**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG – HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO ĐỒ ÁN:**

**PROJECT 1: SEARCH**

**“Let’s chase Pac-Man!”**

**Giảng viên: NGUYỄN TIẾN HUY**

MSSV

23127356

23127338

23127337

23127143

Họ và tên

ĐINH TUẤN DUY

LỮ BẢO ĐẠT

BÙI ĐỨC ĐẠT

THIỀU QUANG VINH

TP. Hồ Chí Minh – 03/2025

**MỤC LỤC**

[**1.** **GIỚI THIỆU** 3](#_Toc192976335)

[**2.** **MÔ TẢ** 3](#_Toc192976336)

[**2.1.** **Luật chơi** 3](#_Toc192976337)

[**2.2.** **Các yếu tố trong game** 4](#_Toc192976338)

[**2.3.**  **Giao diện người chơi** 4](#_Toc192976339)

[**3.** **THUẬT TOÁN** 4](#_Toc192976340)

[**3.1.** **Breadth-first search (BFS)** 4](#_Toc192976341)

[**3.2.** **Uniform-cost search (UCS)** 5](#_Toc192976342)

[**3.3.** **Depth-first search (DFS)** 5](#_Toc192976343)

[**3.4.** **A\* search** 5](#_Toc192976344)

[**4.** **KIỂM TRA THUẬT TOÁN** 5](#_Toc192976345)

[**4.1.** **BFS Performance** 6](#_Toc192976346)

[**4.2.** **UCS Performance** 7](#_Toc192976347)

[**4.3.** **DFS Performance** 8](#_Toc192976348)

[**4.4.** **A\* Search Performance** 9](#_Toc192976349)

[**5.** **BẢNG PHÂN CÔNG CÔNG VIỆC** 9](#_Toc192976350)

[**TÀI LIỆU THAM KHẢO** 11](#_Toc192976351)

**MỤC LỤC HÌNH ẢNH**

# **GIỚI THIỆU**

Pac-Man là một tựa game giải trí – nơi mà người chơi sẽ phải điều khiển nhân vật

Pacman di chuyển khắp mê cung và thu thập nhiều chấm ghi điểm nhất có thể. Trò chơi sẽ kết thúc khi các chấm ghi điểm đã được thu thập hết hoặc khi Pac-Man bị con ma bắt được. Trong đồ án này, nhóm đã tạo ra một phiên bản đơn giản hơn của game bằng cách tối giản giao diện người dung (text-base game) để thể hiện trực quan nhất những gì đang diễn ra và một phiên bản game 2d.

A screenshot of a video game

AI-generated content may be incorrect.

A game of a video game

AI-generated content may be incorrect.

# **MÔ TẢ**

## **Luật chơi**

1. Người chơi sẽ di chuyển khắp mê cung để thu thập các chấm ghi điểm trong khi tránh chạm mặt với các con ma đang đuổi theo.
2. Trò chơi sẽ kết thúc khi các chấm ghi điểm đã hết hoặc khi người chơi đã bị ma bắt.
3. Trên bản đồ sẽ xuất hiện những ô “ám”. Các con ma bước vào ô sẽ được tăng tốc.
4. Người chơi sử dụng 4 phím WASD hoặc ←↑→↓ để di chuyển và ấn phím Q để đầu hàng. Mỗi lần sẽ di chuyển được 1 ô, tối thiểu 0.2 giây di chuyển một lần.

## **Các yếu tố trong game**

A screen shot of a computer

AI-generated content may be incorrect.

Player: Nhân vật do người chơi điều khiển

B(nlue) Ghost: Con ma đuổi theo Player bằng thuật toán BFS

P(pink Ghost: Con ma đuổi theo Player bằng thuật toán A\*

R(red) Ghost: Con ma đuổi theo Player bằng thuật toán UCS

O(orange) Ghost: Con ma đuổi theo Player bằng thuật toán DFS

Haunted Point: Ô gia tốc cho con ma

Wall: Vật cản không thể đi xuyên qua

## **Giao diện người chơi**

A screenshot of a video game

AI-generated content may be incorrect.

A game of a video game

AI-generated content may be incorrect.

