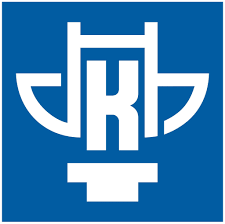
 🙠🙟🕮🙝🙢

TRƯỜNG ĐẠI HỌC KIẾN TRÚC HÀ NỘI

 KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN MÔN HỌC

TRÍ TUỆ NHÂN TẠO

***Đề tài:***

**LẬP TRÌNH TRÒ CHƠI SOKOBAN (SỬ DỤNG TRÍ TUỆ NHÂN TẠO ĐỂ TRÒ CHƠI TỰ HOẠT ĐỘNG)**

Giảng viên hướng dẫn: GV. Nguyễn Quốc Huy

Nhóm sinh viên thực hiện: Nhóm 10

1. Phạm Hải Đức

2. Phan Văn Tươi

3. Lê Đình Triều

4. Lương Sơn Bá

5. Cao Trần Đức Mạnh

**Hà Nội – 2023**

**Mục lục**

[**Lời nói đầu 3**](#_Toc149824784)

[**Nội dung 4**](#_Toc149824785)

[**I. Một số giới thiệu 4**](#_Toc149824786)

[*I.1.Tổng quát về trò chơi Sokoban và đề tài* 4](#_Toc149824787)

[**I.1.1. Giới thiệu trò chơi 4**](#_Toc149824788)

[**I.1.2. Áp Dụng Trí Tuệ Nhân Tạo trong Trò Chơi Sokoban 5**](#_Toc149824789)

[**I.1.3. Ý Nghĩa của Việc Sử Dụng Trí Tuệ Nhân Tạo trong Sokoban 5**](#_Toc149824790)

[I.2. Một số khái niệm : 6](#_Toc149824791)

[**I.2.1. Dạng trò chơi: 6**](#_Toc149824792)

[**I.2.2. Cây trò chơi: 7**](#_Toc149824793)

[I.3. Công nghệ và thuật toán áp dụng : 8](#_Toc149824794)

[**I.3.1. Ngôn ngữ Python 8**](#_Toc149824795)

[**I.3.2. Thuật toán A\* 9**](#_Toc149824796)

[**I.3.3. Thuật toán BFS 10**](#_Toc149824797)

[**I.3.4. Chiến lược tìm kiếm 11**](#_Toc149824798)

[**II. Giao diện ,chức năng, hệ thống trò chơi 13**](#_Toc149824799)

[*II.1.* *Các map trong trò chơi* 13](#_Toc149824800)

[**II.1.1. Tổng quát 13**](#_Toc149824801)

[**II.1.2. Mô hình hóa các map trong trò chơi 13**](#_Toc149824802)

[II.2. Hệ thống trò chơi 15](#_Toc149824803)

[**II.2.1. Các thư viện được áp dụng 15**](#_Toc149824804)

[**II.2.2. Áp dụng thuật toán 20**](#_Toc149824805)

[II.3. Giới thiệu các thư mục code và cách chạy trò chơi 22](#_Toc149824806)

[**II.3.1. Giới thiệu các thư mục code 22**](#_Toc149824807)

[**II.3.2. Cách chạy trò chơi: 23**](#_Toc149824808)

[**III. Kết quả, kiểm thử và đánh giá 25**](#_Toc149824809)

[III.1. Kết quả 25](#_Toc149824810)

[III.2. Kết Luận : 30](#_Toc149824811)

[**Tài liệu tham khảo 30**](#_Toc149824812)

[**phân công công việc 31**](#_Toc149824813)

# **Lời nói đầu**

Trí tuệ nhân tạo là một môn học chuyên ngành bắt buộc đối với sinh viên thuộc chuyên ngành tin học. Môn học này trang bị cho sinh viên một số kiến thức cơ bản nhất về các phơng pháp giải quyết vấn đề và kỹ thuật xử lý tri thức. Đó là một nền tảng vững chắc để đi sâu nghiên cứu các chuyên đề khác. Sự ra đời và phát triển của trí tuệ nhân tạo đã kéo theo hàng loạt công nghệ mới và khoa học kỹ thuật ngày càng phát triển. Các sản phẩm của trí tuệ nhân tạo da số phục vụ cho mục dích tốt và ngày càng da dạng phong phú như : phần mềm phục vụ cho học tập và giải trí v.v...

Từ những năm 50 của thế kỉ XX, trí tuệ nhân tạo mới bắt đầu nhen nhóm, vậy mà giờ đây, ta có thể thấy nó đã có những bước phát triển vượt bậc, không ngừng.

Để hiểu rõ hơn về AI, chúng em đã quyết định lựa chọn trò chơi sokoban, một trò chơi cổ điển nhưng là một trong những trò chơi biểu tượng của trí tuệ để hoàn thành bài tập lớn môn học này. Dưới đây là phần trình bày báo cáo môn học thông qua đề tài đã nêu ở trên. Hiện tại, chương trình của chúng em còn nhiều thiếu sót, rất mong nhận được những ý kiến góp ý và tư vấn từ thầy và các bạn để chúng em có thể trau dồi thêm cho bản thân những kiến thức hữu ích .

***Chúng em xin chân thành cảm ơn !***

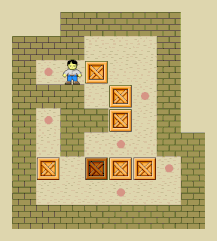
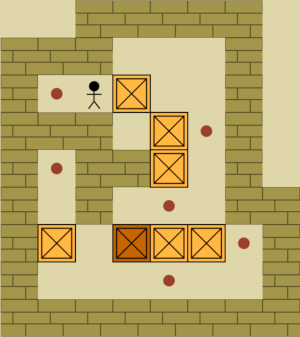
# **Nội dung**

# **I. Một số giới thiệu**

### ***I.1.Tổng quát về trò chơi Sokoban*** ***và đề tài***

#### **I.1.1. Giới thiệu trò chơi**

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

*******(Một số hình ảnh trò chơi)*

* **Cơ chế trò chơi**: Trò chơi Sokoban là một trò chơi logic thú vị trong đó bạn phải di chuyển nhân vật để đẩy các hộp vào các vị trí đích trên một lưới. Dưới đây là cơ chế cơ bản để chơi trò chơi Sokoban:
* *Mục tiêu:* Mục tiêu của trò chơi Sokoban là đẩy tất cả các hộp (được ký hiệu là 'B') vào các ô đích (được ký hiệu là 'G') trên lưới.
* *Thành* phần chính:

1. Nhân vật (Player): Bạn điều khiển một nhân vật trên lưới, thường được ký hiệu bằng 'P'.
2. Hộp (Box): Hộp là đối tượng bạn cần đẩy vào các ô đích. Chúng thường được ký hiệu bằng 'B'.
3. Ô đích (Goal): Ô đích là nơi bạn cần đẩy các hộp tới. Chúng thường được ký hiệu b

#### **I.1.2. Áp Dụng Trí Tuệ Nhân Tạo trong Trò Chơi Sokoban**

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

Ứng dụng Trí Tuệ Nhân Tạo trong Sokoban có nhiều ứng dụng thú vị. Chúng có thể được sử dụng để:

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

#### **I.1.3. Ý Nghĩa của Việc Sử Dụng Trí Tuệ Nhân Tạo trong Sokoban**

Sử dụng Trí Tuệ Nhân Tạo để giải quyết Sokoban không chỉ là về việc tạo ra trò chơi tự chơi thú vị mà còn mang lại những lợi ích thực tiễn trong việc nghiên cứu và phát triển các thuật toán Trí Tuệ Nhân Tạo. Nó giúp thúc đẩy sự hiểu biết về cách máy tính tư duy và ra quyết định trong môi trường có hạn chế, điều này có thể áp dụng vào nhiều lĩnh vực, bao gồm lập trình tự động, hệ thống thông minh, và cả xe tự lái.

### **I.2. Một số khái niệm :**

#### **I.2.1. Dạng trò chơi:**

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

1. **Bản đồ (Map)**: Trò chơi thường diễn ra trên một bản đồ gồm các ô vuông. Mỗi ô có một chức năng cụ thể:
   * **Không gian trống (Floor)**: Được đại diện bởi các ô trống, mà người chơi và hộp có thể di chuyển vào.
   * **Tường (Wall)**: Được đại diện bởi các ô chứa tường, mà không thể di chuyển vào.
   * **Mục tiêu (Goal)**: Là các ô mà người chơi cần di chuyển hộp đồ tới. Mục tiêu thường được ký hiệu bằng các điểm.
   * **Hộp đồ (Box)**: Là các hộp cần di chuyển đến các ô là mục tiêu.
2. **Người chơi (Player)**: Người chơi điều khiển một nhân vật (thường được biểu thị bằng một biểu tượng) để di chuyển các hộp đồ từ vị trí ban đầu đến các ô là mục tiêu.
3. **Quy tắc di chuyển**:
   * Người chơi có thể di chuyển lên, xuống, trái, hoặc phải, nhưng chỉ được di chuyển vào các ô trống.
   * Người chơi có thể đẩy hộp đồ bằng cách đứng bên cạnh hộp và tiến vào hộp, đẩy nó trong hướng tương ứng. Hộp không thể di chuyển qua các ô tường hoặc qua hộp khác.
4. **Mục tiêu của trò chơi**: Mục tiêu của trò chơi là di chuyển tất cả các hộp đồ từ vị trí ban đầu đến các ô là mục tiêu, đặt chính xác trên tất cả các mục tiêu. Người chơi cần phải tìm cách sắp xếp các hộp để đạt được mục tiêu này.

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

#### I.**2.2. Cây trò chơi:**

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

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

1. Trạng Thái (State): Mỗi nút trên cây trò chơi đại diện cho một trạng thái cụ thể của trò chơi. Trạng thái này bao gồm vị trí của người chơi, vị trí của các hộp đồ, vị trí của các mục tiêu và cấu trúc bản đồ.
2. Nút (Node): Mỗi nút trên cây trò chơi là một trạng thái cụ thể của trò chơi tại một thời điểm cụ thể. Cây trò chơi bắt đầu với một nút gốc, đại diện cho trạng thái ban đầu của trò chơi.
3. Cây Trò Chơi (Game Tree): Cây trò chơi là một biểu đồ cây có cấu trúc, trong đó mỗi nút biểu thị một trạng thái của trò chơi, và các cạnh biểu thị các hành động mà người chơi có thể thực hiện để chuyển từ một trạng thái sang một trạng thái khác.
4. Trạng Thái Bắt Đầu (Initial State): Trạng thái ban đầu của trò chơi, thường chứa thông tin về vị trí ban đầu của người chơi, các hộp đồ, và các mục tiêu.
5. Trạng Thái Kết Thúc (Goal State): Trạng thái mà bạn cố gắng đạt được trong trò chơi Sokoban, trong đó tất cả các hộp đồ đã được di chuyển đến các mục tiêu.
6. Hành Động (Action): Các hành động mà người chơi có thể thực hiện để thay đổi trạng thái của trò chơi, bao gồm di chuyển lên, xuống, trái hoặc phải, hoặc đẩy hộp đồ.
7. Thuật Toán Tìm Kiếm (Search Algorithm): Trong cây trò chơi, thuật toán tìm kiếm được sử dụng để duyệt qua các nút trạng thái để tìm ra một dãy hành động dẫn đến trạng thái kết thúc. Thuật toán tìm kiếm có thể làm cho việc giải quyết trò chơi Sokoban tự động trở nên hiệu quả.
8. Cắt Tự Nhiên (Pruning): Cắt tự nhiên là quá trình loại bỏ các nút trong cây trò chơi mà không cần thiết để giảm bớt độ phức tạp của thuật toán tìm kiếm.

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

### **I.3. Công nghệ và thuật toán áp dụng :**

#### **I.3.1. Ngôn ngữ Python**

Python là một ngôn ngữ lập trình phổ biến và mạnh mẽ được sử dụng rộng rãi trong lĩnh vực Trí tuệ nhân tạo (Artificial Intelligence - AI) và phân tích dữ liệu. Trong bài tập Sokoban AI, Python có thể được sử dụng để triển khai thuật toán và xây dựng trò chơi Sokoban.

