TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN

TIỂU LUẬN CUỐI KỲ

Môn học: Trí tuệ nhân tạo

Tên tiểu luận: Thief’s Escape

Giảng viên: Phan Thị Huyền Trang

Danh sách sinh viên thực hiện

Mã số SV	Họ và tên	Mức độ đóng góp (%)
23110220	Ngô Huy Hoàng	100
23110334	Phạm Hữu Thoại	100
23110247	Nguyễn Hoàng Anh Kiệt	100

TP. Hồ Chí Minh, tháng 5 năm 2025

MỤC LỤC

DANH SÁCH HÌNH VẼ

PHẦN 1. MỞ ĐẦU

1.1. Phát biểu bài toán

Thuật toán tìm kiếm là một trong những thuật toán nền tảng và cốt lõi trong lĩnh vực trí tuệ nhân tạo. Nó được ứng dụng rộng rãi trong việc để tìm ra hướng giải quyết vấn đề trong nhiều môi trường khác nhau, phức tạp. Như tìm kiếm không có thông tin, có thông tin hay tìm kiếm có sự ràng buộc...Để củng cố kiến thức đã học, và ứng dụng các thuật toán tìm kiếm một cách linh hoạt. Nhóm đã xây dựng một trò chơi có tên "Thief's Escape..". Cách hoạt động của trò chơi là điều khiển một nhân vật "trộm" di chuyển trên bản đồ để thu thập các vật phẩm và thoát ra khỏi bản đồ thông qua lối thoát (exit) trong khi tránh bị bắt bởi nhân vật người chủ và các bẫy. Trò chơi sử dụng các thuật toán trí tuệ nhân tạođể điều khiển hành vi của trộm và người chủ, kết hợp với cơ chế tầm nhìnđể xác định trạng thái phát hiện giữa hai nhân vật. Với mỗi thuật toán khác nhau có thể đem lại độ hiểu quả khác nhau. Dựa vào kết quả đạt được, nhóm sẽ lập biểu đồ để so sánh, phân tích ưu nhược điểm của từng thuật toán.

Các yếu tố chính của bài toán:

- **Nhân vật trộm**: Di chuyển để thu thập vật phẩm và tìm đường đến lối thoát. Nếu bị phát hiện bởi người chủ nhà sẽ chuyển qua trạng thái rượt đuổi, hoặc khi dính bẫy, trộm sẽ mất máu có thể bị bắt và thua cuộc.
- **Nhân vật người chủ nhà**: Tuần tra trên bản đồ theo các điểm ngẫu nhiên hợp lệ hoặc truy đuổi trộm khi phát hiện. Người chủ nhà có thể lực giới hạn, trong trại thái tuần tra, thể lực sẽ không thay đổi. Khi chuyển sang trạng thái rượt đuổi thể lực bị suy giảm ảnh hưởng đến khả năng truy đuổi.
- **Tầm nhìn**: Mỗi nhân vật có một vùng tầm nhìn hình tròn về mặt đồ họa, thực chất là nhóm ô bản đồ xung quanh nhân vật . Nếu vùng tầm nhìn của người chủ nhà giao với vùng của trộm, trộm sẽ bị phát hiện, dẫn đến trạng thái truy đuổi.
- **Bẫy**: Các bẫy được đặt trên bản đồ, gây sát thương cho trộm nếu va chạm. Được xem như một chi phí ảnh hưởng đến quyết định chọn đường đi của nhân vật khi áp dụng các thuật toán.
- **Bản đồ**: Được tạo từ các file Tiled Map (.tmx), bao gồm tường, đồ nội thất nhằm tạo những rào chắn ngăn cản sự đi lại tự do của nhân vật, vật phẩm được xem là phần thưởng, và lối thoát. Bản đồ được chuyển đổi thành ma trận 2 chiều để dò tìm đường đi hợp lệ cho nhân vật.
- **Thuật toán AI**: Sử dụng các thuật toán tìm kiếm đường đi (BFS, A*, Beam Search, Partial Observation, Q-Learning,) để điều khiển di chuyển của trộm và người chủ nhà.

Kịch bản trò chơi:

- Trộm di chuyển để thu thập vật phẩm khi chưa bị phát hiện, với mỗi thuật toán khác nhau sẽ có đường đi khác nhau, ưu tiên tìm vật phẩm gần nhất.
- Khi bị phát hiện, trộm chuyển sang chế độ chạy thoát tầm nhìn hoặc chạy đến lối thoát.
- Người bảo vệ tuần tra ngẫu nhiên hoặc truy đuổi trộm dựa trên vùng tầm nhìn.

Trò chơi kết thúc khi:

- Trường hợp 1: Thành công, tên trộm thu thập đủ số vật phẩm và trốn thoát thành công.
- Trường hợp 2: Trốn thoát, tên trộm trốn thoát khi bị chủ nhà truy đuổi, mặc dù chưa thu thập đủ số vật phẩm.

- Trường hợp 3: Bị bắt-thua cuộc, tên trạm bị chủ nhà bắt hay hết máu khi dẫm phải quá nhiều bẫy.

1.2. Mục đích và yêu cầu cần thực hiện

Với mục đích là ứng dụng các thuật toán tìm kiếm vào một bài toán cụ thể như tìm đường đi. Dựa vào đó, phát triển một trò chơi 2D mô phỏng quá trình tìm kiếm vật phẩm của tên trộm trong nhiều môi trường khác nhau. Xây dựng một trò chơi thú vị với giao diện đồ họa hấp dẫn, âm thanh sinh động, dễ dàng thao tác với người dùng. Thông qua các dữ liệu thu thập được, nhóm vẽ biểu đồ và phân tích những ưu, nhược điểm để củng cố vững chắc các kiến thức đã được học.

Các yêu cầu cần thực hiện:

Giao diện người dùng:

- Menu chính để chọn thuật toán, bản đồ, và số lần chạy.
- Menu tạm dừng, để bật/tắt âm thanh, quay lại menu chính, hoặc tiếp tục chơi.
- Hiển thị thông tin trạng thái (máu, thể lực, vật phẩm thu thập).

Âm thanh và đồ họa:

- Tích hợp âm thanh hiệu ứng (thu thập vật phẩm, va chạm bẫy, game over, v.v.).
- Tích hợp nhạc nền (menu, chơi game) và chuyển đổi nhạc dựa trên trạng thái (bình thường hoặc truy đuổi).
- Hiển thị hoạt hình (animation) cho nhân vật, vật phẩm, và bẫy.

