**LỜI MỞ ĐẦU**

Trong xã hội hiện đại ngày nay, Trí Tuệ Nhân Tạo đã gần như là một phần không thể thiếu trong đời sống chúng ta. Trong lĩnh vực y tế: ta có chuẩn đoán bệnh ung thư, viêm phổi,… bằng cách cho AI đọc ảnh MRI, X-Quang,… Phát triển thuốc mới mô phỏng phản ứng sinh học, tìm kiếm ADN mới. Trong lĩnh vực giao thông-vận tải: AI có thể điều khiển xe tự lái bằng các cảm biến tích hợp trên xe ta có thể thấy trên các dòng xe của Tesla,…hay Vinfast của Việt Nam hoặc mới đây cục cảnh sát giao thông Việt Nam cũng đã áp dụng AI xử lý phạt nguội. Thấy rõ ràng nhất là trên các ứng dụng giải trí, thương mại điện tử như: TikTok, Shoppe, Facebook,… ta có hệ thống gợi ý dựa trên thói quen và hành vi người dùng. Khi ta xem một nội dung gì đó nhiều hay tìm kiếm một sản phẩm nhiều lần ta có thể lướt thấy nhiều các nội dung tương tự,…hay hệ thống kiểm duyệt trên các nền tảng có hàng trăm triệu hay tỷ người dùng họ không thể làm bằng tay xem nội dung, sản phẩm nào là độc hại mà phải dựa vào AI để lọc. Và còn nhiều ứng dụng khác của AI.

AI là gì? Có nhiều cách để định nghĩa AI nhưng đơn giản ta có thể hiểu là việc khiến cho máy móc, … suy nghĩ, hành động như con người một cách hợp lý. Đồ án của nhóm em nhằm thực hiện một phần nhỏ của AI trong đời sống, thực tiễn thông qua trò chơi nối thú Pikachu bằng các thuật toán tìm kiếm AI cổ điển để tìm các cặp hình hợp lệ.

Nhóm em xin cảm ơn TS. Phan Thị Huyền Trang đã giúp đỡ nhóm em thực hiện đề tài

**PHẦN MỞ ĐẦU**

1. **Lý do chọn đề tài**

Pikachu là một trò chơi để ăn các cặp hình giống nhau nhưng đòi con người phải tư duy, suy nghĩ để tìm kiếm các cặp sao cho hợp lý phù hợp luật chơi. Các thuật toán tìm kiếm AI cổ điển như: BFS, DFS, A\*,… rất phù hợp để giúp ta hiểu và áp dụng vô trò chơi Pikachu một cách nhanh chóng và chính xác hơn

1. **Mục tiêu và nhiệm vụ thực hiện đề tài**

**Mục tiêu**

Đề tài hướng đến việc ứng dụng các thuật toán tìm kiếm vào trò chơi nối thú Pikachu

Qua đó nhóm mong muốn: Hiểu rõ được nguyên lý hoạt động và sự khác biệt giữa các thuật toán tìm kiếm như: DFS, BFS, A\*,…Xây dựng một chương trình game có giao diện với 2- 3 cấp độ khác nhau

**Nhiệm vụ**

Tìm hiểu các thuật toán tìm kiếm AI đã học, phân tích trò chơi, luật nối thú Pikachu để đưa ra giải thuật phù hợp với trò chơi, thiết kế giao diện trò chơi nối thú Pikachu trong Pygame, so sánh, đánh giá hiệu suất của các giải thuật khi giải trò chơi Pikachu

1. **Phương pháp thực hiện đề tài**

**Nghiên cứu tài liệu**

Tìm hiểu lý thuyết về trí tuệ nhân tạo, các thuật toán tìm kiếm AI cổ điển

Tham khảo về các thư viện cần cài trong chương trình như Pygame và cấu trúc dữ liệu cần thiết

**Phân tích – thiết kế**

Xác định mô hình mà ta cần xây trò chơi là mảng 2 chiều, thiết kế thuật toán phù hợp phù hợp với luật chơi

**PHẦN NỘI DUNG**

**CHƯƠNG 1: TỔNG QUÁT VÀ CƠ SỞ LÝ THUYẾT**

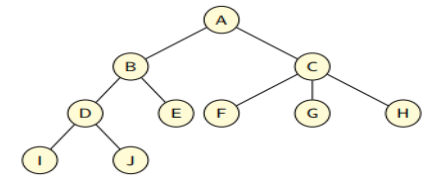
* 1. **Giới thiệu trò chơi Pikachu**

Trò chơi Pikachu là một trò chơi được xuất hiện đầu những năm của thế kỉ XXI dựa trên ý tưởng của việc tìm và nối các cặp hình giống nhau. Trò chơi ngoài mục đích giải trí cũng giúp tăng tính phản xạ, tư duy, quan sát cho người chơi

