BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

BÁO CÁO THÍ NGHIỆM/THỰC NGHIỆM

HỌC PHẦN: TRÍ TUỆ NHÂN TẠO

ĐỀ TÀI:

TÌM HIỂU THUẬT TOÁN A* VÀ ỨNG DỤNG VÀO TRÒ CHƠI SOKOBAN

Sinh viên thực hiện:

Họ và tên: Đỗ Thắng Chiến - MSV: 2021605176

Họ và tên: Nguyễn Văn Chiến - MSV: 2021605430

Họ và tên: Vũ Xuân Điệp - MSV: 2021605707

Lớp: 20231IT6043007 Khóa: 16

Nhóm: 16

Người hướng dẫn**:** Ths. Nguyễn Lan Anh

Hà Nội, 11/2023

MỤC LỤC

DANH MỤC HÌNH ẢNH

LỜI CẢM ƠN

Để hoàn thành bản báo cáo này, chúng em đã nhận được rất nhiều sự hướng dẫn từ phía các thầy các cô trong khoa. Sự giảng dạy chu đáo, tận tình và sự giúp đỡ nhiệt tình từ các thầy các cô đã giúp chúng em hiểu ra nhiều vấn đề và hoàn thành bản báo cáo này tốt nhất.

Chúng em tỏ lòng biết ơn sâu sắc với cô Nguyễn Lan Anh, người cô đã tận tình hướng dẫn và giúp đỡ, chỉ bảo nhóm em trong suốt quá trình nghiên cứu đề tài và hoàn thành báo cáo này.

Sau khoảng thời gian cô Nguyễn Lan Anh đưa ra đề tài, chúng em đã rất nỗ lực và cố gắng trong việc tìm hiểu về đề tài. Các bạn trong nhóm cùng các cộng sự đã rất chăm chỉ cũng như giúp đỡ lẫn nhau để cho ra một báo cáo hoàn hảo nhất đến thời điểm hiện tại. Một lần nữa nhóm em xin cảm ơn giảng viên Nguyễn Lan Anh, các bạn trong lớp và tập thể nhóm làm việc đã cùng nhau hoàn thành tốt được bản báo cáo này.

Chúng em xin chân thành cảm ơn!

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, những 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.

Nội dung chính đề tài gồm 3 chương:

- Chương 1: Tìm hiểu thuật toán tìm kiếm trong không gian trạng thái.
- Chương 2: Thuật toán A* áp dụng trong game sokoban.
- Chương 3: Xây dựng và triển khai trò chơi

CHƯƠNG 1. TÌM HIỂU THUẬT TOÁN TÌM KIẾM TRONG KHÔNG GIAN TRẠNG THÁI

1.1. Tổng quan về trí tuệ nhân tạo

1.1.1. Khái niệm

Trí tuệ nhân tạo (AI) là một lĩnh vực trong khoa học máy tính và công nghệ thông tin tập trung vào việc phát triển máy tính và hệ thống có khả năng thực hiện các nhiệm vụ thông minh mà trước đây chỉ có con người có thể thực hiện. AI có mục tiêu tạo ra các chương trình máy tính hoặc máy tính thông minh có khả năng học, tự điều chỉnh và thực hiện các nhiệm vụ mà yêu cầu sự hiểu biết, lý thuyết và khả năng giải quyết vấn đề.

Hình 1:Hình ảnh AI

1.1.2. Phân loại trí tuệ nhân tạo

Bốn loại của trí tuệ nhân tạo đó là: Máy phản ứng, Bộ nhớ hạn chế, Lý thuyết tâm trí, Tự nhận thức.

- *Máy phản ứng (Reactive Machines)*:

Máy phản ứng là cấp độ đơn giản nhất của AI. AI sẽ có khả năng phân tích những động thái khả nghi nhất của mình và đối thủ. Sau đó, sẽ đưa ra giải pháp tốt nhất. Deep Blue của IBM, một cỗ máy được thiết kế để chơi cờ vua với con người. Deep Blue đánh giá các quân cờ trên bàn cờ và phản ứng với chúng, dựa trên các chiến lược cờ vua được mã hóa trước. Deep Blue không học hỏi hoặc cải

thiện khi chơi – nó chỉ đơn giản là ‘phản ứng’. Và nó đánh bại kiện tướng cờ vua Garry Kasparov vào năm 1997.

- *Bộ nhớ hạn chế (Limited Memory)*:

Máy có bộ nhớ hạn chế, có thể giữ lại một số thông tin học được từ việc quan sát các sự kiện hoặc dữ liệu trước đó. AI có thể xây dựng kiến thức bằng cách sử dụng bộ nhớ đó kết hợp với dữ liệu được lập trình sẵn.

VD: Đối với xe không người lái, nhiều cảm biến được trang bị xung quanh xe và ở đầu xe để tính toán khoảng cách với các xe phía trước, công nghệ AI sẽ dự đoán khả năng xảy ra va chạm, từ đó điều chỉnh tốc độ xe phù hợp để giữ an toàn cho xe.

- *Lý thuyết tâm trí (Theory of Mind)*:

Con người có những suy nghĩ và cảm xúc, ký ức hoặc các mô hình não khác điều khiển và ảnh hưởng đến hành vi của họ.

Dựa trên tâm lý này, các nhà nghiên cứu lý thuyết về tâm trí hy vọng phát triển các máy tính có khả năng bắt chước các mô hình tinh thần của con người. Máy móc có thể hiểu rằng con người và động vật có những suy nghĩ và cảm xúc có thể ảnh hưởng đến hành vi của chính chúng.

Lý thuyết về máy móc tâm trí sẽ được yêu cầu sử dụng thông tin thu được từ con người và học hỏi từ nó, sau đó sẽ thông báo bằng cách máy móc giao tiếp hoặc phản ứng với một tình huống khác.

- *Tự nhân thức (Self-awareness)*:

Công nghệ AI này có khả năng tự nhận thức về bản thân, có ý thức và hành xử như con người. Thậm chí, chúng còn có thể bộc lộ cảm xúc cũng như hiểu được những cảm xúc của con người. Đây được xem là bước phát triển cao nhất của công nghệ AI và đến thời điểm hiện tại, công nghệ này vẫn chưa khả thi.

1.1.3. Mặt tích cực và hạn chế của trí tuệ nhân tạo

Trí tuệ nhân tạo (AI) mang lại nhiều lợi ích và tiềm năng cách mạng hóa nhiều lĩnh vực khác nhau. Tuy nhiên, cũng có những tích cực và hạn chế cần xem xét:

*Tích cực:

- Tăng năng suất và hiệu suất công việc: AI có thể thực hiện nhiều nhiệm vụ một cách nhanh chóng và chính xác, giúp tăng năng suất và hiệu suất làm việc trong các ngành công nghiệp.

- Dự đoán và ứng dụng trong quản lý tài chính: AI có khả năng phân tích dữ liệu tài chính phức tạp và dự đoán xu hướng thị trường, giúp đưa ra quyết định tài chính hiệu quả.