Xây dựng môi trường trò chơi:

- Tải và hiển thị bản đồ từ file .tmx sử dụng thư viện PyTMX.
- Xử lý va chạm với tường, đồ nội thất, và bẫy.
- Tích hợp cơ chế tầm nhìn để xác định trạng thái phát hiện.

Quản lý trạng thái nhân vật:

- Trộm: Quản lý máu và thu thập vật phẩm.
- Người chủ: Quản lý thể lực, chuyển đổi giữa tuần tra và truy đuổi.

Triển khai thuật toán AI:

- BFS tìm đường đi đến các vật phẩm.
- A*, Beam Search để tìm đường đi tối ưu, có tính đến chi phí bẫy.
- Partial Observation để mô phỏng môi trường quan sát hạn chế, nơi trộm chỉ biết một phần bản đồ.
- Q-Learning để học chiến lược di chuyển, tránh bẫy và tối ưu hóa đường đi.

Lưu trữ và phân tích:

- Lưu kết quả mỗi lần chạy (run) vào file CSV, bao gồm thông tin về thuật toán, bản đồ, thành công/thất bại, độ dài đường đi, thời gian hoàn thành, số nút mở rộng, v.v.
- Hỗ trợ lưu trữ Q-table cho Q-Learning để tái sử dụng.

1.3. Phạm vi và đối tượng

Phạm vi:

- Sử dụng các thuật toán AI đã được học để điều khiển nhân vật trong nhiều môi trường khác nhau và các yếu tố ảnh hưởng đến quyết định đường đi (bẫy, thể lực).
- Hỗ trợ nhiều bản đồ với độ khó tăng dần, độ khó càng tăng qua việc đặt thêm nhiều vách ngăn, đồ vật.
- Trò chơi hoàn toàn tự động, không có tính năng điều khiển thủ công.

Đối tượng:

- Người chơi: Những người yêu thích trò chơi chiến lược, mô phỏng AI, hoặc muốn tìm hiểu về các thuật toán tìm kiếm đường đi.

- Người phát triển: Các lập trình viên quan tâm đến việc áp dụng AI trong trò chơi, đặc biệt là các thuật toán tìm kiếm và học tăng cường.

PHẦN 2. CƠ SỞ LÝ THUYẾT

2.1. Ngôn ngữ lập trình

Dựa trên yêu cầu phát triển dự án về trí tuệ nhân tạo, nhóm đã chọn ngôn ngữ lập trình Python làm ngôn ngữ lập trình chính của đồ án này. Bởi vì, Python có hỗ trợ nhiều thư viện hữu ích cho phát triển trò chơi. Dễ dàng dàng tạo giao diện 2D, xử lý đồ họa, âm thanh, và tương tác người chơi. Một ngôn ngữ mạnh mẽ để phát triển các dự án liên quan đến AI.

2.2. Thư viện và công cụ hỗ trợ

- **Pygame**: Dùng để phát triển trò chơi 2D, xử lý đồ họa, âm thanh, và sự kiện người dùng (như nhấn phím, chuột).
- **PyTMX**: Nhằm tải và xử lý các file bản đồ Tiled (.tmx), cho phép tạo bản đồ phức tạp với các lớp (wall, furniture, objects).
- **NumPy**: Hỗ trợ tính toán ma trận và xử lý dữ liệu trong thuật toán Q-Learning.
- **Pickle**: Lưu trữ và tải Q-table cho Q-Learning.
- **CSV**: Lưu trữ kết quả thống kê của các lần chạy vào file CSV.
- **XML.etree.ElementTree**: Phân tích file .tmx để lấy thông tin đối tượng (furniture objects).
- **Heapq**: Hỗ trợ hàng đợi ưu tiên (priority queue) cho thuật toán A* và Beam Search.
- **Math, Random, Time**: Các thư viện Python chuẩn để tính toán khoảng cách, tạo số ngẫu nhiên, và đo thời gian thực thi.
- **Công cụ thiết kế bản đồ:** Tiled Map Editor, dùng để tạo và chỉnh sửa bản đồ (.tmx).

2.3. Môi trường làm việc

- Hệ điều hành: Windows
- IDE: Visual Studio Code
- Phiên bản Python: Python 3.10

2.4. Thuật toán tìm kiếm trong môi trường cục bộ (Local Beam Search)

Local Beam Search là một thuật toán tìm kiếm heuristic, được thiết kế để tìm đường đi trong không gian trạng thái lớn bằng cách duy trì một tập hợp giới hạn các đường đi tiềm năng (gọi là beam) tại mỗi bước. Đối với dự án, Thuật toán được sử dụng để tìm đường đi cho nhân vật trộm từ vị trí bắt đầu đến mục tiêu, có tính đến các ràng buộc như tường, đồ nội thất, và bẫy.

Giải thích các tham số của thuật toán.

start, goal: tọa độ [x, y] của vị trí bắt đầu và mục tiêu.

grid: ma trận 2 chiều biểu diễn bản đồ (0: đi được, 1: tường).

rows, cols: kích thước của bản đồ.

character_size, furniture_rects, scaled_grid_size, offset_x, offset_y: thông tin để kiểm tra va chạm với đồ nội thất.

traps: Danh sách bẫy với vị trí và loại.

beam_width: Số lượng trạng thái tối đa được giữ trong beam (mặc định 200).