# Hướng dẫn cài đặt và sử dụng

3.1 Tổng quan

Phiên bản python: python 3.13.2

Thư viện sử dụng: thư viện chuẩn của python

Thư viện cài đặt thêm:

* pygame
* colorama

Tham số dòng lệnh:

* -text: hiển thị game text-base
* -2d: hiển thị game 2d
* -graph: mô tả trực quan cách mà graph được biểu diễn
* -algo: hiển thị đường đi của thuật toán từ cùng một vị trí đến người chơi; mô tả các thông số kỹ thuật: thời gian xử lý, bộ nhớ , số bước đi, tổng trọng số, nước đi kế tiếp vị trí hiện tại, số vị trí duy nhất (không tính các nước đi lặp lại).

3.2 Hướng dẫn cài đặt.

a) Cài đặt nhanh:

B1: Chuẩn bị sẵn git, python3.13.2

B2: Nhập các lệnh sau trong command prompt để cài đặt nhanh source:

* git clone <https://github.com/dtduy23/Pacman.git>

B3: Cài đặt thư viện:

* pip install pygame
* pip install colorama

B4 : chạy file main (tùy chọn) để hiển thị những thông tin cần thiết.

* python -u “path file main”

ví dụ: python -u "d:\Năm 2\Kỳ 2\csAI\project1\source\main.py"

* màn hình sẽ hiển thị các tham số cho người dung nhập vào:

usage: main.py [-h] (-text | -2d | -graph | -algo)

main.py: error: one of the arguments -text -2d -graph -algo is required

B5 : chạy chương trình

* python -u “path file main” -text

A screenshot of a video game

AI-generated content may be incorrect.

* python -u “path file main” -2d

A game screen with a maze and dots

AI-generated content may be incorrect.

* python -u “path file main” -graph

hiển thị vị trí mặc định : 1,1.

Vị trí thực nghiệm hiện tại 3,3

A black screen with white text

AI-generated content may be incorrect.

* python -u “path file main” -algo

A screenshot of a computer

AI-generated content may be incorrect.

# **THUẬT TOÁN**

4.1 Cài đặt graph: Cài đặt bản đồ game thành graph cho phép cài đặt các thuật toán của ghost.

game\_map.py cài đặt cấu trúc của bản đồ trò chơi thông qua lớp Map:

Cấu trúc lớp Map

class Map:

def \_\_init\_\_(self, layout):

self.layout = layout # Ma trận 2D chứa các ký tự đại diện cho các đối tượng trên bản đồ

self.height = len(layout) # Chiều cao bản đồ

self.width = len(layout[0]) # Chiều rộng bản đồ

self.player\_pos = self.find\_position(PLAYER) # Vị trí người chơi

self.ghost\_positions = { # Dictionary lưu vị trí của các ma

BLUE\_GHOST: self.find\_position(BLUE\_GHOST),

PINK\_GHOST: self.find\_position(PINK\_GHOST),

RED\_GHOST: self.find\_position(RED\_GHOST),

ORANGE\_GHOST: self.find\_position(ORANGE\_GHOST)

}

self.haunted\_points = self.find\_all\_positions(HAUNTED\_POINT) # Danh sách vị trí haunted points

self.dots = self.count\_dots() # Số lượng điểm (dấu chấm) trên bản đồ

Các phương thức chính

1. find\_position(char): Tìm vị trí đầu tiên của ký tự `char` trên bản đồ

2. find\_all\_positions(char): Tìm tất cả vị trí của ký tự `char` trên bản đồ

3. count\_dots(): Đếm số lượng điểm (dấu chấm) trên bản đồ

4. load\_map(filename): Phương thức tĩnh để tải bản đồ từ file

Cách bản đồ được biểu diễn

Bản đồ được biểu diễn bằng một ma trận 2D chứa các ký tự:

- `P`: Pacman (người chơi)

- `B`, `R`, `N`, `O`: Các ma với màu khác nhau (Blue, Red, Pink/Neon, Orange)

- `H`: Haunted point (điểm ma ám)

- `#`: Tường

- `.`: Điểm thường (dot)