Python có nhiều ưu điểm khi sử dụng trong lĩnh vực AI và phân tích dữ liệu:

Dễ học và đọc: Python có cú pháp đơn giản, gần gũi với ngôn ngữ tự nhiên. Điều này làm cho việc học và đọc mã Python dễ dàng hơn so với nhiều ngôn ngữ lập trình khác. Điều này rất hữu ích khi làm việc với các thuật toán phức tạp trong AI.

Thư viện phong phú: Python có một cộng đồng đông đảo và phát triển mạnh mẽ, cung cấp nhiều thư viện và công cụ hỗ trợ cho AI như NumPy, pandas, scikit-learn, TensorFlow và PyTorch. Nhờ vào những thư viện này, việc triển khai các thuật toán AI trở nên dễ dàng và hiệu quả.

Tích hợp dễ dàng: Python có khả năng tích hợp với các ngôn ngữ và công nghệ khác. Bạn có thể sử dụng Python để giao tiếp với các thư viện và công cụ được viết bằng ngôn ngữ khác, như C++ và Java. Điều này cho phép kết hợp các thành phần chuyên môn và tận dụng những ưu điểm của từng ngôn ngữ.

Đa nền tảng: Python được hỗ trợ trên nhiều nền tảng, bao gồm Windows, macOS và Linux. Điều này cho phép bạn phát triển và chạy mã Python trên các môi trường khác nhau mà không cần thay đổi nhiều.

Cộng đồng và tài liệu phong phú: Python có một cộng đồng lớn và nhiệt tình, với nhiều nguồn tài liệu phong phú và hướng dẫn trực tuyến. Bạn có thể dễ dàng tìm kiếm và học hỏi từ những người khác trong cộng đồng Python.

Với những lợi thế này, Python là một lựa chọn tốt để triển khai các dự án trí tuệ nhân tạo.

#### **I.3.2. Thuật toán A\***

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

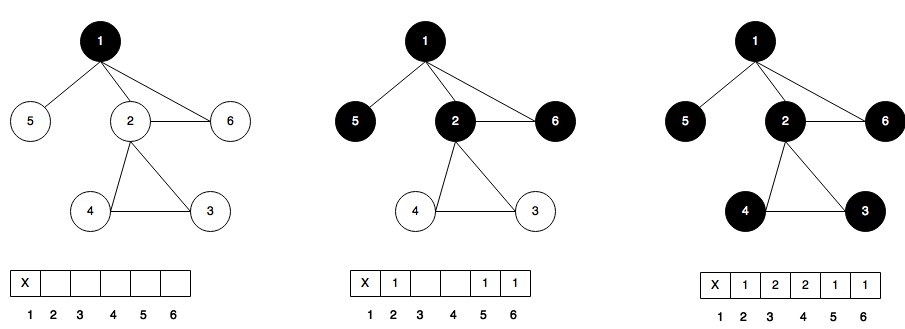
Dưới đây là mô tả cụ thể về cách thuật toán A\* hoạt động:

1. Khởi tạo: Bắt đầu từ trạng thái ban đầu, tạo một danh sách có trạng thái ban đầu và gán chi phí gốc (g) cho nó là 0.
2. Danh Sách Mở (Open List): Tạo một danh sách mở để lưu trữ các trạng thái chưa được duyệt. Đưa trạng thái ban đầu vào danh sách này.
3. Danh Sách Đóng (Closed List): Tạo một danh sách đóng để lưu trữ các trạng thái đã được duyệt.
4. Lặp cho đến khi danh sách mở trống:
   * Chọn trạng thái (nút) có ước tính chi phí thấp nhất (f) từ danh sách mở.
   * Di chuyển trạng thái này từ danh sách mở sang danh sách đóng.
   * Kiểm tra nếu trạng thái này là trạng thái kết thúc. Nếu có, dừng thuật toán và trả về đường đi tối ưu.
   * Duyệt qua các trạng thái con (nếu có) của trạng thái hiện tại:
     + Tính toán chi phí g từ trạng thái ban đầu đến trạng thái con thông qua trạng thái hiện tại.
     + Tính toán chi phí ước tính h từ trạng thái con đến trạng thái kết thúc bằng cách sử dụng hàm heuristic.
     + Tính toán chi phí tổng f = g + h.
     + Nếu trạng thái con không nằm trong danh sách mở hoặc chi phí f từ trạng thái ban đầu đến trạng thái con thấp hơn chi phí đã biết, cập nhật chi phí và thêm trạng thái con vào danh sách mở.
5. Nếu danh sách mở trống mà không tìm thấy đường đi đến trạng thái kết thúc, thì không có đường đi tới trạng thái đó.

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

#### **I.3.3. Thuật toán BFS**

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



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

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

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

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

#### **I.3.4. Chiến lược tìm kiếm**

Trong trò chơi Sokoban, để giải quyết các cấp độ và tìm kiếm đường đi tối ưu từ trạng thái ban đầu đến trạng thái kết thúc, người chơi hoặc các trình điều khiển trí tuệ nhân tạo có thể áp dụng các chiến lược tìm kiếm khác nhau.

1. ***Tìm Kiếm Đường Đi Tối Ưu (A* Search)\*:**

* A\* Search là một chiến lược tìm kiếm thông minh sử dụng hàm heuristic để ước tính chi phí từ trạng thái hiện tại đến trạng thái kết thúc.
* A\* Search duyệt qua các trạng thái con theo thứ tự ước tính chi phí tổng thấp nhất (f = g + h), với g là chi phí đã biết và h là ước tính chi phí từ trạng thái con đến trạng thái kết thúc.
* Hàm Heuristic Euclidean (Khoảng cách Euclide):

Hàm heuristic Euclidean tính khoảng cách theo đường thẳng giữa hai điểm trên bản đồ. Trong Sokoban, nó có thể được sử dụng để đo khoảng cách từ vị trí hiện tại của hộp đến mục tiêu gần nhất. Hàm này đơn giản và cho kết quả tốt nếu không có rào cản trên đường đi.

* Hàm Heuristic Manhattan (Khoảng cách Manhattan):

Hàm heuristic Manhattan tính khoảng cách bằng cách cộng tất cả các khoảng cách theo chiều ngang và chiều dọc giữa hai điểm. Trong Sokoban, nó có thể được sử dụng để đo khoảng cách từ vị trí hiện tại của hộp đến mục tiêu gần nhất. Hàm này đưa ra kết quả chính xác trong môi trường có các hộp và rào cản.

