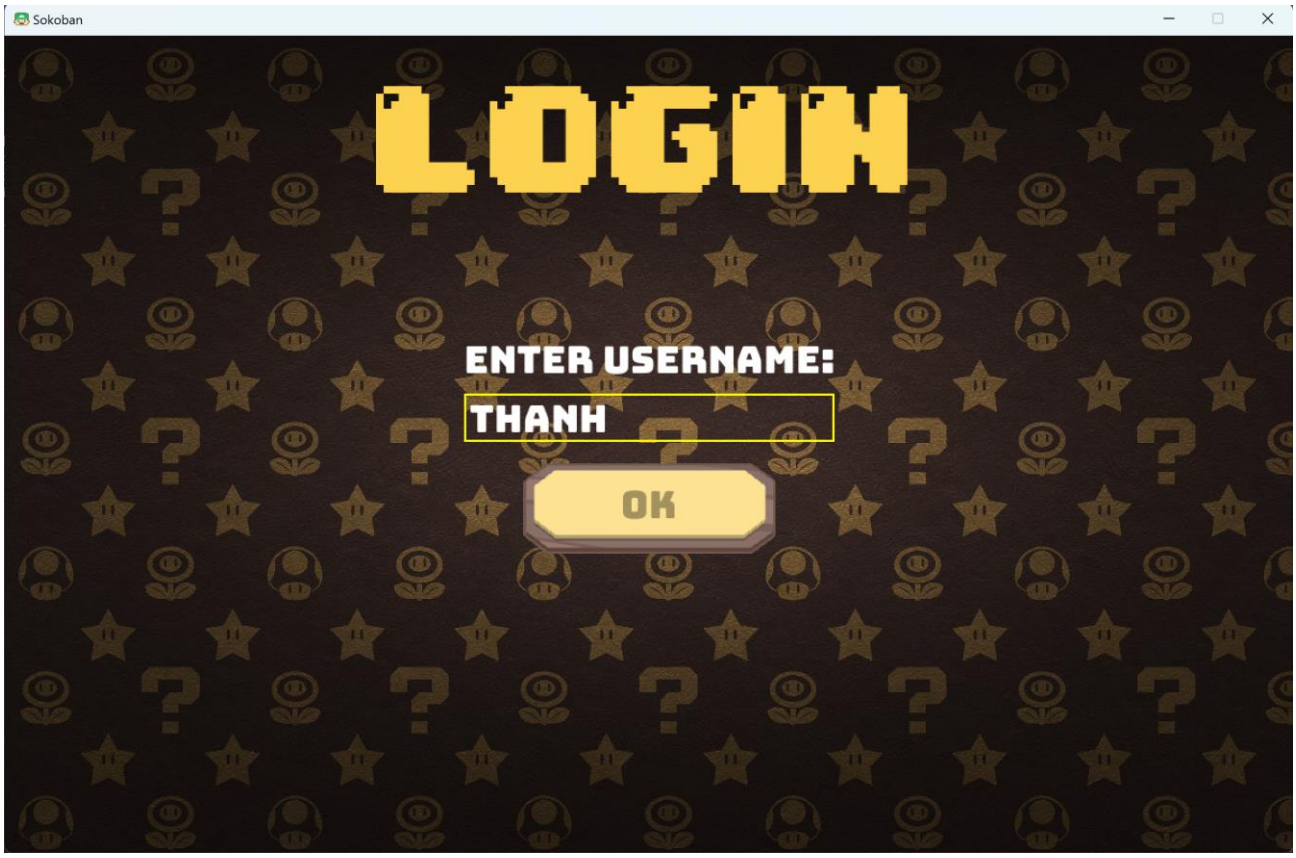
+ Mô tả: Tìm tất cả các vị trí của các hộp trên bảng.

+ Công việc:

- Chuyển đổi bảng thành một mảng NumPy (array).
- Sử dụng np.where để tìm tất cả các vị trí của ký tự $ trên bảng.
- Chuyển đổi các vị trí này thành danh sách (list) và trả về.