Các bước triển khai:

```
if start is None or goal is None:
    print("Error: Start or goal position is None!")
    return None, 0, 0

adjusted_start = find_nearest_free_position(start, character_size, furniture_rects, grid,
                                            scaled_grid_size, offset_x, offset_y, rows, cols)
if adjusted_start != start:
    print(f"Adjusted start position from {start} to {adjusted_start}")
    start = adjusted_start
```

Hình 1 Triển khai thuật toán beam search

Giải thích:

Kiểm tra vị trí bắt đầu hợp lệ, điều chỉnh bằng find_nearest_free_position nếu cần.

```
beam = [(improved_heuristic(start), 0, [start])]
```

Hình 2 Triển khai thuật toán beam search_tạo trạng thái ban đầu

Giải thích: Tạo beam ban đầu với trạng thái [start] và điểm số heuristic

```
def improved_heuristic(pos):
    manhattan_dist = abs(pos[0] - goal[0]) + abs(pos[1] - goal[1])
    trap_penalty = 0
    if traps:
        for trap in traps:
            trap_pos = trap["pos"]
            trap_type = trap["type"]
            dist_to_trap = abs(pos[0] - trap_pos[0]) + abs(pos[1] - trap_pos[1])
            if dist_to_trap == 0:  # Ô chứa bẫy
                trap_penalty += TRAP_COSTS.get(trap_type, 0) * 1000  # Phạt cực lớn
            elif dist_to_trap < 5:  # Ô gần bẫy
                penalty = TRAP_COSTS.get(trap_type, 0) * 100 / (dist_to_trap + 1)
                trap_penalty += penalty
    return manhattan_dist + trap_penalty
```

Hình 3 Hàm heuristic trong beam search

Giải thích:

- Khoảng cách Manhattan đến mục tiêu.
- Phạt nặng (1000 * chi phí bẫy) nếu ô chứa bẫy.
- Phạt nhẹ (100 * chi phí bẫy / khoảng cách) nếu ô gần bẫy (< 5 ô).

```
for _ in range(min(len(beam), beam_width)):
    if not beam:
        break
    _, cost, path = heapq.heappop(beam)
    x, y = path[-1]
```

Hình 4 Vòng lặp trong beam search

Giải thích: Trong mỗi vòng lặp, lấy tối đa beam_width trạng thái từ beam hiện tại

```
if [x, y] == goal:
    end_time = time.time()
    execution_time = end_time - start_time
    print(f"Beam Search found path: {path}")
    return path, expanded_nodes, execution_time
```

Hình 5 Kiểm tra trạng thái có phải là đích

Giải thích: Nếu đúng là trạng thái đích thì trả về đường đi.

```
for dx, dy in [(0, 1), (1, 0), (0, -1), (-1, 0)]:
    next_x, next_y = x + dx, y + dy
    next_pos = [next_x, next_y]
```

Hình 6 Mở rộng trạng thái bằng hành động di chuyển

Giải thích: Mở rộng trạng thái hiện tại bằng cách thực hiện hành động di chuyển.

```
trap_type = check_trap_collision(next_pos, character_size, traps, scaled_grid_size, offset_x, offset_y)
trap_cost = TRAP_COSTS.get(trap_type, 0) if trap_type else 0
move_cost = 1 + trap_cost * 1000  # Tăng chi phí bẫy
```

Hình 7 Gán giá trị cho các biến

Giải thích: Tính phí di chuyển, mặc định là 1 nếu dính bẫy sẽ tăng thêm hệ số traps_cost*1000

```
if (0 <= next_x < len(grid) and 0 <= next_y < len(grid[0]) and
    grid[next_x][next_y] != 1 and
    not check_furniture_collision(next_pos, character_size, furniture_rects,
                                  scaled_grid_size, offset_x, offset_y) and
    not trap_type and  # Loại bỏ ô có bẫy
    (tuple(next_pos) not in visited or (cost + move_cost) < visited[tuple(next_pos)])):
```

Hình 8 Kiểm tra tính hợp lệ của trạng thái mới

Giải thích: Kiểm tra tính hợp lệ của trạng thái mới:

- Trong giới hạn bản đồ (0 <= next_x < rows, 0 <= next_y < cols).
- Không phải tường (grid[next_x][next_y] != 1).
- Không va chạm đồ nội thất (check_furniture_collision).
- Không chứa bẫy (not trap_type).

```
visited[tuple(next_pos)] = cost + move_cost
new_path = path + [next_pos]
score = cost + move_cost + improved_heuristic(next_pos)
heapq.heappush(next_beam, (score, cost + move_cost, new_path))
expanded_nodes += 1
```

Hình 9 Lưu trữ chi phí và tạo đường đi mới

Giải thích:

- Lưu trữ chi phí tích lũy (cost + move_cost) để đến trạng thái next_pos vào từ điển visited.

- Tạo một đường đi mới (new_path) bằng cách thêm next_pos vào đường đi hiện tại (path).
- Tính điểm số (score) cho trạng thái next_pos để xếp hạng trong beam.
- Thêm trạng thái mới (next_pos) vào next_beam, một hàng đợi ưu tiên (priority queue) lưu trữ các trạng thái tiềm năng cho bước tiếp theo.
- Ứng dụng trong trò chơi
- Điều khiển trộm: Được sử dụng để tìm đường đi cho trộm. Nó phù hợp trong các tình huống cần tốc độ xử lý nhanh, đặc biệt khi trộm cần di chuyển đến vật phẩm hoặc lối thoát trong thời gian thực.
- Tránh bẫy: Với cơ chế phạt bẫy trong heuristic, Beam Search giúp trộm tránh các ô nguy hiểm, phù hợp với yêu cầu của trò chơi là giảm sát thương từ bẫy.

Ưu điểm và nhược điểm

- Ưu điểm :

+ Tiết kiệm bộ nhớ hơn đáng kể so với BFS nhờ beam_width. Việc sử dụng hành đợi ưu tiên sẽ giúp ưu tiên chọn các trạng thái tốt nhất, tập trung vào các đường đi tiềm năng

+ Hàm Heuristic được cải tiến để né bẫy nhờ kết hợp khoản cách Manhattan và phạt chi phí bẫy.

- Nhược điểm:

+ Thuật toán có thể bị kẹt trong các cụ bộ vì chỉ giữ lại các trạng thái có điểm số thấp nhất.

+ Thiếu cơ chế dự phòng khi không tìm được đường đi

PHẦN 3. PHÂN TÍCH, THIẾT KẾ GIẢI PHÁP

3.1. Ý tưởng thuật toán

Xây dựng một trò chơi có thể áp dụng các thuật toán tìm kiếm. Sử dụng hình tượng một tên trộm đang cố đột nhập vào căn nhà lục lọi tìm kiếm đồ vật. Trong đó có những cạm bẫy nguy hiểm rình rập cản trở tên trộm. Mục tiêu đưa ra là giúp tên trộm có thể đạt hoàn thành việc thu thập các vật phẩm và thoát ra an toàn.