- Quản lý y tế và chẩn đoán bệnh: Trong lĩnh vực y tế, AI có thể hỗ trợ các bác sĩ trong việc chẩn đoán bệnh và lập kế hoạch điều trị dựa trên phân tích hình ảnh và dữ liệu bệnh lý.

- Ô tô tự hành: AI có tiềm năng thúc đẩy phát triển xe tự hành, giảm tai nạn giao thông và giúp người già hoặc khuyết tật di chuyển dễ dàng hơn.

- Dự đoán thời tiết và khí hậu: AI có khả năng xử lý dữ liệu khí hậu lớn và dự đoán thời tiết một cách chính xác, giúp cảnh báo thiên tai và khắc phục hậu quả.

- Hỗ trợ trong giáo dục: AI có thể cung cấp các công cụ học tập thông minh, cá nhân hóa giáo dục và tạo điều kiện học tập tốt hơn cho học sinh

*Hạn chế:

- Sự lo ngại về đạo đức và quyền riêng tư: Sử dụng AI có thể dẫn đến các vấn đề về quyền riêng tư và đạo đức, bao gồm việc thu thập và sử dụng dữ liệu cá nhân một cách không đúng mục đích.

- Thất nghiệp và thay thế công việc: Các hệ thống tự động hóa dự kiến s thay thế một số công việc, gây ra lo ngại về thất nghiệp và sự bất ổn kinh tế.

- Giới hạn trong việc hiểu và giải quyết bài toán phức tạp: AI hiện tại vẫn gặp khó khăn trong việc hiểu và giải quyết bài toán phức tạp mà con người có thể làm.

- Nguy cơ trục trặc và lỗi hệ thống: AI có thể gặp trục trặc và lỗi hệ thống, đặc biệt khi dựa vào dữ liệu không chính xác hoặc không đủ lớn.

- Khả năng phân biệt đạo đức và quyết định etic: AI hiện tại không có khả năng phân biệt đạo đức và quyết định theo tiêu chuẩn etic, gây ra các vấn đề đạo đức liên quan đến quyết định của chúng.

- Phụ thuộc vào dữ liệu lớn: AI cần dữ liệu lớn để hoạt động hiệu quả, và điều này có thể tạo ra vấn đề về riêng tư và bảo mật dữ liệu.

1.2. Tên đề tài

Ứng dụng thuật toán A* trong xây dựng game Sokoban

1.3. Lý do chọn đề tài

Ngày nay các kỹ thuật về mạng nơron đã được nghiên cứu và phát triển rất rộng rãi. Ứng dụng của HEURISTIC vào các vấn đề tìm thông tin mù là rất tốt . Trong những năm gần đây HEURISTIC hay cụ thể hơn là A* được áp dụng rộng rãi vào làm game

Thị trường game: Sự phát triển của quy mô thị trường và hạ tầng viễn thông internet, nhu cầu tìm kiếm hình thức giải trí thông qua game nở rộ. Những trò chơi dễ hiểu nhưng mang lại tính thách thức cao ngày càng chiếm được sự đón nhận từ đại chúng.

Tăng cường sự vận dụng của trí não: Game Sokoban ra đời với mục đích giúp người chơi vừa giải trí vừa kích thích sự phát triển của não bộ. Điều này giúp cho người chơi rèn luyện được khả năng quan sát, phán đoán tình huống để não bộ nhanh nhạy và luôn ở trạng thái rèn luyện dù trong lúc giải trí. Việc này giúp người chơi cải thiện hiệu suất não bộ.

Phát triển thêm một số tính năng có thể có: gợi ý không còn khả năng thắng hay thu thập các vật phẩm trong các phần ẩn để hiện ra gợi ý các nước đi tiếp theo cho người chơi qua màn.

1.4. Tổng quan về đề tài

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

1.5. Mục tiêu của đề tài

Xây dựng mô hình trò chơi Sokoban và áp dụng thuật toán A* để tìm giải pháp cho trò chơi

Hiển thị kết quả phân tích dựa trên dữ liệu truyền vào

1.6. Phương pháp nghiên cứu

- Sử dụng bộ dữ liệu thu thập từ nhiều nguồn trên Internet.

- Sử dụng kiến thức đã có để viết dữ liệu của game

- Sử dụng kiến thức đã nghiên cứu được để tiến hành viết chương trình, báo cáo thí nghiệm/thực nghiệm.

1.7. Đối tượng nghiên cứu

Thuật toán A* áp dụng cho bài toán phân loại.

1.8. Phạm vi nghiên cứu

Dữ liệu là các loại game được cho phép lưu hành tại Việt Nam.

CHƯƠNG 2. THUẬT TOÁN A* ÁP DỤNG TRONG GAME SOKOBAN

2.1. Khái niệm

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.

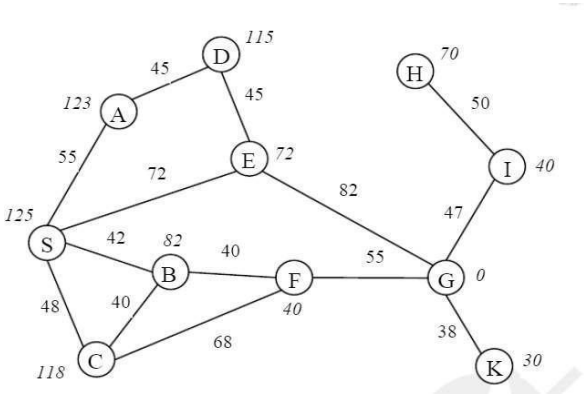
Từ trạng thái hiện tại A* xây dựng tất cả các đường đi có thể đi dùng hàm ước lượng khoảng cách (hàm Heuristic) để đánh giá đường đi tốt nhất có thể đi.

Tùy theo mỗi dạng bài khác nhau mà hàm Heuristic sẽ được đánh giá khác nhau. A* luôn tìm được đường đi ngắn nhất nếu tồn tại đường đi như thế.

2.2. Mô tả thuật toán

- Bắt đầu với trạng thái ban đầu và đặt f(x) = g(x) + h(x).
- Thêm trạng thái ban đầu vào danh sách mở (Open List).
- Lặp lại các bước sau cho đến khi tìm được trạng thái kết thúc hoặc không còn trạng thái nào trong danh sách mở:
- Chọn trạng thái có f(x) nhỏ nhất từ danh sách mở.
- Kiểm tra xem trạng thái này có phải là trạng thái kết thúc không. Nếu có, kết thúc thuật toán và xây dựng đường đi.
- Không thì, đánh dấu trạng thái hiện tại đã được xem xét bằng cách chuyển nó từ danh sách mở sang danh sách đóng.
- Tạo các trạng thái con bằng cách thực hiện các bước di chuyển có thể từ trạng thái hiện tại.
- Đối với mỗi trạng thái con:
    + Nếu trạng thái con đã có trong danh sách mở với chi phí thấp hơn, cập nhật thông tin.
    + Nếu trạng thái con không có trong danh sách mở và danh sách đóng, thêm nó vào danh sách mở.