* A\* sẽ tìm kiếm theo chiều rộng, ưu tiên các trạng thái có tổng chi phí nhỏ nhất. Khi nó tìm thấy một đường đi tiềm năng, nó sẽ tiếp tục mở rộng và kiểm tra các trạng thái con cho đến khi nó tìm thấy một đường đi hoặc xác định rằng không có đường đi nào tồn tại.
* Kết quả là, A\* kết hợp với hàm heuristic Euclidean hoặc Manhattan giúp tìm ra đường đi tối ưu trong trò chơi Sokoban một cách hiệu quả và nhanh chóng bằng cách định rõ mục tiêu và ước tính khoảng cách còn lại đến mục tiêu.

1. ***Tìm kiếm đường đi BFS (Breadth-First Search):***

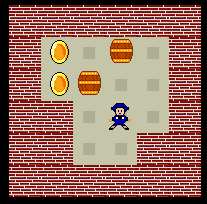
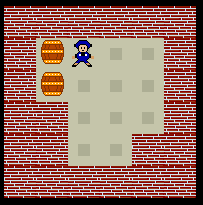
* BFS là một thuật toán tìm kiếm không có trọng số, nó duyệt qua tất cả các trạng thái cùng mức trước khi di chuyển xuống cấp tiếp theo. Trong Sokoban, nó có thể được sử dụng để tìm một đường đi từ vị trí ban đầu đến mục tiêu. Tuy nhiên, BFS không xem xét chi phí của việc đi lại nên nó không đảm bảo tìm đường đi ngắn nhất.
* Để sử dụng BFS trong Sokoban, bạn bắt đầu từ vị trí ban đầu của người chơi và duyệt qua tất cả các trạng thái con có thể đạt được từ trạng thái hiện tại. Bạn tiếp tục duyệt qua các trạng thái mới được tạo ra cho đến khi bạn tìm thấy mục tiêu hoặc bạn đã duyệt qua tất cả các trạng thái mà không tìm thấy lời giải.

# **II. Giao diện ,chức năng, hệ thống trò chơi**

### ***II.1.* *Các map trong trò chơi***

#### **II.1.1. Tổng quát**

Đểmang đến mộttrò chơi thú vị đơn giản nhưng cũng không kém phần thử thách nhóm em đã tạo ra 30 màn chơi khác nhau với đủ 4 cấp độ easy , medium, difficult, super difficult

 *(map ban đầu) (map hoàn thành)*

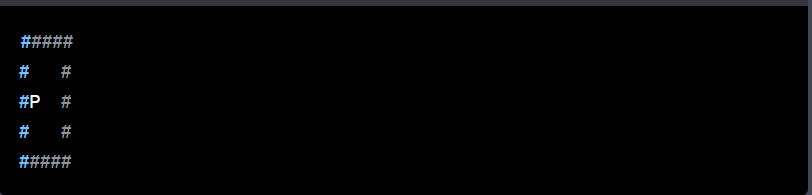
Đây chỉ là một màn đơn giản của trò chơi chúng ta có thể thử các map còn lại để có thể hiểu và đánh giá rõ hơn về các thuật toán tìm kiếm không gian.

#### **II.1.2. Mô hình hóa các map trong trò chơi**

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

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

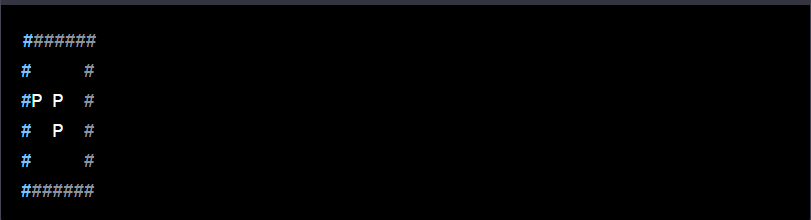
1. Màn Chơi Dễ Dàng:

Bản đồ này có một số hộp và vị trí mục tiêu. Cấu trúc đơn giản giúp bạn làm quen với cách di chuyển hộp và đánh bại màn chơi đầu tiên.

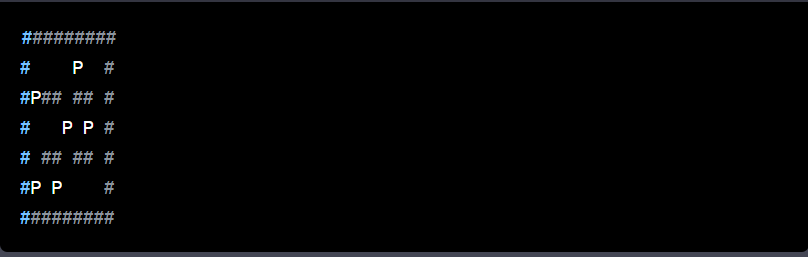
1. Màn chơi trung bình

Bản đồ này phức tạp hơn với nhiều hộp và vị trí mục tiêu. Bạn sẽ cần tính toán kỹ lưỡng và tạo ra một chiến lược để đẩy hộp đúng cách.

1. Màn chơi khó

Bản đồ này thách thức kỹ năng của bạn với nhiều hộp, vị trí mục tiêu và rào cản. Đây là một trong những màn chơi khó nhất mà bạn sẽ phải đối mặt.

1. Màn chơi siêu khó

Bản đồ này là một thách thức thực sự với một mạng lưới phức tạp của hộp và vị trí mục tiêu. Để hoàn thành màn chơi này, bạn cần có sự tư duy chiến lược và khả năng tối ưu hóa.