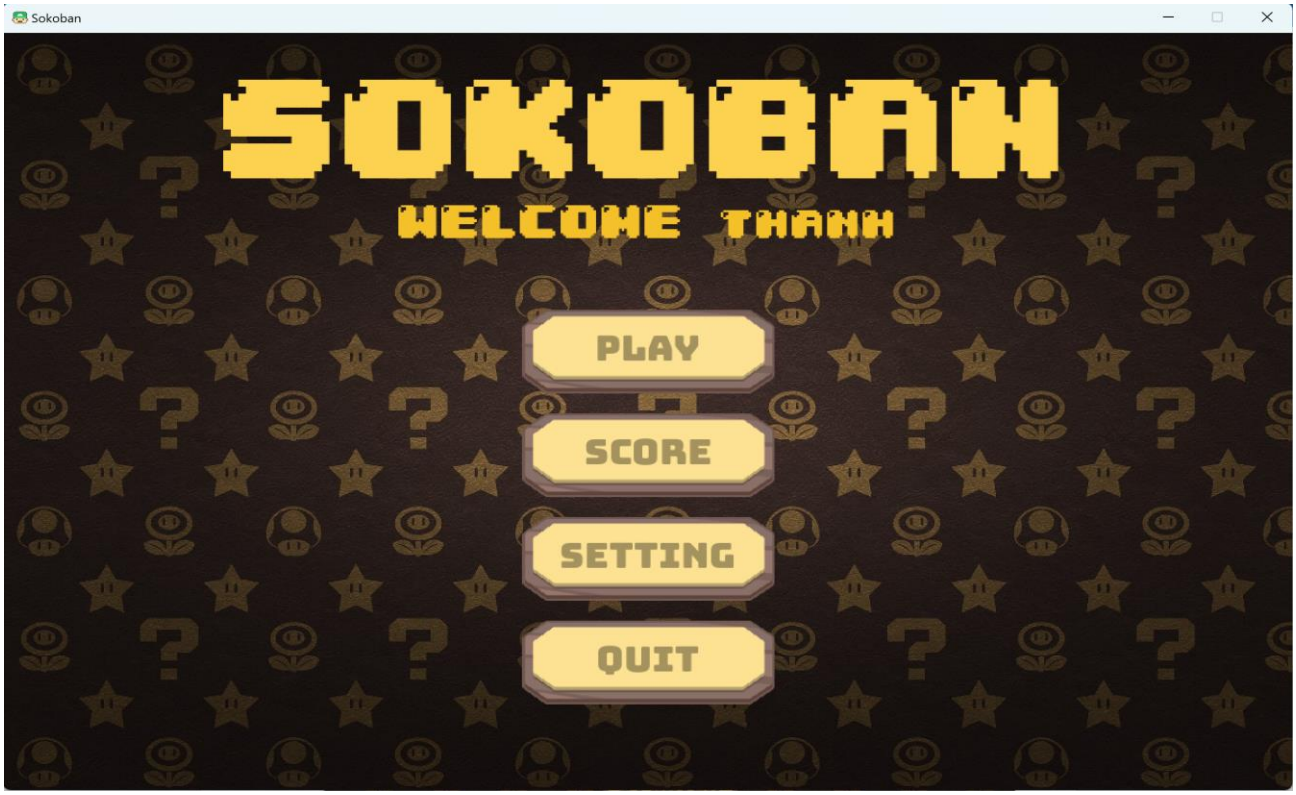
CHƯƠNG 4. CÀI ĐẶT, ĐÁNH GIÁ KẾT QUẢ

4.1 Cài đặt chương trình



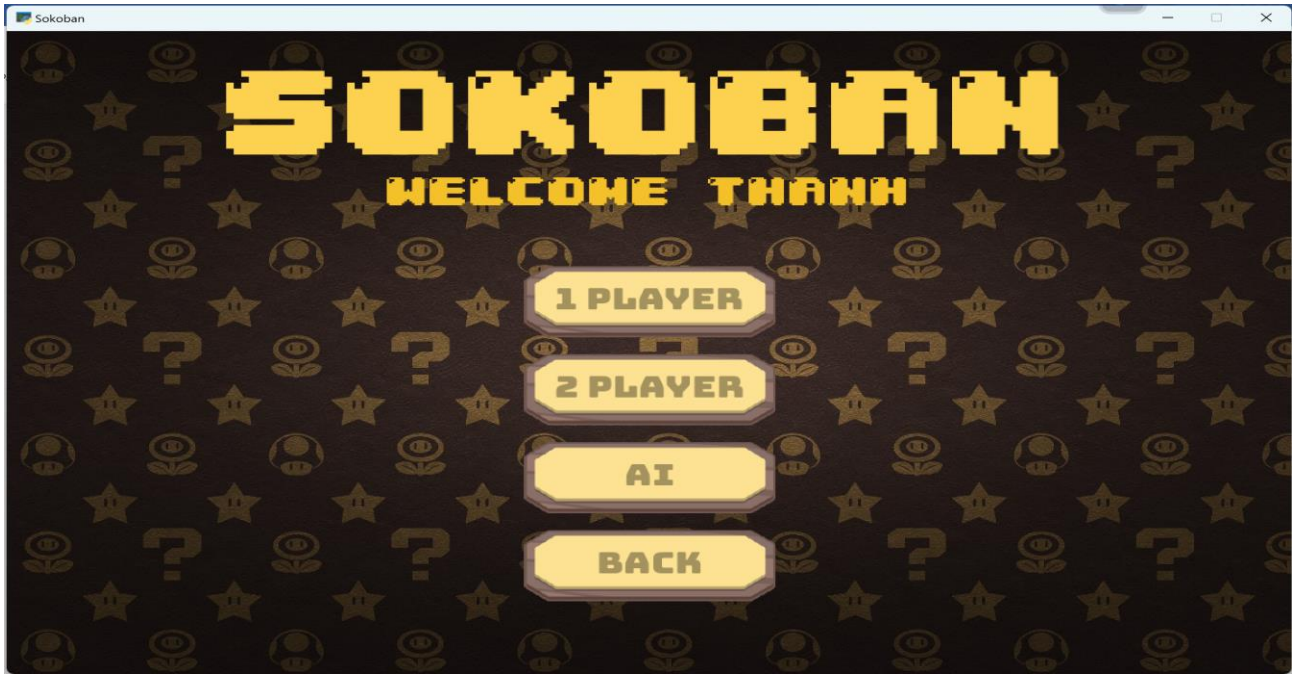
***Hình 5**. Form đăng nhập*

Người chơi phải đăng nhập để được chơi game, tên người chơi và số màn người chơi đã chơi được sẽ được lưu lại và khi lần tiếp theo người chơi đăng nhập sẽ có thể chơi tiếp tục lần chơi trước



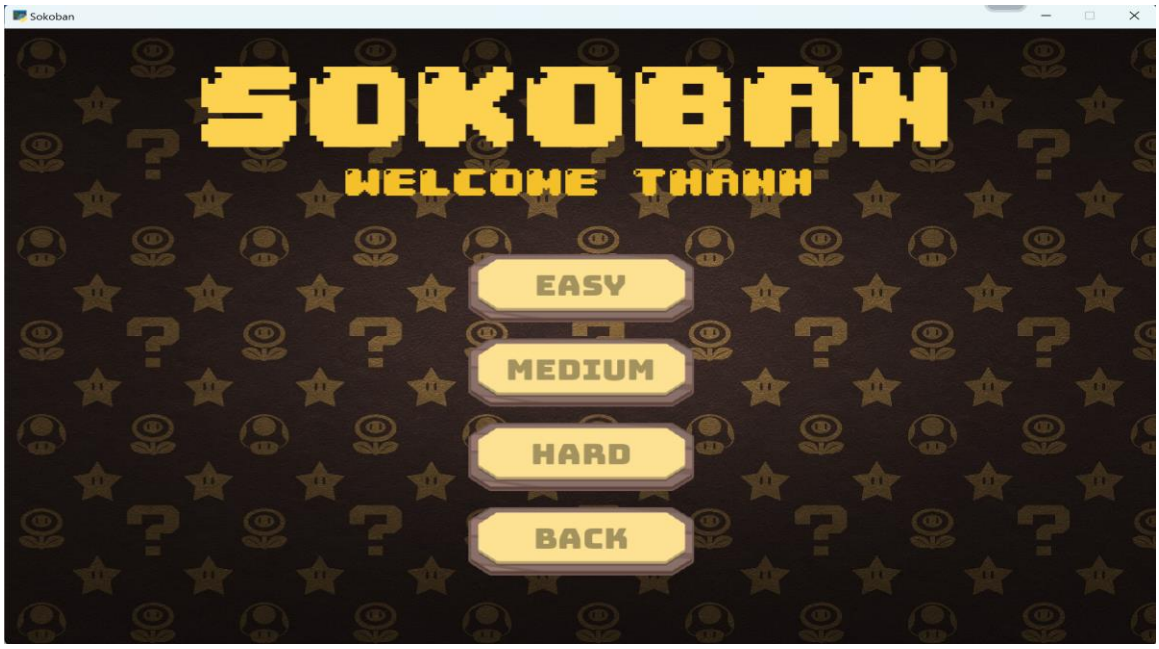
***Hình 6**. Giao diện sau khi đăng nhập*

Người chơi có các lựa chọn: Play để chơi game, Score để xem điểm, Setting để điều chỉnh các nút di chuyển, Quit để kết thúc game.