*Ví dụ: Tìm đường đi từ S đến G trong hình vẽ sa



Hình 2. Ví dụ thuật toán A*

Nút được mở rộng	Tập biên O
	S (125)
S	A_S (178), B_S (124), C_S (166), E_S(144)
B_S	A_S (178), C_S (166), E_S (144), C_B(200), F_B (122)
F_B	A_S (178), C_S (166), E_S (144), C_B(200), G_F (137), C_F (156)
G_F	Đích

Vậy đường đi ngắn nhất tìm được là: G ← F ← B ← S

2.3. Áp dụng vào đề tài

- *Biểu diễn bản đồ Sokoban*: Bản đồ Sokoban thường được biểu diễn dưới dạng một lưới ô vuông, trong đó mỗi ô có thể chứa người chơi, hộp, vị trí đích hoặc các tường. Điều này tạo ra một đồ thị, với mỗi nút trong đồ thị biểu diễn một trạng thái của trò chơi.

- *Khởi tạo A*:* Bắt đầu với trạng thái ban đầu của trò chơi và mục tiêu là đẩy tất cả các hộp vào vị trí đích. Sử dụng thuật toán A* để tìm kiếm đường đi từ trạng thái ban đầu đến một trạng thái mục tiêu.

- *Hàm Heuristic*: Để sử dụng A*, bạn cần xác định một hàm heuristic (h(n)) để ước tính chi phí từ trạng thái hiện tại đến trạng thái mục tiêu. Một hàm heuristic thông thường trong Sokoban là tổng khoảng cách Manhattan giữa các hộp và vị trí đích của chúng.

- *Cập nhật hàng đợi ưu tiên:* A* sẽ duyệt qua các trạng thái kề và xác định trạng thái tiếp theo để duyệt dựa trên giá trị của hàm f(n) (f(n) = g(n) + h(n)). Trạng thái g(n) là chi phí thực tế từ trạng thái ban đầu đến trạng thái hiện tại và h(n) là hàm heuristic. A* sử dụng một hàng đợi ưu tiên (Priority Queue) để duyệt các trạng thái theo thứ tự tăng dần của f(n).

- *Duyệt qua các trạng thái:* A* duyệt qua các trạng thái kề và cập nhật giá trị g(n) và f(n) cho các trạng thái này. Nếu tìm thấy trạng thái mục tiêu (tất cả hộp đều đã được đẩy vào vị trí đích), thuật toán kết thúc và trả về đường đi tìm thấy.

- *Truy vết đường đi*: Sau khi tìm thấy trạng thái mục tiêu, bạn có thể sử dụng thông tin về các trạng thái cha để tái tạo đường đi từ trạng thái ban đầu đến trạng thái mục tiêu.

CHƯƠNG 3. XÂY DỰNG VÀ TRIỂN KHAI TRÒ CHƠI

3.1. Xây dựng trò chơi

3.1.1. Xây dựng các hàm

- **Lớp lưu trạng thái cho mỗi bước:**

```
class state:
    def __init__(self, board, state_parent, list_check_point):
        '''storage current board and state parent of this state'''
        self.board = board
        self.state_parent = state_parent
        self.cost = 1
        self.heuristic = 0
        self.check_points = deepcopy(list_check_point)
      ''' HÀM ĐỆ QUY ĐỂ QUAY LẠI VỀ ĐẦU TIÊN NẾU TRẠNG
      THÁI HIỆN TẠI LÀ MỤC TIÊU '''
      '''sử dụng vòng lặp để tìm bảng danh sách từ đầu đến chỉ số này'''
    def get_line(self):
        if self.state_parent is None:
            return [self.board]
        return (self.state_parent).get_line() + [self.board]
''' TÍNH TOÁN HÀM HEURISTIC ĐƯỢC SỬ DỤNG CHO THUẬT TOÁN A* '''
    def compute_heuristic(self):
        list_boxes = find_boxes_position(self.board)
        if self.heuristic == 0:
            self.heuristic = self.cost + abs(sum(list_boxes[i][0] + list_boxes[i][1]
- self.check_points[i][0] -
          self.check_points[i][1] for i in range(len(list_boxes))))
        return self.heuristic
      ''' HOẠT ĐỘNG QUÁ TẢI RẰNG CHO PHÉP CÁC BANG ĐƯỢC LƯU
      TRỮ TRONG HÀNG ƯU TIÊN '''
    def __gt__(self, other):
```

```
        if self.compute_heuristic() > other.compute_heuristic():
            return True
        else:
            return False
    def __lt__(self, other):
        if self.compute_heuristic() < other.compute_heuristic():
            return True
        else:
            return False
```

Hàm này có vai trò trong việc lưu trữ và quản lý trạng thái của trò chơi Sokoban, cũng như cung cấp các phương thức để tính toán giá trị heuristic và so sánh giữa các trạng thái.

- **Hàm kiểm tra chiến thắng:** trả về "true" nếu tất cả các điểm kiểm tra được bao phủ bởi các hộp

```
''' KIỂM TRA BAN CÓ MỤC TIÊU HAY KHÔNG '''
'''TRẢ VỀ TRUE NẾU TẤT CẢ CÁC ĐIỂM KIỂM TRA ĐƯỢC BAO PHỦ BỞI CÁC HỘP'''
def check_win(board, list_check_point):
    for p in list_check_point:
        if board[p[0]][p[1]] != '$':
            return False
    return True
```

Hàm check_win được sử dụng để kiểm tra xem trạng thái hiện tại của bảng trong trò chơi Sokoban có đạt được mục tiêu (chiến thắng) hay không. Điều này được thực hiện bằng cách kiểm tra xem tất cả các điểm kiểm tra trên bảng có được bao phủ bởi các hộp hay không. Hàm này được coi là một phần quan trọng trong logic kiểm tra chiến thắng của trò chơi Sokoban.

- **Hàm gán ma trận:** lặp lại từng phần tử và trả về ma trận

```
''' GÁN MA TRẬN '''
def assign_matrix(board):
```

'''BẢNG TRỞ LẠI GIỐNG NHƯ BẢNG ĐẦU VÀO'''

'''FOR LẦN LƯỢT TỪNG CÁI VÀ NHẢ RA MA TRẬN'''

return [[board[x][y] for y in range(len(board[0]))] for x in range(len(board))]

Hàm assign_matrix được thiết kế để tạo ra một bản sao của ma trận đầu vào (board). Nó thực hiện điều này bằng cách duyệt qua từng dòng và cột của ma trận đầu vào và tạo ra một ma trận mới với giá trị giống với ma trận đầu vào. Hàm này là một công cụ để tạo một bản sao của ma trận đầu vào trong trò chơi Sokoban, giúp duy trì tính không thay đổi của trạng thái ban đầu khi cần thiết.

- **Hàm tìm vị trí hiện tại của người chơi** trả lại vị trí của người chơi trong bảng, nếu là "@" thì trả về tọa độ

''' TÌM VỊ TRÍ HIỆN TẠI CỦA NGƯỜI CHƠI TRONG MỘT BAN'''

```
def find_position_player(board):
    '''trả lại vị trí của người chơi trong bảng'''
    for x in range(len(board)):
        for y in range(len(board[0])):
            '''nếu là @ thì trả về tọa độ còn k thì'''
            if board[x][y] == '@':
                return (x,y)
    return (-1,-1)  # error board
```

Hàm giúp xác định vị trí của người chơi trên bảng và trả về tọa độ tương ứng.