### **II.2. Hệ thống trò chơi**

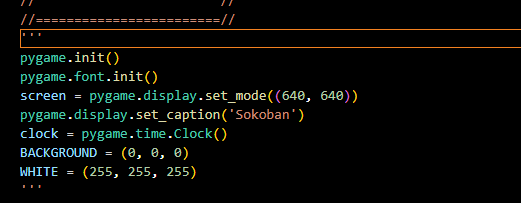
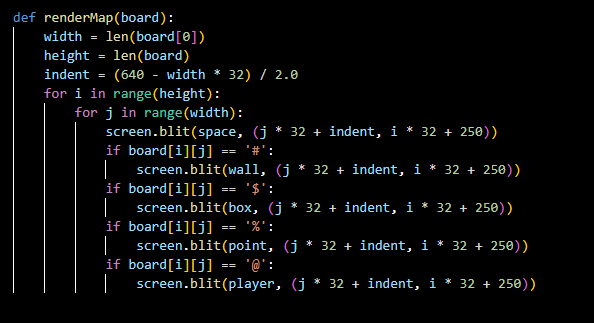
#### **II.2.1. Các thư viện được áp dụng**

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

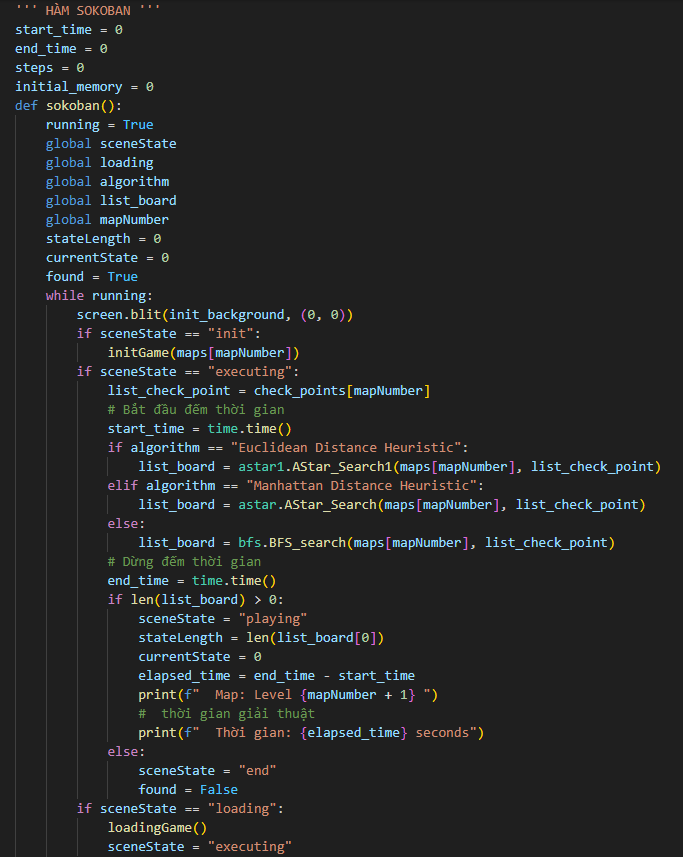
* Thư viện NumPy (Numerical Python) là một thư viện mạnh mẽ và phổ biến trong Python dùng cho tính toán số học và xử lý dữ liệu đa chiều. NumPy cho phép bạn làm việc với mảng và ma trận nhanh chóng và hiệu quả, giúp thực hiện các phép toán số học, thống kê, và xử lý dữ liệu trong Python dễ dàng hơn,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

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

* Thư viện Pygame là một thư viện phát triển trò chơi và đồ họa trong ngôn ngữ lập trình Python. Pygame cung cấp một cơ sở hạ tầng để tạo ra các ứng dụng đa phương tiện, đặc biệt là trò chơi. Nó hỗ trợ đồ họa, âm thanh, nhập liệu và cung cấp các công cụ để xây dựng các trò chơi trên nền tảng đa nền tảng.
* Sử dụng Pygame để tạo một màn chơi Sokoban :

1. Cài đặt và Khởi tạo Pygame
2. Tạo Màn Hình và Biểu Diễn Trạng Thái:
3. Vẽ Trạng Thái Màn Chơi

4) Xử Lý Sự Kiện Người Chơi:





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

#### **II.2.2. Áp dụng thuật toán**

* **Thuật toán A\* (A-star):**
* A\* là một thuật toán tìm kiếm đường đi trong đồ thị hoặc lưới dựa trên kết hợp giữa tìm kiếm theo chiều rộng (BFS) và tìm kiếm theo độ ưu tiên (best-first search).
* Thuật toán A\* sử dụng một hàm heuristic để ước lượng chi phí từ điểm hiện tại đến điểm đích, kết hợp với chi phí thực tế đã đi qua để đưa ra quyết định tìm kiếm.
* A\* duyệt qua các ô trong không gian tìm kiếm dựa trên giá trị heuristic và cost gần nhất, tìm kiếm đường đi ngắn nhất từ điểm bắt đầu đến điểm đích.
* **Hàm Euclide:**
* Hàm Euclide là một hàm heuristic được sử dụng trong thuật toán A\* để ước lượng chi phí từ điểm hiện tại đến điểm đích.
* Hàm Euclide tính khoảng cách Euclide (khoảng cách theo đường thẳng) giữa hai điểm trên mặt phẳng.
* Công thức tính khoảng cách Euclide giữa hai điểm (x1, y1) và (x2, y2) là h = sqrt((x2 - x1)^2 + (y2 - y1)^2).
* Trong trò chơi Sokoban, vị trí hiện tại của người chơi là điểm bắt đầu và vị trí của hộp cần di chuyển đến là điểm mục tiêu.
* Hàm Euclide ước lượng chi phí còn lại từ vị trí hiện tại đến vị trí mục tiêu dựa trên khoảng cách Euclide, giúp thuật toán A\* tìm kiếm đường đi tối ưu.
* **Hàm Manhattan:**
* Manhattan cũng là một hàm heuristic được sử dụng trong thuật toán A\* để ước lượng chi phí từ điểm hiện tại đến điểm đích.
* Hàm Manhattan tính khoảng cách Manhattan (tổng giá trị tuyệt đối) giữa hai điểm trên mặt phẳng.
* Công thức tính khoảng cách Manhattan giữa hai điểm (x1, y1) và (x2, y2) là h = |x2 - x1| + |y2 - y1|.
* Trong trò chơi Sokoban, vị trí hiện tại của người chơi là điểm bắt đầu và vị trí của hộp cần di chuyển đến là điểm mục tiêu.
* Hàm Manhattan ước lượng chi phí còn lại từ vị trí hiện tại đến vị trí mục tiêu dựa trên khoảng cách Manhattan, giúp thuật toán A\* tìm kiếm đường đi tối ưu.
* **Thuật toán tìm kiếm theo chiều rộng (BFS - Breadth-First Search):**
* BFS là một thuật toán tìm kiếm đồ thị hoặc cây dựa trên việc khám phá và duyệt qua các đỉnh/tuần tự các cấp của đồ thị/cây.
* Thuật toán bắt đầu từ một đỉnh gốc và mở rộng các đỉnh kề của đỉnh đang xét trước khi di chuyển đến các đỉnh kề tiếp theo.
* BFS sử dụng cơ chế hàng đợi (queue) để lưu trữ các đỉnh chờ xét, đảm bảo rằng các đỉnh gần nhất sẽ được duyệt trước.