- ` `: Khoảng trống

Graph

Đồ thị được cài đặt trong `map\_implement.py` thông qua lớp `MapGraph`:

### Cấu trúc đồ thị

class MapGraph:

def \_\_init\_\_(self, game\_map):

self.map = game\_map

self.graph = {} # Đồ thị biểu diễn dưới dạng adjacency list

self.haunted\_points = set(game\_map.haunted\_points) # Set các vị trí haunted

self.moves\_since\_haunted = 0 # Số bước đi sau khi qua haunted point

self.build\_graph() # Xây dựng đồ thị

### Xây dựng đồ thị

Đồ thị được xây dựng dựa trên bản đồ trò chơi:

1. Nút (Node): Mỗi nút trong đồ thị là một tuple `(position, direction)` gồm:

- `position`: Vị trí (x, y) trên bản đồ

- `direction`: Hướng di chuyển (UP, DOWN, LEFT, RIGHT)

2. Cạnh (Edge): Mỗi cạnh trong đồ thị biểu diễn một bước di chuyển có thể giữa hai nút với một trọng số:

- Trọng số phản ánh chi phí di chuyển, phụ thuộc vào sự thay đổi hướng

- Hướng di chuyển hiện tại → Thẳng: STRAIGHT (2)

- Rẽ 90°: TURN (10)

- Quay đầu 180°: BACK (20)

- Sau khi đi qua haunted point, tất cả chi phí di chuyển là STRAIGHT trong 10 bước tiếp theo

3. Thuật toán xây dựng đồ thị:

- Quét tất cả các ô trên bản đồ

- Đối với mỗi ô không phải tường, tạo 4 nút tương ứng với 4 hướng di chuyển có thể

- Với mỗi nút, tìm các bước di chuyển hợp lệ và tạo cạnh với các nút khác

- Tính toán trọng số của mỗi cạnh dựa trên sự thay đổi hướng

Tìm đường đi

Các thuật toán tìm đường (UCS, BFS, DFS, A\*) sử dụng đồ thị này để tìm đường đi từ ma đến người chơi:

1. Đầu vào:

- Vị trí hiện tại của ma

- Vị trí đích (người chơi)

2. Đầu ra:

- Đường đi hoàn chỉnh (danh sách các vị trí cần đi qua)

- Tổng chi phí của đường đi

- Vị trí tiếp theo mà ma nên di chuyển đến

3. Mỗi thuật toán tìm đường sử dụng đồ thị theo cách khác nhau:

- UCS: Sử dụng priority queue để luôn mở rộng nút có chi phí thấp nhất

- BFS: Mở rộng tất cả các nút ở độ sâu hiện tại trước khi đi đến độ sâu tiếp theo

- DFS: Mở rộng càng sâu càng tốt trước khi quay lại

- A\*: Tương tự UCS nhưng kết hợp với heuristic (khoảng cách Manhattan) để ưu tiên các nút gần đích

Chi tiết xử lý đặc biệt

1. Haunted Points:

1. **Ưu tiên đường đi qua H**:
   * Thuật toán tìm đường như UCS và A\* sẽ tự nhiên ưu tiên đường đi qua các điểm H vì chúng có thể tiết kiệm chi phí đáng kể, đặc biệt là khi cần thực hiện nhiều lần chuyển hướng
   * Ví dụ: Đường đi thông thường với 5 lần rẽ có chi phí 5×7=35 đơn vị, nhưng nếu đi qua H thì chỉ còn 5×1=5 đơn vị
2. **Giảm thiểu chi phí chuyển hướng**:
   * Sau khi đi qua H, ma có thể thực hiện các chuyển động phức tạp (nhiều lần rẽ, quay đầu) với chi phí thấp
   * Điều này có thể cho phép ma đi theo các đường đi ít tối ưu hơn về mặt khoảng cách nhưng linh hoạt hơn trong việc di chuyển
3. **"Cửa sổ cơ hội"**:
   * Trong 10 bước sau khi qua H, ma có "cửa sổ cơ hội" để thực hiện các di chuyển phức tạp với chi phí thấp
   * Thuật toán sẽ tận dụng tối đa cửa sổ này trước khi chi phí trở lại bình thường