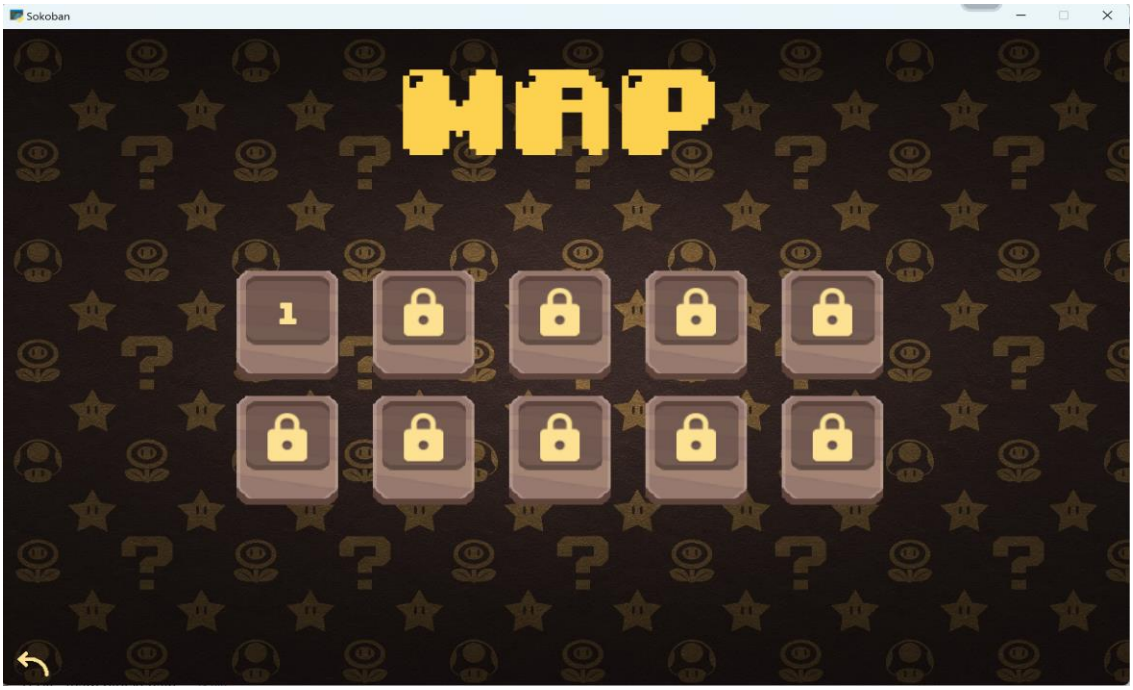
***Hình 7**. Giao diện khi nhấn Play*

Có các hình thức chơi như: Chơi 1 người, Chơi 2 người, AI chơi, và quay trở lại giao diện Home.



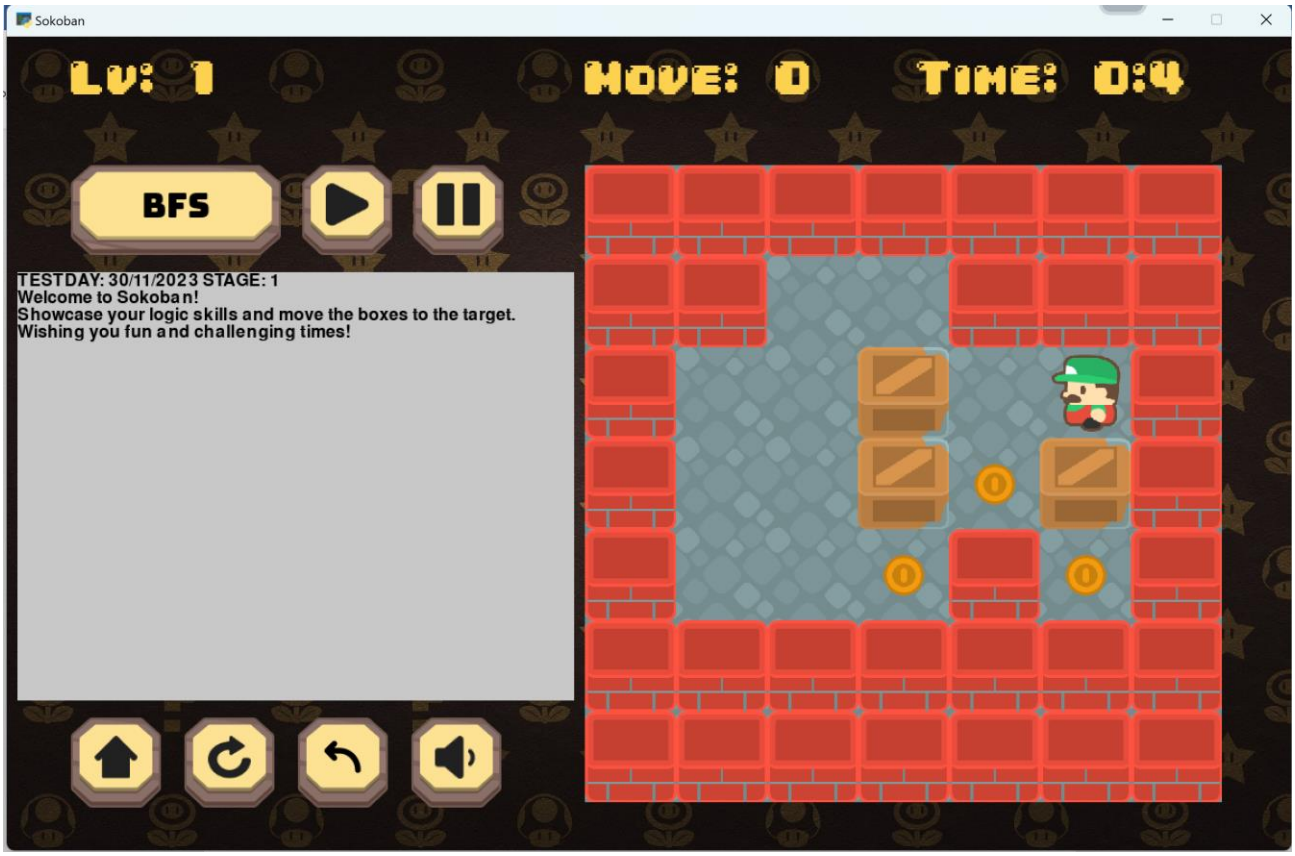
Hình 8. *Giao diện chơi 1 player*

Với hình thức chơi 1 player, sẽ có các mức độ: Dễ, trung bình, khó. Mỗi mức độ sẽ có các level khác nhau.



Hình 9. *Giao diện khi chọn một mức độ*

Với mỗi mức độ, sẽ có từng map tương ứng. Khi người chơi vượt qua map thứ nhất sẽ tự động đượng mở khóa map thứ 2, tương tự cho các map sau.



Hình 10. *Giao diện chơi 1 level*

Giao diện chơi game sẽ hiển thị level người chơi đang chơi, thời gian vào game, số bước di chuyển. Có các nút đi kèm các chức năng như:

+ Nút Home: Quay về trang chủ

+ Nút Reset: Quay lại trạng thái ban đầu

+ Nút Back: Quay lại bước đi trước đó

+ Nút Volume: Điều chỉnh âm lượng

Ngoài ra, nếu như người chơi không chơi được, có thể nhờ AI trợ giúp. Game bao gồm 9 thuật toán AI: BFS, DFS, IDFS, Greedy, UCS, Dijkatra, A*, Beam Search và Hill Climbing

+ Nút play: Chơi bằng thuật toán

+ Nút Stop: Dừng chơi bằng thuật toán

Ô Textbox, sẽ hiện thông tin về level, số đỉnh được duyệt, thời gian tìm kiếm lời giải của

mỗi thuật toán để có thể so sánh sự tối ưu.

Hình 11. *Giao diện sau khi chiến thắng map*

Mỗi lần người chơi chiến thắng map đó, sẽ hiện thi thông báo điểm, số bước đi và thời gian chơi. Người chơi có thể lựa chọn chơi lại, quay về giao diện Home hoặc tiếp tục qua màn mới.