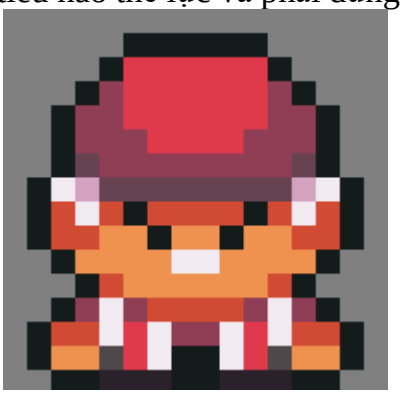
Mô tả: Ban đầu tên trộm xuất phát từ bên ngoài cửa đi vào, mục tiêu là các vật phẩm có giá trị được phân bố xung quanh các căn phòng. Tại thời điểm đó, chủ nhà đang đi tuần tra khắp nơi, cùng với hệ thống bẫy được bố trí buộc tên trộm phải tìm ra giải pháp nhanh nhất để trốn thoát.

Logic trò chơi:

Các thành phần chính

Chủ nhà, có 2 mode di chuyển:

- Tuần tra: random các tọa độ để chủ nhà có những bước di chuyển hợp lí bên trong căn nhà.
- Rượt đuổi: khi phát hiện trộm đột nhập chủ nhà tiến hành rượt đuổi, truy bắt. Ở chế độ này chủ nhà sẽ bị tiêu hao thể lực và phải dừng lại nghỉ ngơi để hồi sức.

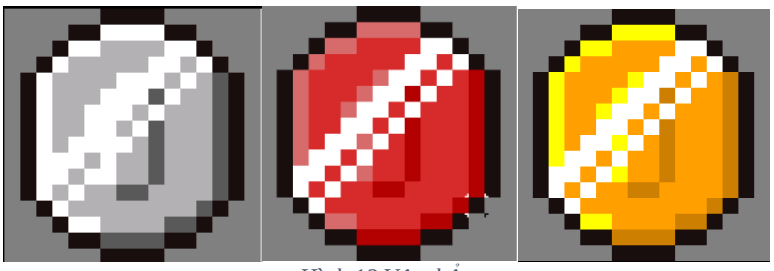


Hình 10 Chủ nhà

Trộm: Ưu tiên nhặt các vật phẩm, sau đó trốn thoát thật nhanh. Khi bị chủ nhà phát hiện thì tiến hành dằn co nếu không được thì chạy trốn. Khi dẫm phải bẫy thì bị mất đi lượng máu nhất định, khi bị chủ nhà bắt thì thất bại.

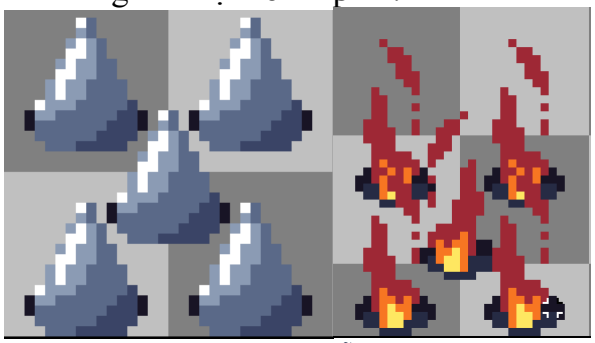
Hình 11 Tên trộm

Vật phẩm: phân bố rải rác trong căn nhà.



Hình 12 Vật phẩm

Bẫy (trap): gây sát thương khi trộm dẫm phải.



Hình 13 Bẫy

Cửa thoát: điểm đến cuối cùng để chiến thắng.
Bối cảnh: nhà và các vật dụng nội thất trang trí đồng thời cũng là vật cản.

Hình 14 Bối cảnh

3.2. Chi tiết về các thuật toán chính đã sử dụng

Danh sách các thuật toán đã sử dụng:

- Nhóm thuật toán tìm kiếm không có thông tin: Breadth-First Search
- Nhóm thuật toán tìm kiếm có thông tin: A* Search
- Nhóm thuật toán tìm kiếm cục bộ: Beam Search
- Nhóm thuật toán tìm kiếm trong môi trường phức tạp: Search with Partial Observation
- Nhóm thuật toán học tăng cường: Q Learning

Breadth-First Search:

- Là một thuật toán tìm kiếm cơ bản trong lý thuyết đồ thị, được sử dụng để duyệt hoặc tìm kiếm trên cây hoặc đồ thị.
- Thuật toán BFS có thể tìm kiếm hiệu quả trong không gian và thời gian tối thiểu. Đồng thời tìm được đường đi ngắn nhất giữa 2 điểm không thông qua chi phí, tức là mỗi bước di chuyển là như nhau. Vì thế, trong bài toán nhân vật có thể đi qua bẫy nếu có bãy xuất hiện trên đường đi, đó cũng là một phần hạn chế.
- Ưu điểm: dễ cài đặt, tìm kiếm đường đi nhanh chóng.
- Chi tiết:

B1: Khởi tạo hàng đợi queue chứa đường đi [start], và tập visited để lưu ô đã thăm.

B2: Lặp cho đến khi queue rỗng:

+ Lấy đường đi đầu tiên khỏi queue → path.
+ Lấy ô cuối của đường đi: (x, y).
+ Nếu (x, y) == goal → trả path.

B3: Mở rộng 4 hướng: phải, xuống, trái, lên.

B4: Với mỗi hướng:

+ Tính tọa độ mới (next_x, next_y).
+ Kiểm tra: nằm trong bản đồ, không phải tường, không va chạm vật, chưa thăm.
+ Nếu hợp lệ, thêm path + [next_pos] vào queue và đánh dấu là đã thăm.

B5: Nếu queue hết → không tìm được đường.

A* Search:

- Thuật toán này sử dụng một "đánh giá heuristic" để xếp loại từng nút theo ước lượng về tuyến đường tốt nhất đi qua nút đó. Thuật toán này duyệt các nút theo thứ tự của đánh giá heuristic này. Do đó, thuật toán A* là một trong những thuật toán tìm kiếm theo lựa chọn tốt nhất (*best-first search*).
- Được áp dụng trong bài toán để tìm đường đi nhanh nhất đồng thời đánh giá được mức độ nguy hiểm của bẫy thông qua chi phí heurictic, vì thế tên trộm có thể né bẫy xuất hiện trên đường đi.
- Ưu diểm: dễ cài đặt, có thể tìm được đường đi tối ưu.
- Chi tiết:

B1: Tính heuristic: h(n) = |x1 - x2| + |y1 - y2|.