**Cách chơi**: Người chơi sẽ được cung cấp 1 mảng lưới dạng 2D, trên lưới sẽ có các hình ảnh của các loại pokemon khác nhau, nhiệm vụ của người chơi sẽ tìm ra các cặp hình giống nhau và nối chúng lại với nhau nhưng nhằm bảo đường nối giữa các cặp hình không vướng một hoặc nhiều hình khác, đường nối chỉ được tạo bởi 3 đoạn thẳng ( chỉ được rẽ 2 lần ) cho mỗi cặp hình. Khi nối xong cặp hình vừa nối phải biến mất ra khỏi lưới

* 1. **Khái quát các thuật toán** 
     1. **Thuật toán BFS**

Các cấu trúc dữ liệu cần dùng: Queue (Nguyên tắc FiFo)



Các điểm tròn trên hình lần lượt là các node của cây ví dụ A là một node, B là một Node, C là một Node,…

A là Node gốc, G Node đích

|  |  |  |
| --- | --- | --- |
| **State** | **Queue** | **Visited** |
|  | A |  |
| A | B,C | A |
| B | C,D,E | A,B |
| C | D,E,F,G,H | A,B,C |
| D | E,F,G,H,I,J | A,B,C,D |
| E | F,G,H,I,J | A,B,C,D,E |
| F | G,H,I,J | A,B,C,D,E,F |
| * G là Trạng Thái Đích | H,I,J | A,B,C,D,E,F |

Ta thấy thấy rằng thuật toán sẽ duyệt từ trái sang phải rồi từ trên xuống dưới lần lượt theo thứ tự theo thứ tự không bỏ sót node cho đến khi tìm thấy đích hoặc duyệt hết node tức không tìm thấy đích trong cây

Thuật toán bắt đầu từ node gốc, đưa nó đầu tiên vào hàng đợi(queue hoặc deque), sau đó lấy nó ra khỏi hàng đợi, kiểm tra nó có phải đích không, nếu chưa phải thêm các node con (node con: tức là các node nằm phía dươi có đường nối node đang xét) vào cuối hàng đợi sau đó lấy lần lượt các phần tử trong queue ra xét y chang node gốc lặp lại cho đến khi tìm thấy đích hoặc hàng đợi rỗng ( không tìm thấy đích)

* + 1. **Thuật toán DFS**

Nó cũng giống thuật toán BFS nhưng thay vì sử dụng Queue ta sử dụng Stack(nguyên tắc LiFo)

Tức là thay vì duyệt theo từ trái sang phải rồi từ trên xuống dưới tuần tự nó sẽ duyệt theo từng nhánh trước hết nhánh này mới sang nhánh kia

|  |  |  |
| --- | --- | --- |
| **State** | **Stack** | **Visited** |
|  | A |  |
| A | B,C | A |
| C | B,F,G,H | A,C |
| H | B,F,G | A,C,H |
| * G là trạng thái đích | B,F | A,C,H,G |

Nó cũng bắt đầu với node gốc đưa vào Stack, sau đó lấy ra khỏi stack, nếu chưa phải đích, đưa các node con vào Stack, lấy node mới thêm vào mới nhất ra để xét , lần lượt làm như vậy cho đến khi gặp đích hoặc nếu hết nhánh quay lui sang nhánh khác

* + 1. **Thuật toán Greedy**

Thuật toán dựa dùng cấu trúc dữ liệu priority queue (hàng đợi ưu tiên) tức cũng giống như queue nhưng có thêm 1 chỉ số nữa thay vì cứ vô trước ra trước chỉ số này sẽ chọn ra phần tử nào cần duyệt trước, giống như vô một rạp chiếu phim nếu bình thường không có gì xảy ra thì ta cứ xếp hàng mua vé bình thường người nào đến trước mua trước nhưng rạp chiếu phim ra thêm 1 chỉ số về độ tuổi tức ai là người già trên trên 60 tuổi thì được ưu tiên mua vé trước thì nếu có người già thì dù ta có vô trước hay không cũng phải nhường cho người già

Trong thuật toán Greedy chỉ số để lựa chọn là hàm ước lượng heristic h(n) là hàm ước lượng giá trị tới mục tiêu đích

Để xây hàm h(n) mỗi bài toán lại có một hàm h(n) khác nhau nhưng đối với bài toán liên quan đến ma trận 2 chiều thì tính theo khoảng cách Manhattan hay Euclid

Thuật toán Greedy như BFS nhưng chỉ khác có thêm bước chọn chỉ số nào nhỏ nhất hoặc lớn nhất trong hàng đợi ra để xét nếu bằng nhau thì vô trước ra trước như BFS, thuật toán này sẽ nhanh hơn BFS, tốn ít bộ nhớ hơn nhưng không đảm bảo đường đi tối ưu

* + 1. **Thuật toán A\***

A sao hay A\* là thuật toán tìm kiếm kết hợp giữa hàm h(n) mà ta đã biết với hàm chi phí g(n) hay f(n) = g(n) + h(n) với f(n) là chi phí mà ta dùng để tham số lựa chọn trong priority queue