Hình 12. *Giao diện chơi 2 players*

Với hình thức 2 người chơi, mỗi người chơi sẽ có thời gian 5 phút. Ai qua được nhiều màn hơn người đó sẽ thắng

Có các nút chức năng như:

+ Nút Home: Quay về trang chủ

+ Nút Reset: Quay lại trạng thái ban đầu

+ Nút Back: Quay lại bước đi trước đó

+ Nút Volume: Điều chỉnh âm lượng

+ Nút Resume: Dừng thời gian

4.2 Phân tích kết quả

4.2.1 Phân tích map số 1

Khi nhấn vào Map 1, sẽ hiển thị giao diện chơi game

Hình 13. *Giao diện chơi Map 1*

Giao diện này có các nút chức năng và một Button hiển thị tên thuật toán . Nút Play khi nhấn vào thì thuật toán sẽ được thực thi

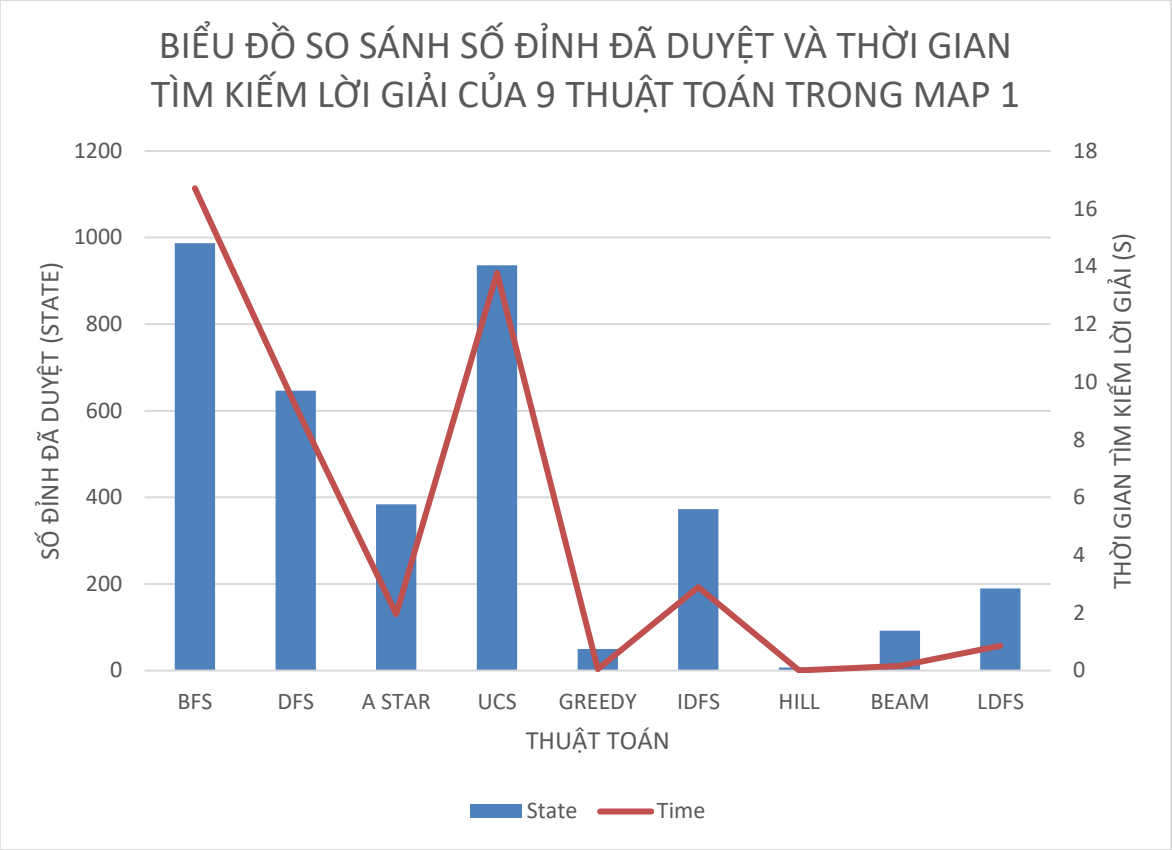
Có 9 thuật toán để lựa chọn để giải Sokoban: BFS, DFS, IDFS, Greedy, UCS, Dijkatra, A*, Beam Search và Hill Climbing

Khi nhấn vào BFS sẽ thực hiện thuật toán BFS tương tự như các button còn lại. Sau khi thực hiện xong thuật toán sẽ hiển thị thời gian thực hiện thuật toán, số bước di chuyển để được trạng thái đích

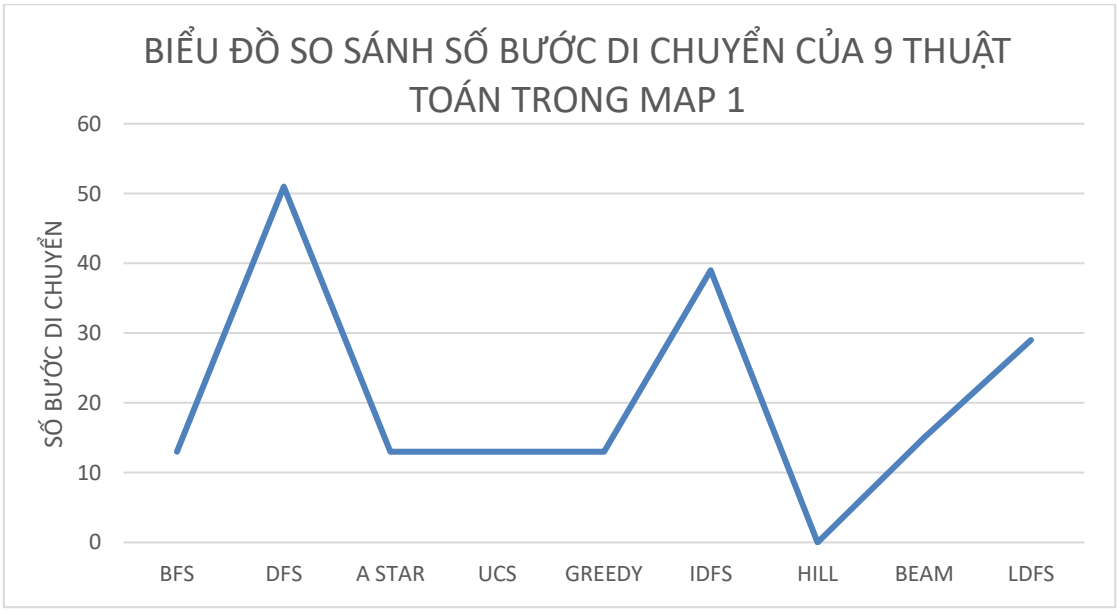
Hình 14. *Áp dụng các thuật toán vào Map 1*

Alogorithm	State	Time	Move
BFS	987	16.709	13
DFS	646	9.148	51
A STAR	384	1.96	13
UCS	936	13.78	13
GREEDY	50	0.053	13
IDFS	373	2.891	39
HILL	7	0.005	NULL
BEAM	92	0.167	15
LDFS	190	0.86	29