B2: Khởi tạo priority queue (heap) với phần tử đầu là (f_cost, g_cost, [start]).

B3: Lặp cho đến khi heap rỗng:

+ Lấy phần tử có f_cost nhỏ nhất.
+ Lấy ô cuối của đường đi → (x, y).
+ Nếu đến goal → trả path.

B4 Mở rộng 4 hướng như BFS.

B5 Với mỗi hướng:

+ Tính chi phí move_cost = 1 + trap_cost nếu có bẫy.
+ Tính new_cost = g + move_cost, f = new_cost + heuristic.
+ Nếu chưa thăm hoặc tìm được đường rẻ hơn → thêm vào heap.

B6 Nếu heap hết → không có đường.

Beam Search:

- Là một thuật toán tìm kiếm theo phương pháp heuristic điều hướng một đồ thị bằng cách mở rộng một cách có hệ thống các nút hứa hẹn nhất trong một tập hợp bị hạn chế.
- Beam Search đặt ra giới hạn về kích thước của biên giới; điều đó khiến nó không đầy đủ và không tối ưu, nhưng nó thường tìm thấy các giải pháp khá tốt và chạy nhanh hơn các tìm kiếm hoàn chỉnh.
- Ưu điểm: sử dụng tài nguyên ít hơn A*
- Chi tiết:

  B1: **Tính improved_heuristic:** khoảng cách + phạt nếu gần bẫy.

  B2: **Khởi tạo** beam (heap) với (heuristic, cost, [start]).

  B3: **Lặp** mỗi vòng:

  + Chỉ duyệt tối đa beam_width phần tử tốt nhất.
  + Với mỗi phần tử:
    - Lấy (x, y) cuối đường đi.
    - Nếu đến goal → trả đường đi.
    - Mở rộng 4 hướng.
    - Bỏ qua ô có bẫy.
    - Nếu hợp lệ → thêm vào next_beam.
  + Cập nhật beam = next_beam.

  B4: Nếu beam rỗng → không tìm được đường.

Search with Partial Observation:

- Trong môi trường nhận thấy không đầy đủ, cảm biến của tác nhân chỉ cung cấp quyền truy cập vào thông tin một phần hoặc không đầy đủ về trạng thái của môi trường tại mỗi thời điểm.
- Tên trộm không có thông tin đầy đủ về toàn bộ bản đồ, vị trí vật phẩm, bẫy, hoặc đối thủ. Nó chỉ quan sát được một phần - ví dụ, các ô trong phạm vi tầm nhìn (vision range).
- Thuật toán chỉ có thểm lặp kế hoạch dựa trên phần tầm nhìn có thể quan sát sau đó tiếp tục phát triển mở rộng ra môi trường xung quanh để tìm kiếm.
- Chi tiết:

  B1: Tạo vùng nhìn thấy từ vị trí thief (hình tròn với bán kính nhất định).

  B2: Cập nhật bản đồ riêng map_thief chỉ gồm các ô nhìn thấy.

  B3: Với mỗi ô nhìn thấy:Nếu đi được và chưa từng ghé → thêm vào queue_visited.

  B4: Nếu thấy item trong vùng nhìn thấy: Dùng A* để tìm đường đến item.

  B5: Nếu không có item:

  Lấy ô đầu tiên trong queue_visited, tìm đường đến đó bằng A*. Nếu không tìm được → đánh dấu đã ghé, bỏ qua.

  B6: Nếu không còn gì trong hàng đợi → không tìm thêm đường nữa.

Q Learning:

- Là thuật toán học tăng cường không cần mô hình được sử dụng để đào tạo các tác nhân (chương trình máy tính) đưa ra quyết định tối ưu bằng cách tương tác với môi trường. Thuật toán này giúp tác nhân khám phá các hành động khác nhau và tìm hiểu hành động nào dẫn đến kết quả tốt hơn. Tác nhân sử dụng

phương pháp thử và sai để xác định hành động nào dẫn đến phần thưởng (kết quả tốt) hoặc hình phạt (kết quả xấu).

- Theo thời gian, thuật toán sẽ cải thiện khả năng ra quyết định của mình bằng cách cập nhật bảng Q, lưu trữ các giá trị Q biểu thị phần thưởng dự kiến khi thực hiện các hành động cụ thể trong các trạng thái nhất định.
- Trong giai đoạn đầu thuật toán có thể khởi tạo bảng Q trống. Một số phiên bản có thể dùng thuật toán có sẵn như BFS để tạo ra một đường đi mẫu giúp khởi động quá trình học nhanh hơn (gọi là *guided initialization*), nhưng sau đó sẽ tự điều chỉnh qua nhiều lần chơi.

Chi tiết:

Giai đoạn 1: Kiểm tra Q-table

1. Nếu đã có Q-table → đánh giá chất lượng (có tạo được đường an toàn không?).
2. Nếu không đạt yêu cầu → huấn luyện thêm.

Giai đoạn 2: Training

1. Với mỗi episode:

+Bắt đầu tại start.

+Lặp đến khi đến goal hoặc quá số bước:

- Tính trạng thái state = (x, y).
- Chọn hành động: ngẫu nhiên (explore) hoặc tốt nhất theo Q-table (exploit).
- Thực hiện hành động → next_state.
- Tính reward (nếu vào bẫy thì âm, gần bẫy thì ít, đến đích thì cao…).
- Cập nhật Q-value:

Q[state][action] = Q[state][action] + α * (reward + γ * max(Q[next_state]) - Q[state][action])

2. Lưu Q-table ra file.

Giai đoạn 3: Dùng Q-table để tìm đường

1. Bắt đầu từ start, mỗi bước chọn action có Q-value cao nhất.
2. Dừng nếu đến goal, rơi vào bẫy, bị lặp hoặc quá số bước.
3. Nếu thất bại → fallback về BFS để bảo đảm có đường.