2. Obstacles:

- Chỉ có thể di chuyển qua các ô không phải tường

- Các ô tường (`#`) không được đưa vào đồ thị

3. State Representation:

- Để xác định đường đi chính xác, trạng thái không chỉ bao gồm vị trí mà còn cả hướng di chuyển

- Điều này đảm bảo chi phí chuyển hướng được tính đúng

4. Path Reconstruction:

- Sau khi tìm thấy đường đi, thuật toán tái tạo đường đi từ nút bắt đầu đến nút đích

- Đường đi được chuẩn hóa để chỉ bao gồm các vị trí, không bao gồm thông tin hướng

## **Breadth-first search (BFS)**

1. Tổng quan

Đây là một thuật toán tìm kiếm đồ thị tổng quát. Thuật toán bắt đầu bằng việc tìm

ra node gốc rồi từ đó mở rộng phạm vi tìm kiếm đến các độ sâu tiếp theo cho đến khi tìm được đích đến. Quá trình tìm kiếm được chia thành các lớp: Bắt đầu từ node gốc → Khám phá tất cả các node kề với node gốc → Khám phá tất cả các node kề với các node đã khám phá ở bước trước → Tiếp tục khám phá theo từng lớp cho đến khi tìm thấy được đích đến hoặc đồ thị đã được duyệt hết.

1. Cách hoạt động

Nút chưa mở rộng nông nhất được chọn để mở rộng bằng cách sử dụng hàng đợi FIFO cho biên. Các nút mới (luôn sâu hơn nút cha) sẽ được đưa vào cuối hàng đợi, và các nút cũ (node cha) sẽ được mở rộng trước và rời khỏi hàng đợi trước. Thuật toán sẽ tiếp tục đến khi node đích được đẩy ra khỏi hàng đợi

Gọi độ phức tạp thuật toán để tìm kiếm node gốc là O(b)

→ Độ phức tạp thuật toán: b+b2+b3+…+bd= O(bd) với d: độ sâu của node lá

A close-up of a diagram

AI-generated content may be incorrect.

## **Uniform-cost search (UCS)**

## **Depth-first search (DFS)**

## **A\* search**

# **KIỂM TRA THUẬT TOÁN**

## Các yếu tố quan sát

A black screen with white text

AI-generated content may be incorrect.

Ghost: Vị trí đầu tiên của con ma – Node gốc của thuật toán cần xét

Player: Đích đển của thuật toán tìm đường đi. Trạng thái: bất động

Next Position: Node tiếp theo trên đường đi tìm được

Path: Node nằm trong chuỗi các node mà thuật toán đã tìm được để tới đích P

Wall: Các node không thể đi được

## **BFS Performance**

A screenshot of a computer game

AI-generated content may be incorrect.

## **UCS Performance**

A screenshot of a computer

AI-generated content may be incorrect.

## **DFS Performance**

A screenshot of a computer game

AI-generated content may be incorrect.

## **A\* Search Performance**

A screenshot of a computer game

AI-generated content may be incorrect.

# **BẢNG PHÂN CÔNG CÔNG VIỆC**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **THÀNH VIÊN** | **MSSV** | **NHIỆM VỤ** | **MỨC HOÀN THÀNH** |
| 1 | ĐINH TUẤN DUY | 23127356 | Cài đặt thuật toán BFS, UCS, A\*, DFS.  Cài đặt độ thị thuật toán.  Chỉnh sửa level5, level 6. | 100% |
| 2 | LỮ BẢO ĐẠT | 23127338 | Cài đăt thuật toán A\* Search, DFS  Cài đặt Level 5 | 100% |
| 3 | THIỀU QUANG VINH | 23127143 | Viết báo cáo, quay dựng video, kiểm tra lỗi. |  |
| 4 | BÙI ĐỨC ĐẠT | 23127337 | Kiểm tra Search performance  Cài đặt Level 6 | 100% |

# **TÀI LIỆU THAM KHẢO**

[1] Stuart Russell and Peter Norvig. April 2020. *Artificial Intelligence: A Modern*

*Approach* (4th ed.). Pearson.