Bảng 1. *So sánh số đỉnh, thời gian và số bước đi của các thuật toán trong Map 1*



Hình 15. *Biểu đồ so sánh số đỉnh duyệt và thời gian tìm kiếm của 9 thuật toán vào Map 1*



Hình 16. *Biểu đồ so sánh số bước đi 9 thuật toán trong Map 1*

Nhận xét: Nhìn vào biểu đồ dựa trên phân thích Map số 1, ta thấy có sự khác nhau rõ rệt về số đỉnh được duyệt, thời gian tìm kiếm và số bước đi của mỗi thuật toán khi áp dụng vào bài toán Sokoban. Về số đỉnh duyệt BFS, UCS và LDFS khá lớn, còn Greedy

có số đỉnh duyệt ít nhất. Về thời gian, BFS và UCS chiếm thời gian lớn nhất, BFS là 16,709s, UCS là 13,78 s. Greedy nhanh nhất là 0,053s. Về số bước di chuyển, BFS, ASTAR, UCS có số bước ít nhất là 13 bước, DFS có số bước nhiều nhất là 59 bước. Thuật toán Hill Climbing không tìm được lời giải.

4.2.2 Phân tích map số 2

Khi nhấn vào Map 2, sẽ hiển thị giao diện chơi game

Hình 17. *Giao diện chơi game Map 2*

Giao diện này có các nút chức năng và một Button hiển thị tên thuật toán . Nút Play khi nhấn vào thì thuật toán sẽ được thực thi

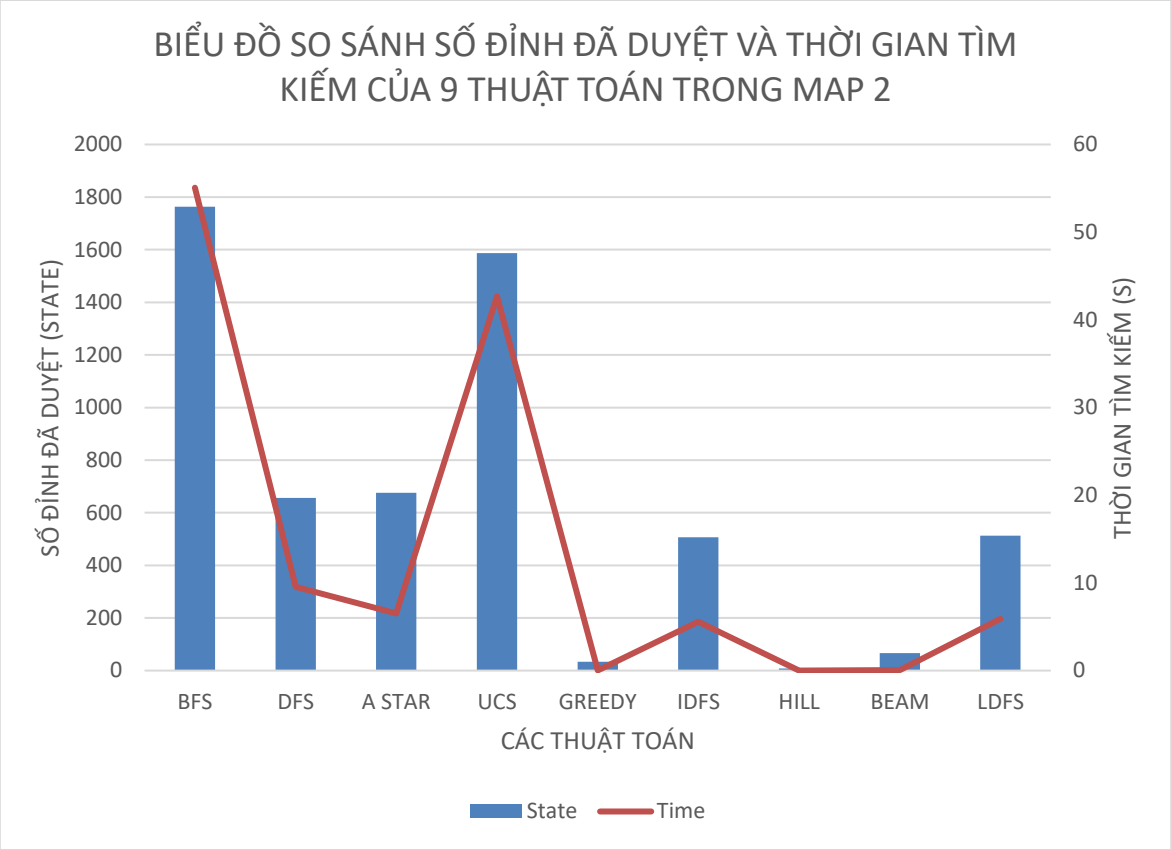
Có 9 thuật toán để lựa chọn để giải Sokoban: BFS, DFS, IDFS, Greedy, UCS, Dijkatra, A*, Beam Search và Hill Climbing

Khi nhấn vào BFS sẽ thực hiện thuật toán BFS tương tự như các button còn lại. Sau khi thực hiện xong thuật toán sẽ hiển thị thời gian thực hiện thuật toán, số bước di chuyển để được trạng thái đích

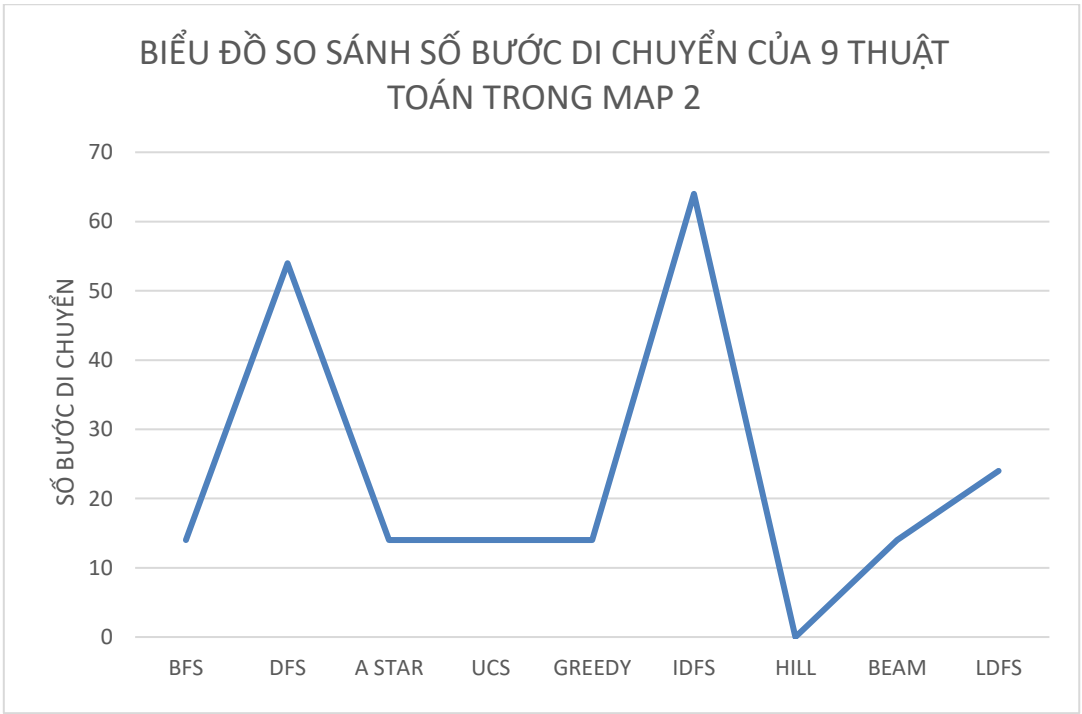
Hình 18. *Áp dụng thuật toán LDFS vào Map 2*