Cách thức thực hiện BFS:

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

Bước 2: Lặp cho đến khi hàng đợi trống:

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

Bước 3: Nếu hàng đợi trống và không tìm thấy đỉnh đích,

thuật toán kết thúc và trả về kết quả không tìm thấy.

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

Kiểm tra tính khả thi:

* Trước khi thực hiện một bước di chuyển, thuật toán BFS có thể được áp dụng để kiểm tra tính khả thi của hướng di chuyển đó.
* Bằng cách tạo một trạng thái mới và kiểm tra xem trạng thái đó có hợp lệ hay không (ví dụ: không đi vào tường hoặc đẩy hộp ra khỏi bản đồ), ta có thể xác định các bước di chuyển hợp lệ mà người chơi có thể thực hiện.

Tạo các cấp độ mới:

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

### **II.3. Giới thiệu các thư mục code và cách chạy trò chơi**

#### **II.3.1. Giới thiệu các thư mục code**

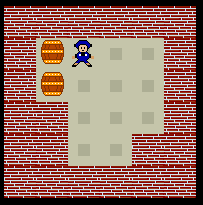
* Thư mục Assets: Chứa các file tài nguyên chính của game như hình ảnh, font chữ,logo game,...
* Thư mục Testcases: Chứa các file .txt với mỗi file là một map của game,trong từng file.txt chứa các ký hiệu được chỉ định là một đối tượng trong một map.
* Thư mục Checkpoints: Chứa các file .txt với mỗi file là chứa tọa độ x,y của các điểm cần phải đẩy thùng tương ứng với số thứ tự mỗi file trong thư mục Testcases.
* Thư mục Sources: chứa các file code python chính cho game.
* File astar : triển khai thuật toán tìm kiếm A\* với hàm **Manhattan Heuristic.**
* File astar1 : triển khai thuật toán tìm kiếm A\* với hàm **Euclidean Heuristic.**
* File bfs : Thư mục "bfs" triển khai thuật toán tìm kiếm theo chiều rộng (BFS) để giải quyết tìm đường đi trong trò chơi Sokoban.
* File main: Thư mục “main” chứa các logic chính của game ví dụ như cách render giao diện, xử lý tuần tự các trạng thái như lúc khởi tạo,bắt đầu,…..
* File support\_function: Thư mục “support\_function” chứa một tập hợp các hàm hỗ trợ cho việc giải bài toán trong trò chơi Sokoban bằng Python. Ở đây file này hỗ trợ các hàm phụ cho thuật toán BFS và thuật toán A\* chứa Mahattan Heuristic.
* File support\_function1: Thư mục “support\_function1” chứa một tập hợp các hàm hỗ trợ cho việc giải bài toán trong trò chơi Sokoban bằng Python. Ở đây file này hỗ trợ các hàm phụ cho thuật toán A\* chứa Mahattan Heuristic.

#### **II.3.2. Cách chạy trò chơi:**

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

 *(giao diện bắt đầu)*

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

 *(sau khi hoàn thành)*

# **III. Kết quả, kiểm thử và đánh giá**

### **III.1. Kết quả**

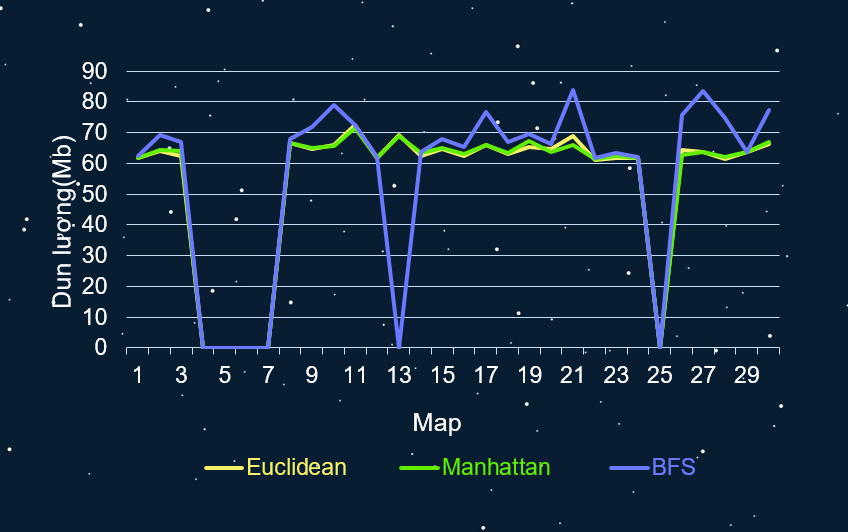
* Các thuật toán được thực hiện bằng cách sử dụng hai đặc điểm quan trọng. Một là thời gian chờ và cái còn lại là độ sâu tối đa. Các thuật toán được thực hiện để dừng khi đạt đến độ sâu được xác định trước nhất định được gọi là độ sâu tối đa và cũng bị chấm dứt khi vượt quá thời gian tối đa, hết thời gian chờ. Điều này là cần thiết để ngăn chặn các thuật toán tìm kiếm trong không gian tìm kiếm vô hạn.
* Chúng em đã xem xét hai phương pháp phỏng đoán cho Thuật toán A\*. Phương pháp phỏng đoán đầu tiên được triển khai bằng cách xem xét khoảng cách tối thiểu được tính bằng khoảng cách Euclide giữa mục tiêu và các hộp. Heuristic thứ hai được tính toán bằng cách tính khoảng cách tối thiểu với khoảng cách Manhattan giữa chúng.