PHẦN 4. THỰC NGHIỆM, ĐÁNH GIÁ, PHÂN TÍCH KẾT QUẢ

4.1. Thu thập các kết quả thử nghiệm

Để đảm báo tính khách quan, nhóm đã tiến hành chạy từng thuật toán trên 9 map, mỗi map 10 lần và ghi vào file stats.csv

stats.csv > data

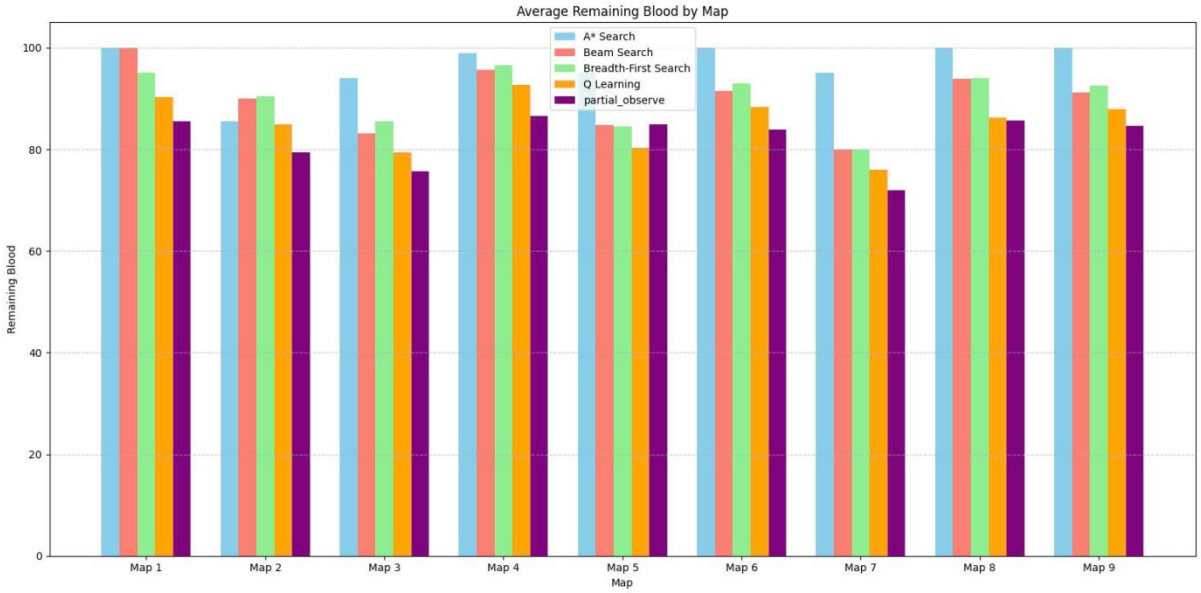
You, 37 minutes ago | 1 author (You)

```
Run,Algorithm,Map,Success,Path_Length,Completion_Time,Expanded_Nodes,Collected_Coins,Remaining_Blood
1,Breadth-First Search,Map 1,1,194.0,19.32042932510376,6,3,95
2,Breadth-First Search,Map 1,1,194.0,19.507269144058228,6,3,95
3,Breadth-First Search,Map 1,0,82.0,8.251069784164429,557,2,95
4,Breadth-First Search,Map 1,1,194.0,19.50899624824524,6,3,95
5,Breadth-First Search,Map 1,1,146.0,14.675755500793455,6,2,95
6,Breadth-First Search,Map 1,1,194.0,19.5100417137146,6,3,95
7,Breadth-First Search,Map 1,0,170.0,17.101802825927734,623,2,95
8,Breadth-First Search,Map 1,1,124.0,12.474438905715942,6,2,95
9,Breadth-First Search,Map 1,1,124.0,12.465887784957886,6,2,95
10,Breadth-First Search,Map 1,0,82.0,8.248355627059937,557,2,95
1,Breadth-First Search,Map 2,1,195.0,19.42934513092041,7,3,85
2,Breadth-First Search,Map 2,1,121.0,12.168896913528442,7,2,95
3,Breadth-First Search,Map 2,1,121.0,12.1644389629364,7,2,95
4,Breadth-First Search,Map 2,1,121.0,12.169084310531616,7,2,95
5,Breadth-First Search,Map 2,0,161.0,16.190863847732544,408,3,90
6,Breadth-First Search,Map 2,1,195.0,19.605574131011963,7,3,85
7,Breadth-First Search,Map 2,0,112.0,11.260071039199827,115,2,95
8,Breadth-First Search,Map 2,1,145.0,14.585403442382812,7,2,95
9,Breadth-First Search,Map 2,1,195.0,19.596811771392822,7,3,85
10,Breadth-First Search,Map 2,1,195.0,19.60308837890625,7,3,85
1,Breadth-First Search,Map 3,1,108.0,10.673293352127075,7,1,90
2,Breadth-First Search,Map 3,1,150.0,15.088952779769896,7,2,80
3,Breadth-First Search,Map 3,1,108.0,10.857571363449097,7,1,90
4,Breadth-First Search,Map 3,1,116.0,11.66598105430603,7,1,90
5,Breadth-First Search,Map 3,1,102.0,10.254326105117798,7,1,90
6,Breadth-First Search,Map 3,1,142.0,14.282713174819946,7,2,80
7,Breadth-First Search,Map 3,1,168.0,16.88669490814209,7,3,80
8,Breadth-First Search,Map 3,1,84.0,8.44611930847168,7,1,95
9,Breadth-First Search,Map 3,1,168.0,16.894033908843994,7,3,80
10,Breadth-First Search,Map 3,1,168.0,16.887824535369873,7,3,80
1,Breadth-First Search,Map 4,1,175.0,17.40503692626953,8,3,95
2,Breadth-First Search,Map 4,1,277.0,27.84297752380371,8,3,90
3,Breadth-First Search,Map 4,1,175.0,17.596896171569824,8,3,95
4,Breadth-First Search,Map 4,1,99.0,9.955679178237917,8,2,100
5,Breadth-First Search,Map 4,1,97.0,9.75803565979004,8,2,100
6,Breadth-First Search,Map 4,1,97.0,9.761669635772703,8,2,100
7,Breadth-First Search,Map 4,0,237.0,23.836925745010376,701,3,90
8,Breadth-First Search,Map 4,1,175.0,17.600999116897583,8,3,95
9,Breadth-First Search,Map 4,1,97.0,9.759149074554443,8,2,100
10,Breadth-First Search,Map 4,1,113.0,11.362082958221436,8,2,100
```