Alogorithm	State	Time	Move
BFS	1763	55.071	14
DFS	656	9.543	54
A STAR	676	6.531	14
UCS	1587	42.682	14
GREEDY	34	0.029	14
IDFS	507	5.564	64
HILL	8	0.004	NULL
BEAM	66	0.074	14
LDFS	513	5.884	24

Bảng 2*. So sánh số đỉnh, thời gian và số bước đi của các thuật toán trong Map 2*



Hình 19. *So sánh số đỉnh duyệt và thời gian tìm kiếm của các thuật toán trong Map 2*



Hình 20. *So sánh số bước đi của 9 thuật toán trong Map 2*

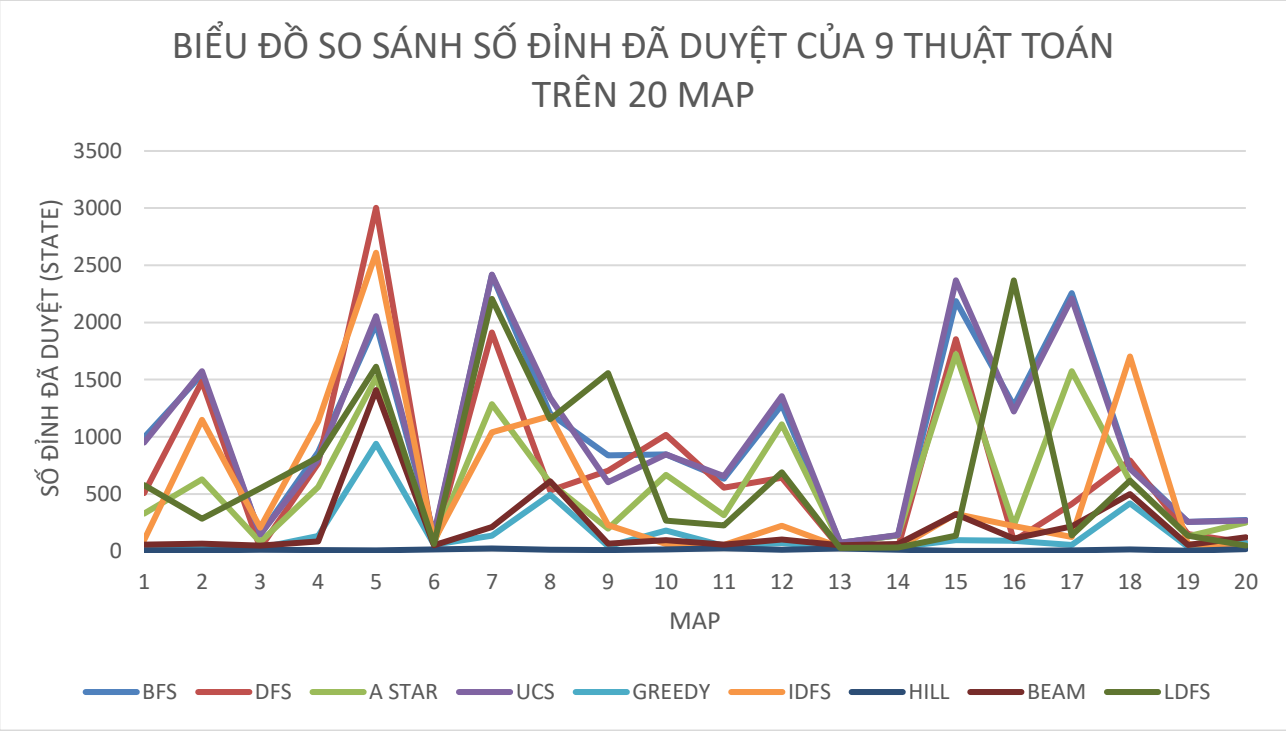
Nhận xét: Nhìn vào biểu đồ dựa trên phân thích Map số 2, ta thấy có sự khác nhau rõ rệt về số đỉnh được duyệt, thời gian tìm kiếm và số bước đi của mỗi thuật toán khi áp dụng vào bài toán Sokoban. Về số đỉnh duyệt BFS, UCS và LDFS khá lớn, còn Greedy có số đỉnh duyệt ít nhất. Về thời gian, BFS chiếm thời gian lớn nhất là 55,071s. Greedy nhanh nhất là 0,029s. Về số bước di chuyển, BFS, ASTAR, UCS và BEAM có số bước ít nhất là 13 bước, IDFS có số bước nhiều nhất là 64 bước, tiếp đến là DFS với 54 bước. Thuật toán Hill Climbing không tìm được lời giải.

4.2.3 Phân tích tổng hợp 20 map

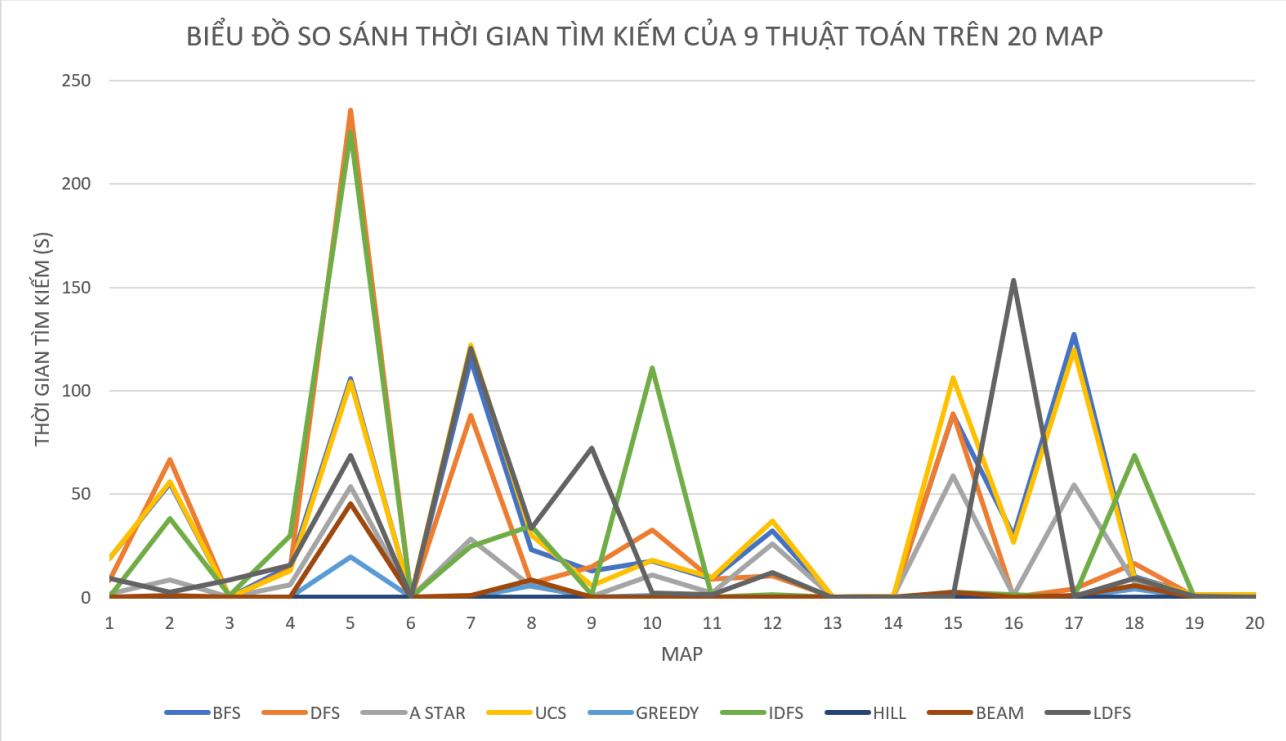
Sau khi thống kê số liệu từ 20 map của game.