- **Hàm so sánh 2 bảng:** trả về "true" nếu bảng A giống như bảng B def compare_matrix(board_A, board_B):

''' SO SÁNH 2 BAN '''

```
def compare_matrix(board_A, board_B):
```

'''TRẢ VỀ TRUE NẾU BẢNG A GIỐNG NHƯ BẢNG B'''

```
    if len(board_A) != len(board_B) or len(board_A[0]) != len(board_B[0]):
        return False
    for i in range(len(board_A)):
```

```
            for j in range(len(board_A[0])):
                if board_A[i][j] != board_B[i][j]:
                    return False
        return True
```

Hàm compare_matrix được sử dụng để so sánh hai ma trận (board_A và board_B) trong trò chơi Sokoban và trả về True nếu chúng giống nhau, ngược lại trả về False.

- **Hàm kiểm tra bảng có tồn tại trong danh sách chuyển đổi không**: trả về "true" nếu có cùng một bảng trong danh sách:

```
def is_board_exist(board, list_state):
    '''TRẢ VỀ TRUE NẾU CÓ CÙNG MỘT BẢNG TRONG DANH SÁCH'''
    for state in list_state:
        if compare_matrix(state.board, board): # kiểm tra ma trận tồn tại hay chưa
            return True
    return False
```

Hàm is_board_exist được thiết kế để kiểm tra xem một bảng trò chơi cụ thể (board) đã tồn tại trong danh sách các trạng thái (list_state) hay chưa. Nếu có trạng thái nào đó có bảng giống với bảng đầu vào, hàm trả về True, biểu thị rằng bảng đã tồn tại trong danh sách. Nếu không tìm thấy bảng nào giống với bảng đầu vào trong danh sách, hàm trả về False, biểu thị rằng bảng chưa tồn tại trong danh sách.

- **Hàm kiểm tra có hộp nào bị dính vào góc không:** nếu thỏa mãn một trong các điều kiện thì sẽ không di chuyển được nữa và thất bại

```
def check_in_corner(board, x, y, list_check_point):
    '''TRẢ VỀ TRUE NẾU BOARD[X][Y] Ở GÓC KIỂM TRA XEM HỘP KÝ HIỆU CÓ BỊ KÍCH VÀO GÓC HAY KHÔNG'''
    # ktra xem hộp của mình có nằm trong góc hay k
    if board[x-1][y-1] == '#':
        if board[x-1][y] == '#' and board[x][y-1] == '#': # nếu là dấu #- tường
    # nếu mình thấy thỏa mãn một trong các điều kiện thì mình sẽ k di chuyển nnuwa-> thất bại
```

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

Hàm check_in_corner được thiết kế để kiểm tra xem ô có tọa độ (x, y) trên bảng trò chơi có nằm trong góc và có hộp bị kẹt ở góc hay không.

- **Hàm tìm vị trí tất cả các hộp:**

```
    ''' TÌM VỊ TRÍ TẤT CẢ CÁC HỘP '''
    def find_boxes_position(board):
        result = []
        for i in range(len(board)):
            for j in range(len(board[0])):
                if board[i][j] == '$':
                    result.append((i,j)) # ktra vị trí đã đúng hay chưa
        return result
```

Hàm find_boxes_position giúp tìm và trả về tất cả các vị trí của hộp trên bảng trong trò chơi Sokoban. Nếu có hộp, thêm tọa độ (i, j) của hộp vào danh sách kết quả

- **Hàm kiểm tra xem có thể dịch chuyển 1 hộp trong ít nhất 1 hướng:**

```
    ''' KIỂM TRA XEM CÓ THỂ CHUYỂN ĐỔI MỘT HỘP TRONG ÍT NHẤT 1 HƯỚNG'''
def is_box_can_be_moved(board, box_position):
    # ktra xem hộp có di chuyển hay k
    left_move = (box_position[0], box_position[1] - 1)      # qua trái
    right_move = (box_position[0], box_position[1] + 1)     # qua phải
    up_move = (box_position[0] - 1, box_position[1])        # lên trên
    down_move = (box_position[0] + 1, box_position[1])      # xuống dưới
 if(board[left_move[0]][left_move[1]]=='' or
board[left_move[0]][left_move[1]] == '%' or
board[left_move[0]][left_move[1]] == '@') and
board[right_move[0]][right_move[1]] != '#' and
board[right_move[0]][right_move[1]] != '$':
        return True
    if (board[right_move[0]][right_move[1]] == ' ' or
board[right_move[0]][right_move[1]] == '%' or
board[right_move[0]][right_move[1]] == '@') and
board[left_move[0]][left_move[1]] != '#' and
board[left_move[0]][left_move[1]] != '$':
        return True
    if (board[up_move[0]][up_move[1]] == ' ' or
board[up_move[0]][up_move[1]] == '%' or
board[up_move[0]][up_move[1]] == '@') and
board[down_move[0]][down_move[1]] != '#' and
board[down_move[0]][down_move[1]] != '$':
        return True
    if (board[down_move[0]][down_move[1]] == ' ' or
board[down_move[0]][down_move[1]] == '%' or
board[down_move[0]][down_move[1]] == '@') and
board[up_move[0]][up_move[1]] != '#' and
board[up_move[0]][up_move[1]] != '$':
```

```
            return True
        return False # nếu k di chuyển đc nữa thì sẽ false
```

Hàm is_box_can_be_moved được sử dụng để kiểm tra xem một hộp tại vị trí cụ thể có thể di chuyển được ít nhất một bước theo một trong các hướng (trái, phải, lên trên, xuống dưới) hay không.

- **Hàm kiểm tra các hộp đang bị mắc:**

```
    ''' KIỂM TRA DÒNG TẤT CẢ CÁC HỘP ĐANG BỊ MẮC '''
    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): # list ktra các dấu dona có trùng với dấu % hay chưa
                return False
            if is_box_can_be_moved(board, box_position): # box chưa di chuyển đc thì ...nếu chưa phải thì di chuyển tiếp
                result = False
        return result
```

Hàm này là một phần của logic kiểm tra xem tất cả các hộp trên bảng có đang bị mắc hay không và xem chúng có đặt đúng lên các điểm kiểm tra hay không trong trò chơi Sokoban.

- **Hàm kiểm tra xem có ít nhất một hộp có bị khóc ở góc không:** nếu hộp đẩy vào góc tường thì k đi đc nữa và thất bại

```
    ''' KIỂM TRA XEM CÓ ÍT NHẤT MỘT HỘP CÓ BỊ KHÓC Ở GÓC KHÔNG'''
    def is_board_can_not_win(board, list_check_point):
        '''trả về true nếu hộp ở góc tường -> không thể thắng'''
        # nếu hộp đẩy vào góc tường thì k đi đc nữa-> false
        for x in range(len(board)):
            for y in range(len(board[0])):
                if board[x][y] == '$':
```

```
                if check_in_corner(board, x, y, list_check_point):
                    return True
        return False
```

Hàm is_board_can_not_win được thiết kế để kiểm tra xem có ít nhất một hộp trên bảng bị kẹt ở góc của tường hay không, khiến cho trò chơi không thể chiến thắng.