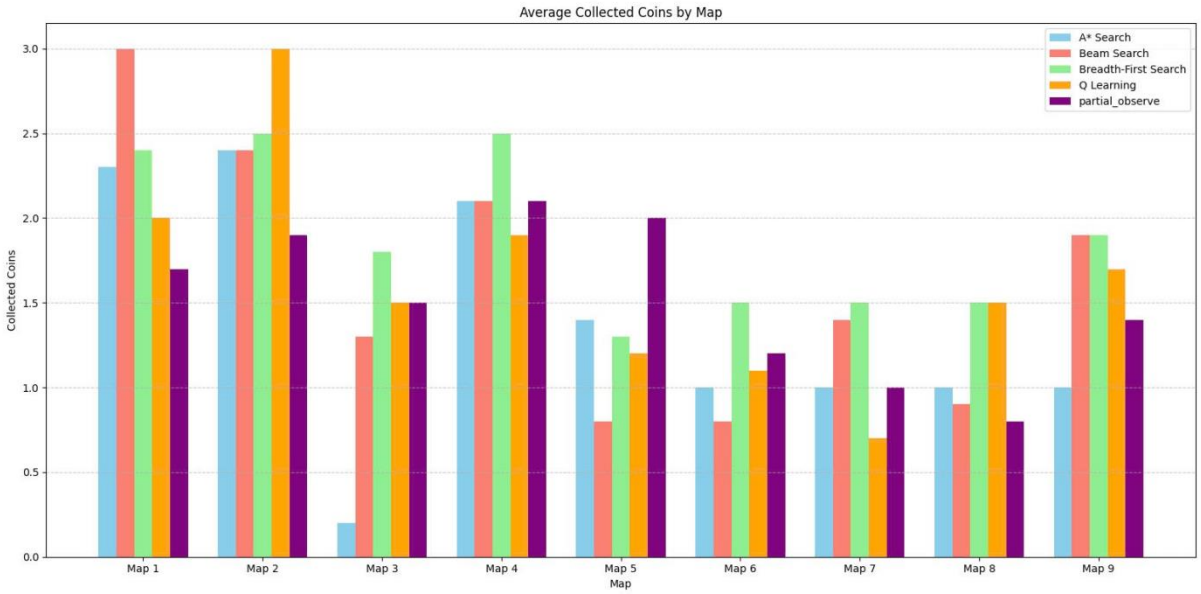
Hình 15 Thu thập kết quả thử nghiệm

4.2. Trình bày các kết quả thử nghiệm

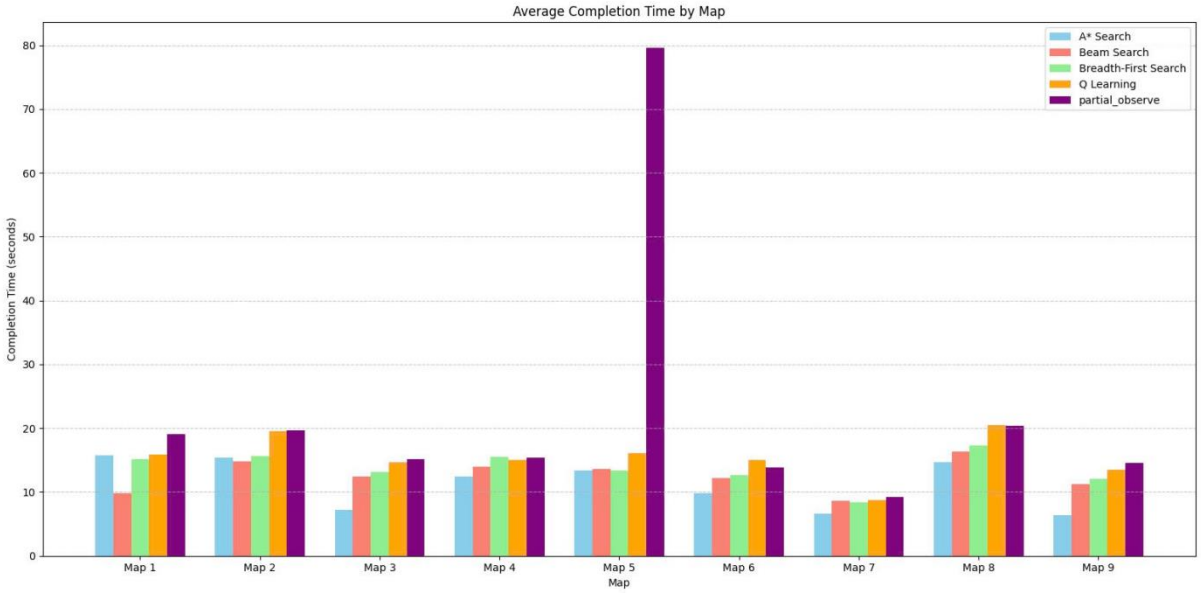
Sau khi thu được file stats.csv hoàn chỉnh, nhóm tiến hành xử lý và vẽ các biêu đồ so sánh giữa các nhóm thuật toán trên nhiều phương diện bằng thư viện matplotlib:



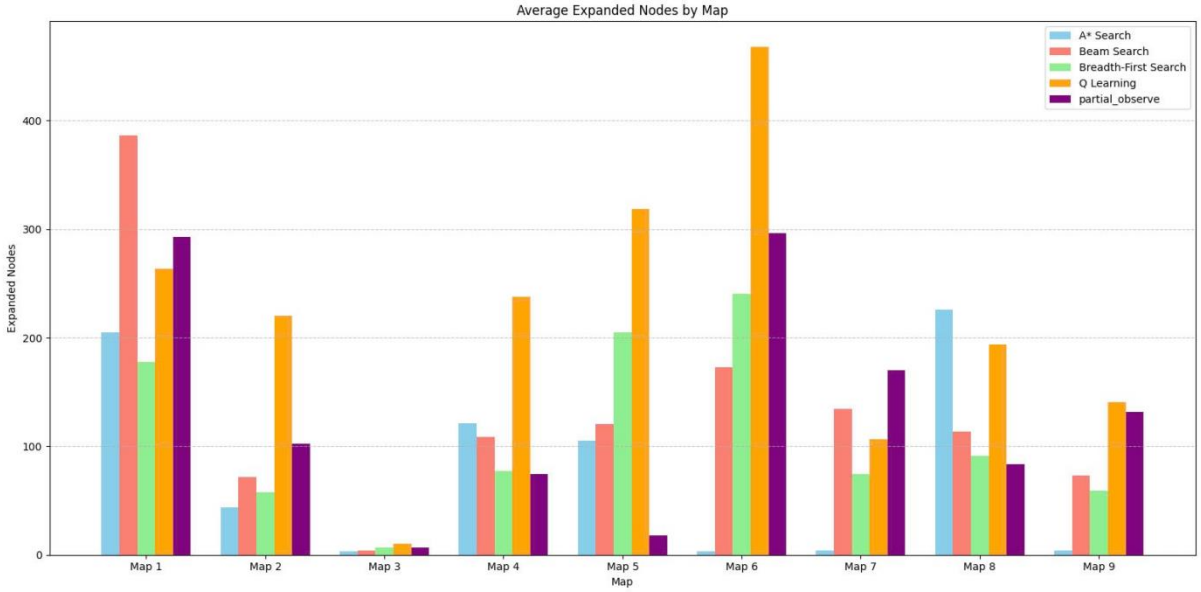
Hình 16 Biểu đồ lượng máu sau thử nghiệm