Số thứ tự	BFS			DFS			A STAR			UCS			GREEDY			IDFS			HILL			BEAM			LDFS		
	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move	States (state)	Time (s)	Move
1	994	18.691	13	506	7.949	33	328	1.934	13	950	18.66	13	41	0.04	13	97	0.03	27	7	0.006	NULL	56	0.12	15	577	9.13	39
2	1547	55.31	14	1483	66.72	52	629	8.39	14	1575	55.9	14	35	0.03	14	1149	38.18	46	5	0.004	NULL	64	0.79	14	264	2.38	32
3	144	0.37	11	26	0.02	15	78	0.12	11	146	0.37	11	29	0.02	11	204	1.17	21	12	0.09	NULL	48	0.06	13	548	8.372	23
4	866	15.25	20	767	13.61	32	561	6.32	21	800	12.84	20	132	0.04	21	1133	29.81	34	9	0.006	NULL	84	0.16	21	821	15.68	50
5	1988	105.914	23	3003	235.85	41	1517	53.511	23	2056	104.351	23	939	19.512	29	2610	225.011	37	5	0.003	NULL	1409	45.361	23	1614	68.61	23
6	136	0.34	12	56	0.08	14	76	0.1	12	150	0.43	12	58	0.07	14	90	0.19	12	15	0.01	NULL	50	0.06	12	61	0.07	14
7	2415	115.081	17	1913	88.089	31	1285	28.447	17	2419	121.956	17	134	0.37	21	1039	24.745	17	23	0.02	NULL	212	0.94	17	2205	120.399	25
8	1202	23.202	12	530	7.006	16	597	5.283	13	1342	30.399	12	493	5.648	28	1183	34.442	40	13	0.01	NULL	612	8.308	28	1153	33.537	28
9	837	12.771	10	703	14.657	20	197	0.707	10	603	5.932	10	43	0.05	10	227	1.567	10	9	0.008	NULL	65	0.104	10	1558	72.237	28
10	845	17.607	23	1017	32.659	51	668	11.04	23	845	17.896	23	181	0.99	23	63	111	25	16	0.016	NULL	96	0.26	23	268	2.053	25
11	634	9.039	13	554	8.782	33	313	2.056	13	657	9.743	13	45	0.061	13	55	0.084	19	25	0.032	NULL	57	0.088	13	224	1.341	25
12	1280	32.253	19	643	10.535	41	1108	25.863	23	1355	37.065	19	71	0.136	25	223	1.342	45	12	0.009	NULL	102	0.255	25	689	12.109	45
13	73	0.115	18	27	0.021	22	70	0.113	18	73	0.115	18	27	0.024	18	34	0.036	24	24	0.019	24	52	0.086	18	28	0.02	24
14	139	0.421	14	32	0.026	20	137	0.394	14	142	0.393	14	33	0.034	18	27	0.02	18	11	0.006	NULL	61	0.091	18	32	0.026	18
15	2186	88.369	19	1853	1853	51	1723	58.822	25	2370	106.332	19	95	0.227	27	326	2.559	23	2	0.002	NULL	322	2.578	27	136	0.52	31
16	1271	30.009	19	89	0.199	19	225	0.887	19	1220	26.903	19	90	0.19	25	218	1.202	29	2	0.001	NULL	111	0.257	23	2368	153.476	33
17	2258	127.066	24	411	4.2	32	1575	54.25	24	2208	119.817	24	53	0.071	24	124	0.382	30	5	0.003	NULL	216	1.113	24	139	0.511	30
18	740	9.953	19	794	16.251	41	614	7.343	19	719	9.335	19	418	4.168	31	1702	68.855	55	16	0.013	NULL	498	5.666	31	617	9.323	47
19	255	1.371	15	150	0.559	25	129	0.33	15	256	1.391	15	36	0.043	15	59	0.091	15	4	0.002	NULL	54	0.072	15	136	0.423	25
20	271	1.496	22	74	0.121	34	249	1.278	22	268	1.448	22	72	0.13	24	43	0.043	26	18	0.014	NULL	121	0.318	24	48	0.052	36

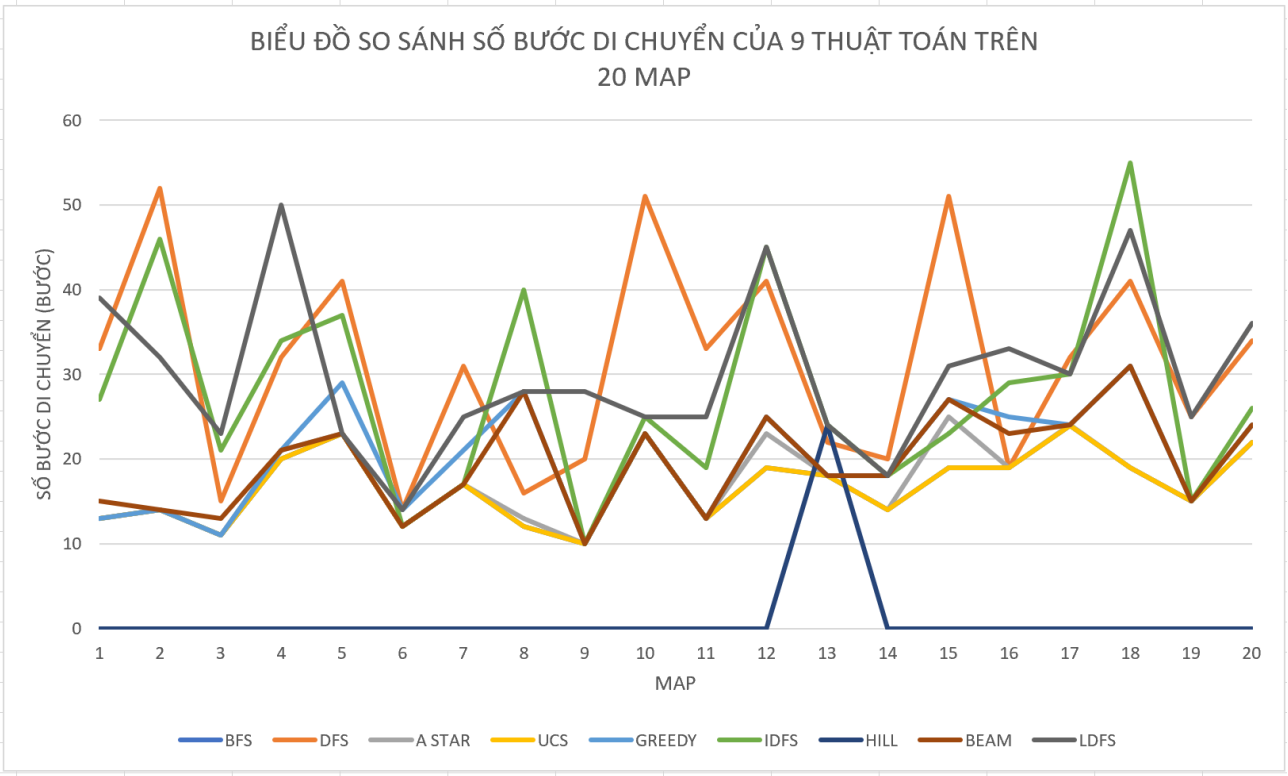
Hình 21*. Số liệu thống kê các thông số của các thuật toán trên 20 map*