- **Hàm lấy ra các vị trí tiếp theo có thể di chuyển từ vị trí hiện tại:**

```
    ''' NHẬN DI CHUYỂN CÓ THỂ TIẾP THEO '''
    '''TRẢ VỀ DANH SÁCH CÁC VỊ TRÍ MÀ NGƯỜI CHƠI CÓ THỂ  DI CHUYỂN ĐẾN TỪ VỊ TRÍ HIỆN TẠI'''
    def get_next_pos(board, cur_pos):
        x,y = cur_pos[0], cur_pos[1]
        list_can_move = []
        # MOVE UP (x - 1, y)
        if 0 <= x - 1 < len(board):     # 0<= x-1 < chiều dai ma trận
            value = board[x - 1][y]
            if value == ' ' or value == '%':
                list_can_move.append((x - 1, y))    # append: thêm phần tu vào cuối một list
            elif value == '$' and 0 <= x - 2 < len(board):
                next_pos_box = board[x - 2][y]
                if next_pos_box != '#' and next_pos_box != '$':
                    list_can_move.append((x - 1, y))
        # MOVE DOWN (x + 1, y)
        if 0 <= x + 1 < len(board):
            value = board[x + 1][y]
            if value == ' ' or value == '%':
                list_can_move.append((x + 1, y))
            elif value == '$' and 0 <= x + 2 < len(board):
                next_pos_box = board[x + 2][y]
```

```
            if next_pos_box != '#' and next_pos_box != '$':
                list_can_move.append((x + 1, y))
        # MOVE LEFT (x, y - 1)
        if 0 <= y - 1 < len(board[0]):
            value = board[x][y - 1]
            if value == ' ' or value == '%':
                list_can_move.append((x, y - 1))
            elif value == '$' and 0 <= y - 2 < len(board[0]):
                next_pos_box = board[x][y - 2]
                if next_pos_box != '#' and next_pos_box != '$':
                    list_can_move.append((x, y - 1))
        # MOVE RIGHT (x, y + 1)
        if 0 <= y + 1 < len(board[0]):
            value = board[x][y + 1]
            if value == ' ' or value == '%':
                list_can_move.append((x, y + 1))
            elif value == '$' and 0 <= y + 2 < len(board[0]):
                next_pos_box = board[x][y + 2]
                if next_pos_box != '#' and next_pos_box != '$':
                    list_can_move.append((x, y + 1))
        return list_can_move
```

Hàm get_next_pos được thiết kế để trả về danh sách các vị trí mà người chơi có thể di chuyển đến từ vị trí hiện tại trên bảng trong trò chơi Sokoban.

- **Hàm tìm tất cả các điểm kiểm tra trên bảng:**

```
    '''  TÌM TẤT CẢ CÁC ĐIỂM KIỂM TRA TRÊN BẢNG'''
    def find_list_check_point(board):
        '''danh sách trả lại điểm kiểm tra từ bảng
            nếu không có bất kỳ điểm kiểm tra nào, hãy trả về danh sách trống
            nó sẽ kiểm tra số hộp, nếu số hộp < số điểm kiểm tra
                danh sách trả về [(-1,-1)]'''
```

```
list_check_point = []
num_of_box = 0
''' KIỂM TRA TOÀN BỘ BẢNG ĐỂ TÌM ĐIỂM KIỂM TRA VÀ SỐ HỘP'''
for x in range(len(board)):
    for y in range(len(board[0])):
        if board[x][y] == '$':
            num_of_box += 1
        elif board[x][y] == '%':
            list_check_point.append((x,y)) # append vị trí
''' KIỂM TRA NẾU SỐ HỘP < SỐ ĐIỂM KIỂM TRA'''
if num_of_box < len(list_check_point): # ktra có đủ thùng hay k
    return [(-1,-1)]
return list_check_point
```

Danh sách trả lại điểm kiểm tra từ bảng, nếu không có bất kỳ điểm kiểm tra nào sẽ trả về danh sách trống. Kiểm tra số hộp, nếu số hộp < số điểm kiểm tra danh sách trả về [(-1,-1)].

3.1.2. Xây dựng thuật toán A*

Như đã trình bày ở chương 2, thuật toán A* đóng vai trò quan trọng trong trò chơi để xác định đường đến đích cho nhân vật. Trong đề tài này, nhóm em sẽ biểu diễn thuật toán A* theo source code như sau:

- Import các thư viện và hàm cần thiết

```
import support_function as spf
import time
from queue import PriorityQueue
```

- Viết hàm để xây dựng thuật toán A*:

```
def AStart_Search(board, list_check_point):
  start_time = time.time()
  if spf.check_win(board, list_check_point):
     print("Found win")
```

```
        return [board]
    start_state = spf.state(board, None, list_check_point)
    list_state = [start_state]
    heuristic_queue = PriorityQueue() # tạo hàng đơi
    heuristic_queue.put(start_state)# tiến hành đưa từng đưa từng trường hợp vào hàng đợi để xét
    while not heuristic_queue.empty():
        now_state = heuristic_queue.get()
        cur_pos = spf.find_position_player(now_state.board)
        list_can_move = spf.get_next_pos(now_state.board, cur_pos)
        for next_pos in list_can_move:
            new_board = spf.move(now_state.board, next_pos, cur_pos, list_check_point)
            # kiểm tra các điều kiện
            if spf.is_board_exist(new_board, list_state): #
                continue
            if spf.is_board_can_not_win(new_board, list_check_point):
                continue # ktra xem nước đi có dẫn tới chiến thắng hay k
            if spf.is_all_boxes_stuck(new_board, list_check_point):
                continue # ktra hôp có bị kẹt hay k
            new_state = spf.state(new_board, now_state, list_check_point)
            if spf.check_win(new_board, list_check_point):
                print("Found win")
                return (new_state.get_line(), len(list_state))
            list_state.append(new_state)
            heuristic_queue.put(new_state) # hàng đợi duyệt qua các trường hợp để tìm ra cách theo nó là chiến thắng
            end_time = time.time()
            if end_time - start_time > spf.TIME_OUT:
                return []
```

```
            end_time = time.time()
            if end_time - start_time > spf.TIME_OUT:
                return []
        print("Not Found")
        return []
```