Cả hai hàm heuristic Euclidean và Manhattan đều có thể cung cấp một ước lượng về khoảng cách từ trạng thái hiện tại đến trạng thái đích. Tuy nhiên, hiệu suất tìm kiếm có thể khác nhau vì cách mà hai hàm đánh giá khoảng cách.

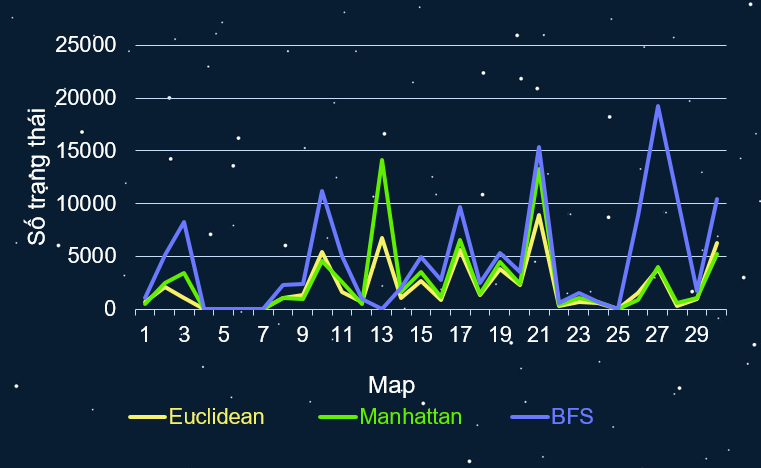
Cả hai hàm Euclidean và Manhattan đều có thể cung cấp lời giải chính xác trong trò chơi Sokoban. Tuy nhiên, độ chính xác của lời giải có thể khác nhau và phụ thuộc vào cấu trúc của bản đồ và các trạng thái trong trò chơi.

Sử dụng hàm Euclidean có thể yêu cầu tính toán khoảng cách theo đường thẳng giữa các trạng thái, trong khi hàm Manhattan đánh giá khoảng cách theo đường chéo hoặc theo số bước ngang, dọc. Việc tính toán theo đường thẳng có thể tốn nhiều thời gian và tài nguyên tính toán hơn so với tính toán theo đường chéo hoặc số bước. Do đó, sử dụng hàm Manhattan có thể có hiệu quả thời gian và tài nguyên tốt hơn so với hàm Euclidean.

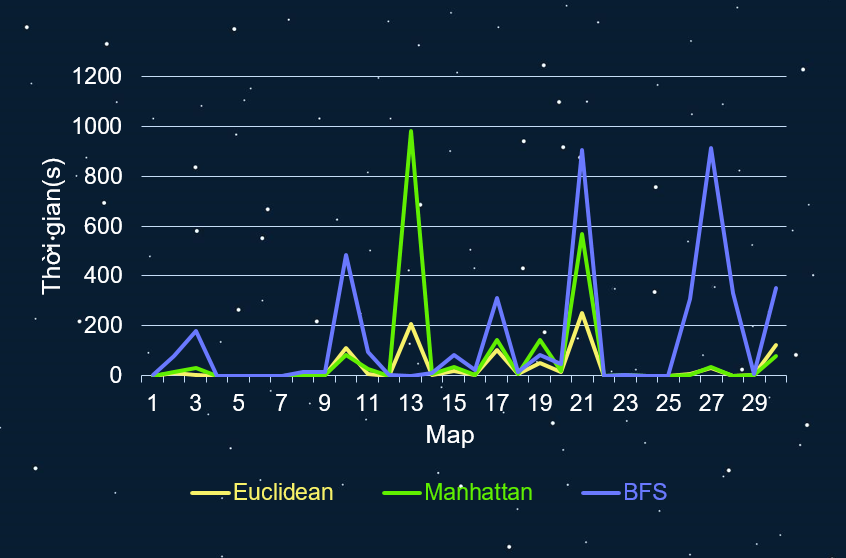
* Đối với BFS:
* Độ phức tạp thời gian và không gian: BFS có độ phức tạp thời gian và không gian lớn. Trong trò chơi Sokoban với bản đồ lớn và nhiều hộp, số lượng trạng thái có thể phát sinh rất lớn, dẫn đến việc tạo ra một hàng đợi rất lớn và yêu cầu nhiều bộ nhớ. Điều này có thể làm cho việc tìm kiếm trở nên chậm và không khả thi trong các trường hợp phức tạp.
* Không tối ưu hóa: BFS duyệt qua tất cả các trạng thái có thể từ trạng thái ban đầu, ngay cả khi chúng không đưa đến trạng thái đích. Điều này có nghĩa là BFS không có khả năng ưu tiên tìm kiếm các trạng thái tiềm năng tốt nhất hoặc tìm kiếm theo hướng nhanh nhất đến trạng thái đích. Trong trò chơi Sokoban, có thể tồn tại những cấu trúc bản đồ phức tạp và các trường hợp mà BFS không tối ưu.
* Không xử lý được tình huống quay lui: BFS không xử lý tốt các tình huống quay lui trong trò chơi Sokoban. Trong một số trường hợp, để đẩy một hộp vào ô đích, người chơi có thể cần đẩy hộp qua một vị trí cần phải quay lại sau đó. BFS không tự động xử lý được tình huống này mà cần phải sử dụng các phương pháp khác như backtracking để giải quyết.
* Không sử dụng thông tin heuristic: BFS không sử dụng thông tin heuristic (đánh giá ước lượng) để ưu tiên tìm kiếm các trạng thái tiềm năng tốt nhất hoặc tìm kiếm theo hướng nhanh nhất đến trạng thái đích. Trong trò chơi Sokoban, có thể sử dụng các thông tin heuristic như khoảng cách Manhattan giữa các hộp và ô đích để tăng hiệu suất tìm kiếm. Tuy nhiên, BFS không có khả năng tự động áp dụng các heuristic này.