Hình 22. *Biểu đồ so sánh số đỉnh đã duyệt của 9 thuật toán trên 20 map*



Hình 23*. Biểu đồ so sánh thời gian tìm kiếm của 9 thuật toán trên 20 map*



***Hình 24**. Biểu đồ so sánh số bước di chuyển của 9 thuật toán trên 20 map*

Nhận xét:

Nhìn vào biểu đồ dựa trên phân thích Map số 2, ta thấy có sự khác nhau rõ rệt về số đỉnh được duyệt, thời gian tìm kiếm và số bước đi của mỗi thuật toán khi áp dụng vào bài toán Sokoban.

Về số đỉnh đã duyệt, thuật toán Greedy duyệt số đỉnh ít nhất, thuật toán BFS và UCS duyệt số đỉnh nhiều nhất.

Về thời gian tìm kiếm, Greedy tìm kiếm nhanh nhất, sau đó là đến Beam, BFS, UCS, DFS có thời gian tìm kiếm khá lớn, lớn hơn các thuật toán còn lại.

Về số bước di chuyển, các thuật toán BFS, UCS, ASTAR có số bước đi bằng nhau và ít nhất, LDFS, IDFS, DFS có số bước di chuyển khá cao (một số map có số bước đi tối ưu).

CHƯƠNG 5. KẾT LUẬN

5.1 Đánh giá kết quả thực hiện được

- Về mặt giải thuật:

+ Đã hoàn thành triển khai thực hiện các thuật toán BFS, DFS, IDFS, LDFS, Greedy, UCS, A*, Beam Search và Hill Climbing ứng dụng chúng vào chương trình

+ Đã cải tiến các thuật toán tối ưu hơn về thời gian xử lý

+ Làm rõ được sự khác nhau giữa các thuật toán và so sánh chúng một cách trực quan thông qua chương trình đã xây dựng.

- Về mặt ứng dụng:

+ Đã hoàn thiện chương trình sử dụng Pygame để mô tả các thuật toán

+ Xây dựng giao diện khá đẹp mắt, đáp ứng được các yêu cầu đã cho như kết hợp 9 thuật toán AI, chế độ 1 người chơi, 2 người chơi, …

+ Người học có thể tự tay xây dựng bản đồ phù hợp với nhu cầu

+ Giao diện thân thiện với người chơi, có thể tích hợp các âm thanh tương ứng với sự kiện trong chương trình.

5.2 Định hướng phát triển

Chương trình tìm kiếm đường đi của nhóm chúng em có tiềm năng phát triển thành một ứng dụng Game Sokoban có thể thu hút nhiều người chơi. Người chơi sẽ có khả năng tùy chỉnh độ khó từng màn chơi với số vật cản tỉ lệ thuận với độ khó được chọn. Tại mỗi màn chơi, người chơi sẽ di chuyển thủ công nhân vật mình để đi từ điểm đầu đến đích. Chương trình sẽ dựa vào đường đi của người chơi để phân tích hiệu quả, đồng thời so sánh đường đi đó với các giải thuật cài sẵn. Tiêu chí so sánh dựa trên thời gian, chi phí, quãng đường. Từ những thông số được so sánh, người dùng sẽ hiểu thêm về cách thức hoạt động của thuật toán và cách tư duy tìm

đường dựa trên giải thuật tương ứng

Qua phần đánh giá ở trên, nhóm chúng em sẽ đề xuất 1 số hướng phát triển trong tương lại để tối ưu giải thuật và tăng trải nghiệm tốt nhất cho người chơi

- Xây dựng thêm chức năng tính số bước di chuyển của người chơi và so sánh với số bước ngắn nhất từng chơi
- Tích hợp các nền tảng công nghệ mới: AR/VR,…
- Tạo ra các cấp độ phức tạp và đa dạng hơn
- Mở rộng phạm vi bản đồ, giúp người chơi phân biệt được ưu nhược điểm của từng phương pháp trong phạm vi lớn.

BẢNG PHÂN CÔNG CÔNG VIỆC

Danh sách thành viên	Công việc thực hiện được	Tỉ lệ hoàn thành
Nguyễn Phú Thành	Xây dựng giao diện chơi game, xây dựng thuật toán Beam , Làm báo cáo Word + PPT	**100%**
Nguyễn Văn Hào	Vẽ map, xây dựng thuật toán Hill, LDFS, IDFS, UCS, Greedy. Đưa thuật toán vào giao diện.	**100%**
Bùi Quốc Khang	Thiết kế map, thiết kế Textbox so sánh số đỉnh duyệt, thời gian tìm kiếm và số bước đi của các thuật toán. Xây dựng thuật toán BFS, DFS, thiết kế các nút Back, Reset, Volume,…	**100%**
Phạm Hùng Phong	Thiết kế giao diện Log in, thiết kế chơi game 2 player, các phần liên quan đến Score, các mức độ, dễ khó trung bình,..	**100%**

Bảng 3*. Bảng phân công nhiệm vụ*

TÀI LIỆU THAM KHẢO

1. Pygame documentation, https://www.pygame.org/docs/ref/display.html
2. Đọc ghi file với thư viện Tkinter, https://www.pythontutorial.net/tkinter/tkinter-open-file-dialog/
3. Thuật toán A*, https://www.geeksforgeeks.org/a-search-algorithm/
4. Thuật toán BFS, https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
5. Thuật toán DFS, https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/
6. Thuật toán IDFS, https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs/
7. Thuật toán LDFS, https://ai-master.gitbooks.io/classic-search/content/what-is-depth-limited-search.html
8. Thuật toán Greedy, https://www.geeksforgeeks.org/greedy-algorithms/
9. Thuật toán UCS, https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/
10. Thuật toán Hill Climbing, https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/
11. Thuật toán Beam Search, https://www.geeksforgeeks.org/introduction-to-beam-search-algorithm/