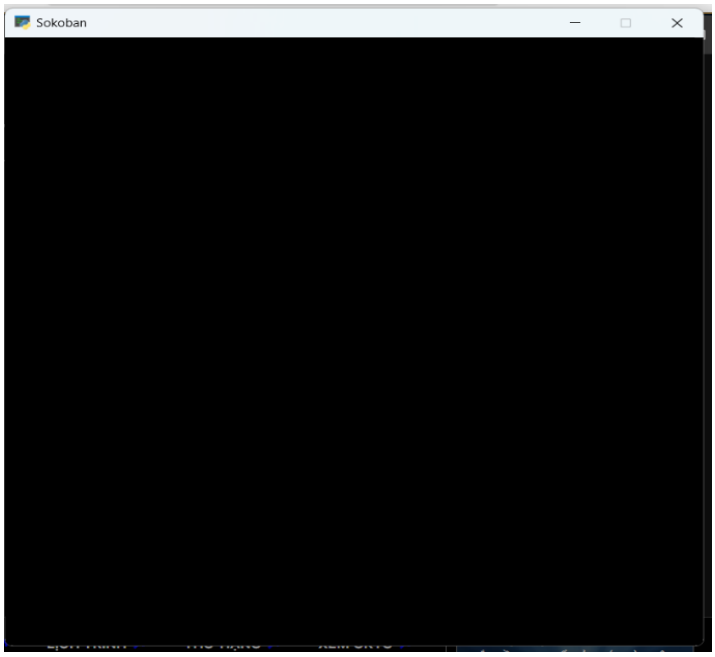
3.1.3. Xây dựng giao diện

- Tạo giao diện cho game sử dụng thư viện pygame

```
import pygame, sys, random
from pygame.locals import *
```

- Khởi tạo Pygame và thiết lập màn hình trò chơi:

```
pygame.init()
pygame.font.init()
screen = pygame.display.set_mode((640, 640))
pygame.display.set_caption('Sokoban')
clock = pygame.time.Clock()
```

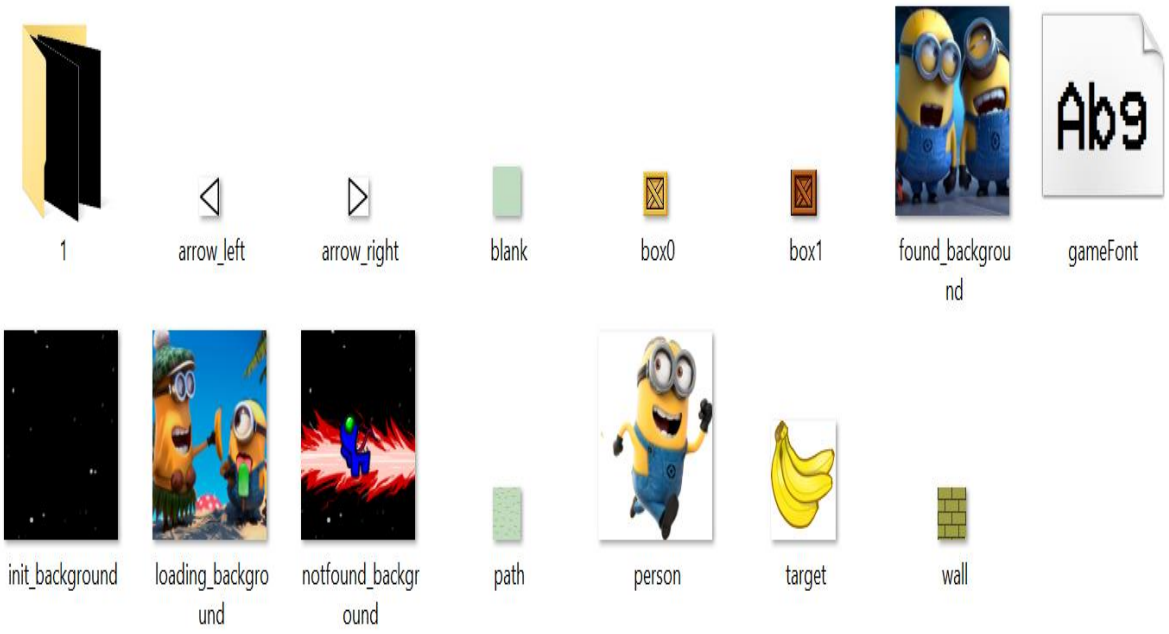


Hình 3: Tạo cửa sổ với kích thước 640x640

- Tải các hình ảnh và tạo biến cho các tài nguyên trò chơi:

```
assets_path = os.getcwd() + "\\..\\Assets"
```

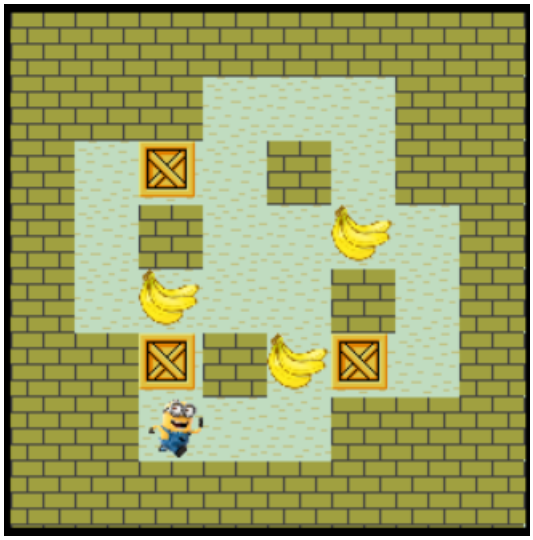
```
os.chdir(assets_path)
person = pygame.image.load(os.getcwd() + '\\person.png')
person = pygame.transform.scale(person,(33, 30))
wall = pygame.image.load(os.getcwd() + '\\wall.png')
box0 = pygame.image.load(os.getcwd() + '\\box0.png')
box1 = pygame.image.load(os.getcwd() + '\\box1.png')
target = pygame.image.load(os.getcwd() + '\\target.png')
target = pygame.transform.scale(target,(30,30))
path = pygame.image.load(os.getcwd() + '\\path.png')
arrow_left = pygame.image.load(os.getcwd() + '\\arrow_left.png')
arrow_right = pygame.image.load(os.getcwd() + '\\arrow_right.png')
init_background= pygame.image.load(os.getcwd() +
'\\init_background.png')
loading_background = pygame.image.load(os.getcwd() +
'\\loading_background.png')
notfound_background = pygame.image.load(os.getcwd() +
'\\notfound_background.png')
found_background = pygame.image.load(os.getcwd() +
'\\found_background.png')
```



Hình 4: File hình ảnh nhân vật và các đối tượng trong trò chơi

- Định nghĩa hàm renderMap(board) để vẽ trạng thái trò chơi lên màn hình:

```
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(path, (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(box0, (j * 32 + indent, i * 32 + 250))
            if board[i][j] == '%':
                screen.blit(target, (j * 32 + indent, i * 32 + 250))
            if board[i][j] == '@':
                screen.blit(person, (j * 32 + indent, i * 32 + 250))
```