*( BIỂU ĐỒ SO SÁNH DUNG LƯỢNG KHI CHẠY BFS VÀ HAI HÀM HEURISTIC TRONG GIẢI THUẬT A\* TRÊN 30 MAP )*

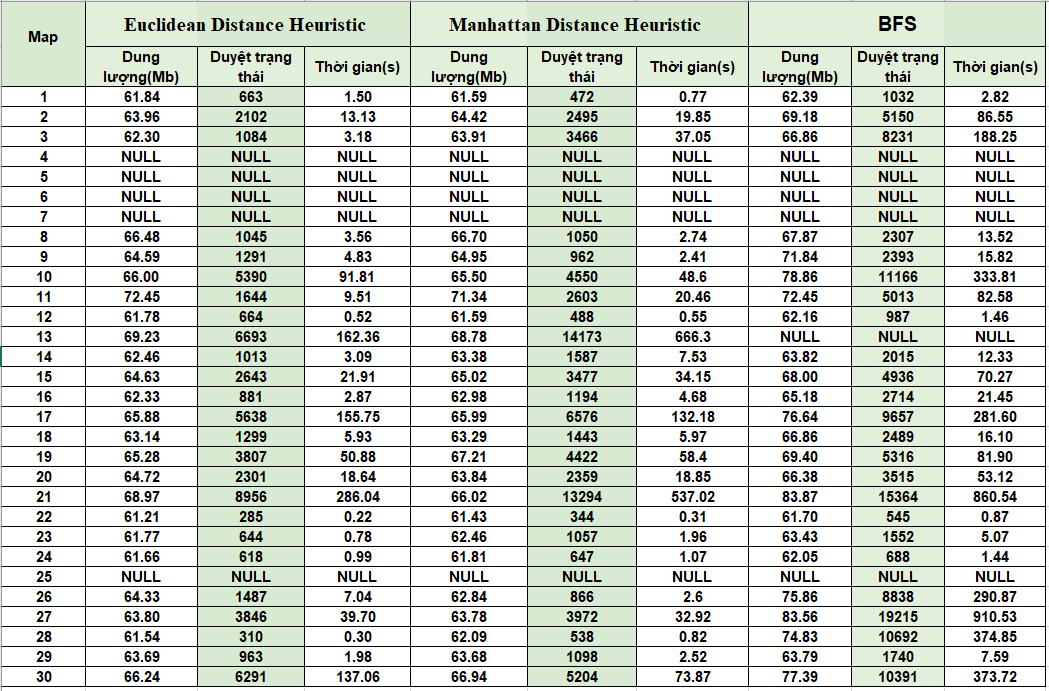
*( BIỂU ĐỒ SO SÁNH SỐ TRẠNG THÁI CẦN DUYỆT KHI CHẠY BFS VÀ HAI HÀM HEURISTIC TRONG GIẢI THUẬT A\* TRÊN 30 MAP )*

**

*( BIỂU ĐỒ SO SÁNH THỜI GIAN CHẠY BFS VÀ HAI HÀM HEURISTIC TRONG GIẢI THUẬT A\* TRÊN 30 MAP )*

****

*BẢNG THU THẬP DỮ LIỆU QUA 30 MAP*



**Nhận xét**:

Qua bảng trên cho ta thấy rằng:

* Thuật toán BFS thường có thời gian tìm đường đi, sử dụng dung lượng và số trang thái cần duyệt cao hơn hai thuật toán còn lại vì bắt buộc nó phải vét cạn cây (Map 27: 83,56Mb - 910,53s – 19215 trạng thái).
* Hàm Euclidean luôn duyệt ít trạng thái nhất (Map 22: 285 trạng thái)
* Giữa 2 hàm Euclidean Distance Heuristic và Manhattan Distance Heuristic thì hàm Eucidean tối ưu hơn về mọi mặt
* Một số map đặc biệt như:
* Map 4, 5, 6, 7: Cả giải thuật A\* và BFS đều không đưa ra được kết quả vì thời gian để thực hiện giải thuật vượt quá thời gian cho phép
* Map 13: Chỉ hai hàm Euclidean và Manhattan của giải thuật A\* chạy được ngược lại giải thuật BFS lại có thời gian chạy quá lâu dẫn đến Time Out và thoát game 🡪 từ ví dụ map 13 trên càng cho ta thấy giải thuật A\* tối ưu hơn BFS về thời gian đến mức nào
* Map 25: Hai giải thuật đều không đưa ra được kết quả và in ra Not Found

### **III.2. Kết Luận :**

Chúng em đã triển khai nhiều thuật toán để phát triển tác nhân AI cho trò chơi Sokoban. Chúng em đã thử nghiệm một số kỹ thuật tối ưu và quan sát thấy rằng hầu hết các thuật toán đều có những bước đi tương tự khi chơi trò chơi nhưng hiệu suất của chúng khác nhau về thời gian và trạng thái được khám phá. Với kích thước cấp độ nhỏ hơn, thuật toán A\* với số liệu được tính bằng Manhattan và Eucliden cho thấy có hiệu suất tốt hơn so với thuật toán BFS được xem xét. Nhưng với kích thước cấp độ lớn hơn, tất cả các thuật toán đều bị hạn chế bởi thời gian .

# **Tài liệu tham khảo**

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

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

1. <http://pavel.klavik.cz/projekty/solver/solver.pdf>
2. Taylor, Joshua, and Ian Parberry. "Procedural generation of Sokoban levels." Proceedings of the International North American Conference on Intelligent Games and Simulation. 2011. (<https://ianparberry.com/pubs/GAMEON-NA_METH_03.pdf> )

# phân công công việc

1. Phạm Hải Đức: tìm hiểu Python, các thuật toán tìm kiếm, tổng hợp nội dung, làm báo cáo, thuyết trình.

2. Phan Văn Tươi: tìm hiểu thuật toán, lập trình cơ sở dữ liệu, kiểm thử, đánh giá kết quả, thuyết trình.

3. Lê Đình Triều: tìm hiểu thuật toán BFS, thiết kế giao diện, cài đặt thử nghiệm.

4. Lương Sơn Bá: tìm hiểu ngôn ngữ Python và thuật toán A\*, áp dụng thuật toán tìm kiếm vào code, thuyết trình.

5. Cao Trần Đức Mạnh: tìm hiểu các hàm Heuristics ( Euclide và Manhathan ..) và cách hoạt động của hàm vào game, thuyết trình.