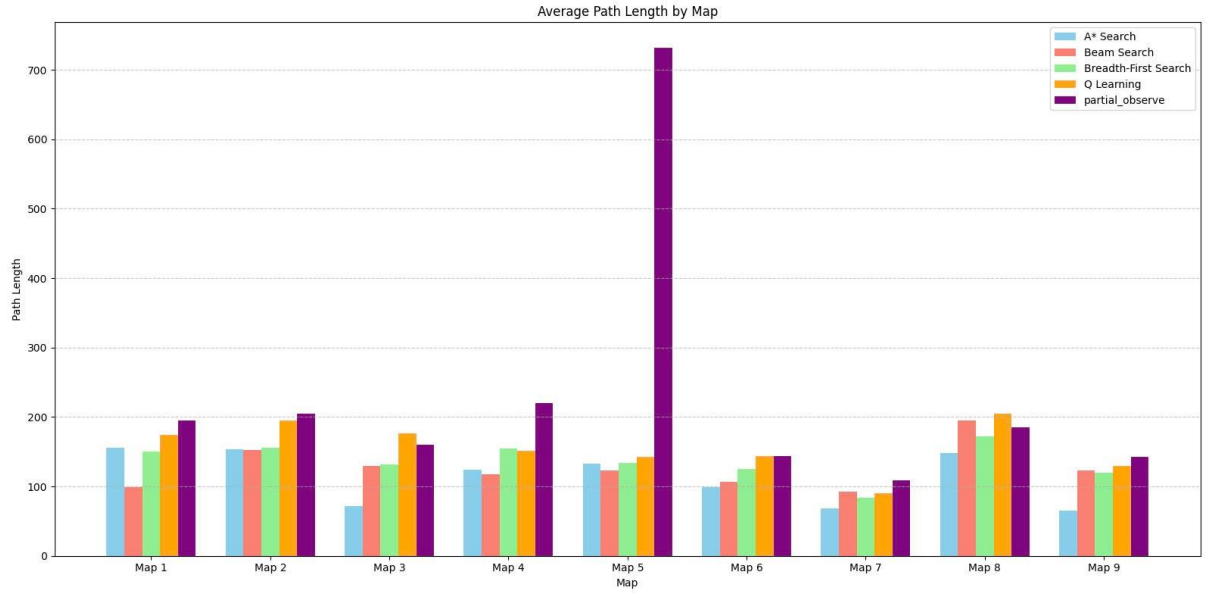
Hình 17 Biểu đồ trung bình số lượng vật phẩm thu thập sau thử nghiệm



Hình 18 Biểu đồ thời gian trung bình hoàn thành thử nghiệm



Hình 19 Biểu đồ trung bình số lượng node mở rộng



Hình 20 Biểu đồ trung bình độ dài di chuyển

- Nhận xét :

+ BFS: Đáng tin cậy, phù hợp khi cần đảm bảo tìm đường, nhưng tốn tài nguyên và không tối ưu.

+ A*: Tốt nhất cho các bản đồ có cấu trúc rõ ràng, tối ưu về đường đi và thời gian, nhưng cần heuristic phù hợp.

+ Beam Search: Nhanh trong một số trường hợp nhưng không đáng tin cậy, phù hợp cho các tình huống cần tốc độ hơn độ chính xác.

+ Partially Observation: Phù hợp cho môi trường có thông tin hạn chế, nhưng hiệu suất kém khi quan sát bị giới hạn nghiêm trọng.

+ Q-Learning: Linh hoạt, thích nghi tốt với môi trường phức tạp, nhưng tốn tài nguyên và không ổn định về đường đi.

4.3. Dẫn link github

https://github.com/finntranne/DoANGameAI

PHẦN 5. PHẦN KẾT LUẬN

5.1. Đánh giá

Đã làm được:

- Xây dựng trò chơi Thief's Escape một cách linh hoạt bằng việc tận dụng nhiều nhóm thuật toán tìm kiếm và học máy để mô phỏng hành động và tương tác của các nhân vật trong game.
- Áp dụng nhiều hành động đa dạng, tuần tra, rượt đuổi, chạy trốn, và nhiều yếu tố đồ vật như bẫy, vật phẩm, trang trí một cách sống động. . Giao diện menu đẹp mắt dễ sử dụng với đa dạng các map và nhiều mức độ từ dễ tới khó.
- Hiểu rõ các thuật toán được học, áp dụng được vào một số vấn đề cụ thể

Chưa làm được:

- Chưa thể thêm nhóm thuật toán ràng buộc vào trò chơi
- Chưa hạn chế tầm nhìn của nhân vật khi chạm vật cản.
- Tên trộm chưa có chiến lược nâng cao như tránh vùng tầm nhìn của chủ nhà, chọn lộ trình an toàn nhất, hay xử lý tình huống bị kẹt.
- Chưa có học tăng cường theo thời gian thực

5.2. Định hướng trong tương lai

- Cải thiện những phần chưa làm được
- Triển khai nhiều nhóm thuật toán AI hơn (Deep Learning, Multi-Agent RL, AI dựa trên thị giác (CNN)…)
- Mở rộng môi trường, quy mô trò chơi, có thể là game FPS giữa người dùng và AI.
- Phát triển công cụ thống kê và huấn luyện, cho phép người dùng xem lại quá trình học, so sánh thuật toán và tối ưu chiến lược.

TÀI LIỆU THAM KHẢO

1. **Artificial Intelligence: A Modern Approach, 3rd ed. (RUSSELL & NORVIG)**
2. **Geeksforgeeks, https://www.geeksforgeeks.org/**