Hình 5: Bản đồ cho trận đấu

- Sử dụng các hàm và biến Pygame trong vòng lặp chính của trò chơi:

```
while running:
        screen.blit(init_background, (0, 0))
        if sceneState == "init":
            initGame(maps[mapNumber])
        if sceneState == "executing":
            list_check_point = check_points[mapNumber]
            # Chọn giữa người dùng chơi và máy chơi
            if algorithm == "Player":
                print("Player")
                list_board = maps[mapNumber]
            else:
                print("AStar")
            list_board = astar.AStart_Search(maps[mapNumber], list_check_point)
            if len(list_board) > 0:
                sceneState = "playing"
                stateLenght = len(list_board[0])
                currentState = 0
            else:
```

```
            sceneState = "end"
            found = False
    if sceneState == "loading":
        loadingGame()
        sceneState = "executing"
    if sceneState == "end":
        if algorithm == "Player":
            foundGame(list_board)
        else:
            foundGame(list_board[0][stateLenght - 1])
    if sceneState == "playing":
        clock.tick(4)
        if(algorithm == "Player"):
         new_list_board = player.Player(list_board, list_check_point, pygame)
            list_board = new_list_board
            if np.all(list_board == True):
                sceneState = "end"
                list_board = list_board_win
                found = True
            else:
                renderMap(list_board)
                list_board_win = list_board
        if (algorithm == "AI"):
            renderMap(list_board[0][currentState])
            currentState = currentState + 1
        if currentState == stateLenght:
            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"
                # Press SPACE key board to switch algorithm
                if event.key == pygame.K_SPACE and sceneState == "init":
                    if algorithm == "Player":
                        algorithm = "AI"
                    else:
                        algorithm = "Player"
            if event.key== pygame.K_SPACE and sceneState== "playing":
                    initGame(maps[mapNumber])
                    sceneState = "init"
        pygame.display.flip()
    pygame.quit()
```

- Hàm khởi tạo trò chơi:

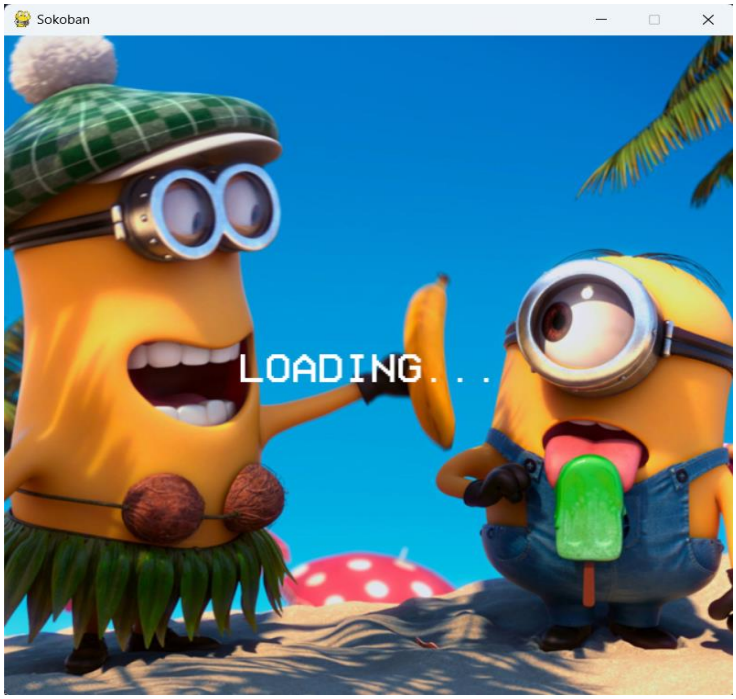
```
def initGame(map):
    titleSize = pygame.font.Font('gameFont.ttf', 60)
    titleText = titleSize.render('MINIONS - TTNT', True, WHITE)
    titleRect = titleText.get_rect(center=(320, 80))
    screen.blit(titleText, titleRect)
```

```
titleLevSize = pygame.font.Font('gameFont.ttf', 20)
titleLevText = titleLevSize.render(str('Chon level: [LEFT] or [RIGHT]'), True, WHITE)
titleLevRect = titleLevText.get_rect(center=(320, 150))  # căn chỉnh A star search
screen.blit(titleLevText, titleLevRect)
mapSize = pygame.font.Font('gameFont.ttf', 30)
mapText = mapSize.render("Lv." + str(mapNumber + 1), True, WHITE)
mapRect = mapText.get_rect(center=(320, 200))
screen.blit(mapText, mapRect)
screen.blit(arrow_left, (246, 188))
screen.blit(arrow_right, (370, 188))
titleFunSize = pygame.font.Font('gameFont.ttf', 20)
titleFunText = titleFunSize.render(str('Chon che do choi: [Space]'), True, WHITE)
titleFunRect = titleFunText.get_rect(center=(320, 550))  # căn chỉnh A star search
screen.blit(titleFunText, titleFunRect)
algorithmSize = pygame.font.Font('gameFont.ttf', 30)
algorithmText = algorithmSize.render(str(algorithm), True, WHITE)
algorithmRect = algorithmText.get_rect(center=(320, 600)) # căn chỉnh A star search
screen.blit(algorithmText, algorithmRect)
screen.blit(arrow_left, (230, 590))
screen.blit(arrow_right, (380, 590))
renderMap(map)
```

Hình 6: Giao diện chính trò chơi

- Hàm tạo giao diện chờ:

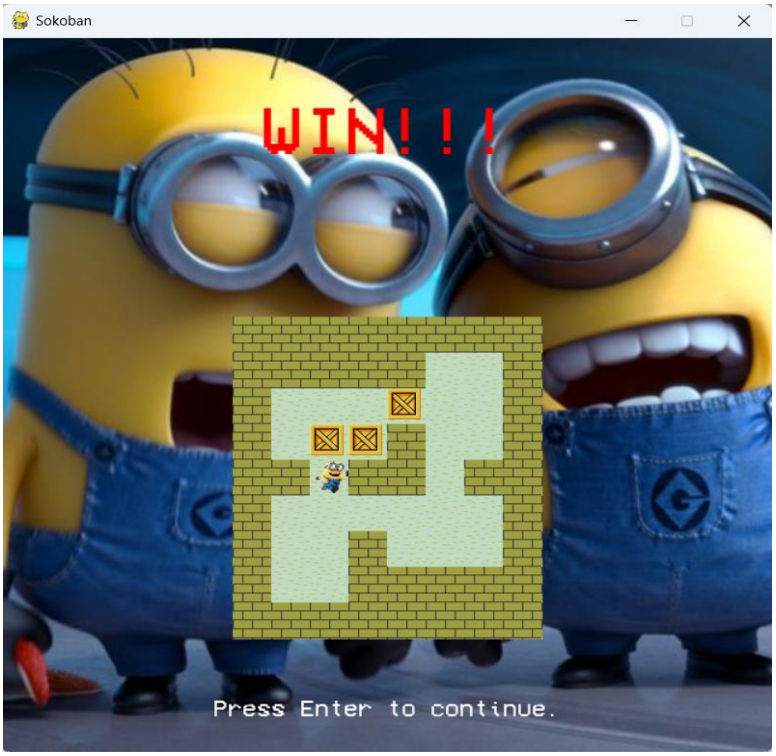
```
def loadingGame():
    screen.blit(loading_background, (0, 0))
    fontLoading_1 = pygame.font.Font('gameFont.ttf', 40)
    text_1 = fontLoading_1.render('LOADING...', True, WHITE)
    text_rect_1 = text_1.get_rect(center=(320, 320))
    screen.blit(text_1, text_rect_1)
```



Hình 7: Giao diện chờ

- Hàm tạo giao diện kết thúc trò chơi

```
def foundGame(map):
    screen.blit(found_background, (0, 0))
    titleWSize = pygame.font.Font('gameFont.ttf', 60)
    titleWText = titleWSize.render('WIN!!!', True, RED)
    titleWRect = titleWText.get_rect(center=(320, 80))
    screen.blit(titleWText, titleWRect)
    font_2 = pygame.font.Font('gameFont.ttf', 20) # chỉnh phông chữ text_2
    text_2 = font_2.render('Press Enter to continue.', True, WHITE)
    text_rect_2 = text_2.get_rect(center=(320, 600)) # căn giữa
    screen.blit(text_2, text_rect_2)
    renderMap(map)
```



Hình 8: Giao diện kết thúc trận đấu

3.2. Các chức năng chính

a) Màn hình chính:

- Chọn chế độ chơi:

Game có 2 chế độ chơi là: Player, AI. Người chơi có thể đổi chế độ qua lại bằng cách ấn [SPACE] trên bàn phím.

- Chọn level:

Người chơi có thể tùy chọn level bằng cách ấn mũi tên trái (<) hoặc phải (>) trên bàn phím

b) Màn hình chờ

Sau khi đã chọn xong chế độ và level chơi, người chơi sẽ được dẫn đến màn hình chờ chuẩn bị vào trận

c) Màn hình trận đấu

Sau khi đợi khoảng 1s, trận đấu sẽ hiện ra và người chơi có thể thao tác tùy theo chế độ chơi.

- Chế độ Player: Đây là chế độ để người chơi có thể trải nghiệm trò chơi 1 cách trực tiếp. Tại đây người chơi sẽ điều khiển các nút trên bàn phím: lên, xuống, trái, phải để di chuyển hình người minions trong game. Mỗi lần di chuyển sẽ được

tịnh tiến 1 ô sao cho các vị trí được tịnh tiến lên không bị đâm vào tường. Mục tiêu cuối cùng là di chuyển để đẩy các hộp vào các vị trí đích đã có sẵn.

- Chế độ AI (máy): Ở chế độ này, người chơi chỉ cần đợi AI giải xong ván đấu mà không cần trực tiếp tham gia trò chơi. Đây cũng là cách để người chơi có thể tham khảo cách giải với sự hướng dẫn của AI

d) Màn hình kết thúc trò chơi

Sau khi đã giải xong level, trò chơi sẽ lập tức thông báo rằng bạn đã chiến thắng và có thể ấn “Enter” để quay về màn hình chính.

KẾT LUẬN

1. Đánh giá

Dựa vào các kiến thức về không gian trạng thái và đặc biệt là thuật giải A* đã được học trong môn học trí tuệ nhân tạo và dưới sự trợ giúp nhiệt tình của cô giáo, chúng em đã áp dụng được thuật toán này vào trong một ứng dụng thực tế cụ thể là trò chơi Sokoban.

Trong quá trình nghiên cứu về thuật toán A* và ứng dụng vào trò chơi Sokoban, chúng ta đã đạt được những kết quả quan trọng. Thuật toán A* đã được áp dụng thành công để giải quyết vấn đề tìm đường đi trong trò chơi Sokoban, nâng cao hiệu suất và chính xác của giải thuật. Nghiên cứu đã đánh giá những đóng góp mới của đề tài bằng cách cải thiện thuật toán A* để phù hợp với bản chất đặc biệt của trò chơi Sokoban. Các thử nghiệm và so sánh kết quả đã chứng minh tính hiệu quả của phương pháp mới, giúp giải quyết các bài toán Sokoban phức tạp một cách hiệu quả hơn.

Tóm lại: đề tài "Tìm hiểu thuật toán A* và ứng dụng vào trò chơi Sokoban" đã đạt được các mục tiêu đề ra như sau:

- Nghiên cứu và phân tích thuật toán A*, một trong những thuật toán tìm kiếm đường đi hiệu quả nhất hiện nay.
- Xây dựng mô hình trò chơi Sokoban và áp dụng thuật toán A* để tìm giải pháp cho trò chơi.
- Đánh giá hiệu quả của thuật toán A* trong việc tìm giải pháp cho trò chơi Sokoban.
- Hiểu rõ hơn về thuật toán A* được áp dụng vào thực tế.

2. Hướng phát triển.

Từ kết quả nghiên cứu, chúng tôi đề xuất tiếp tục nghiên cứu và phát triển thuật toán A* để áp dụng vào các loại trò chơi khác, đặc biệt là những trò chơi có tính chất tương tự như Sokoban. Đồng thời, nên tập trung vào việc tối ưu hóa thuật toán để có thể xử lý các bài toán lớn hơn và phức tạp hơn.

Ngoài ra, cần thiết lập các cơ chế và chính sách hỗ trợ cho việc áp dụng kết quả nghiên cứu vào thực tế, có thể thông qua việc hợp tác với các doanh nghiệp và tổ chức có liên quan.

Tóm lại, đề tài có một số kiến nghị sau:

- Tiếp tục nghiên cứu và phát triển thuật toán A* để cải thiện hiệu năng và khả năng mở rộng.
- Nghiên cứu và phát triển các phương pháp tối ưu hóa việc tìm kiếm giải pháp cho trò chơi Sokoban.
- Nghiên cứu và ứng dụng thuật toán A* vào các trò chơi khác.
- Xây dựng thêm một số chức năng, chẳng hạn như tính năng tính số bước di chuyển của người chơi và so sánh với số bước ngắn nhất từng chơi.
- Ở chế độ player, khi người chơi hoàn thành xong 1 level chương trình sẽ tự động thông báo cho người chơi 2 lựa chọn: chơi lại hoặc chơi tiếp.
- Trường hợp thùng hàng bị kẹt ở góc: kiểm tra xem nếu tất cả các thùng hàng bị kẹt ở góc và không thể di chuyển thì trò chơi kết thúc và hiện thông báo chơi lại hoặc thoát trò chơi.

TÀI LIỆU THAM KHẢO

[1] https://codelearn.io/sharing/lap-trinh-game-co-ban-voi-pygame (Hướng dẫn lập trình với thư viện pygame)

[2] A* Search Algorithm: https://en.wikipedia.org/wiki/A*_search_algorithm

[3] Sokoban: https://en.wikipedia.org/wiki/Sokoban

[4] Tìm hiểu thuật toán A*: https://vi.wikipedia.org/wiki/Thuật_toán_A*

[5] Trò chơi Sokoban: https://vi.wikipedia.org/wiki/Sokoban

[6] https://www.youtube.com/watch?v=7eb0xpzraDo(Ý tưởng thiết kế trò chơi)

[7] Giáo trình Trí tuệ nhân tạo Trường Đại học Công Nghiệp Hà Nội