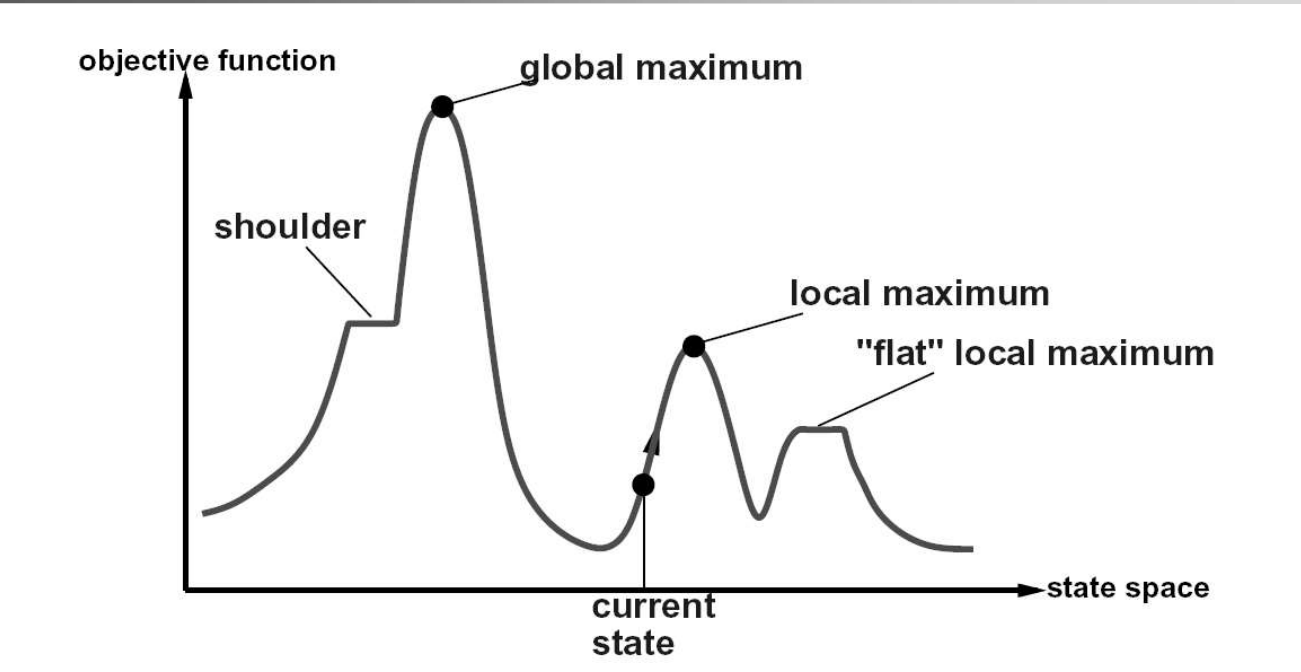
g(n) là hàm chi phí điểm xuất phát tới điểm hiện tại mà ta đang xét ví dụ ta đi du lịch từ A -> C (xuất phát A, đích là C, B là trung gian) quảng đường A -> B dài 150 km, mất khoảng thời gian 3 tiếng, B->C dài 100 km, mất 2 tiếng đi xe thì có thể xem chi phí g(n) tại B là 150 do xét tại A là điểm xuất phát , còn h(n) tại B là 2 tức 2 tiếng đi xe từ B -> C (ước lượng)

Tùy thuộc vào đặt tính bài toán đưa ra ta sẽ có g(n), h(n) khác nhau nên f(n) cũng khác, có thể f(n) càng nhỏ sẽ được chọn cũng có thể f(n) lớn hơn sẽ được chọn. Cách cài đặt dùng cấu trúc dữ liệu priority queue

Thuật toán bắt đầu từ node gốc, đưa nó đầu tiên vào hàng đợi ưu tiên (priority queue), sau đó lấy nó ra khỏi hàng đợi ưu tiên, kiểm tra nó có phải đích không, nếu chưa phải thêm các node con vào cuối hàng đợi sau đó lấy các phần tử có f(n) lớn nhất hoặc bé nhất (tùy bài toán) ra xét trước nếu f(n) bằng nhau thì lấy phần tử vào trước ra xét và xét y chang node gốc lặp lại cho đến khi tìm thấy đích hoặc hàng đợi rỗng ( không tìm thấy đích)

* + 1. **Thuật toán Hill-Clambing**

Hill Climbing là thuật toán tối ưu dựa trên tìm kiếm cục bộ (local search). Nó bắt đầu từ một trạng thái ban đầu, sau đó liên tục di chuyển đến trạng thái lân cận có giá trị heuristic tốt hơn, cho đến khi không thể cải thiện thêm



Tức là nếu tốt thì local maximum có thể là global maximum giá trị ta muốn

đạt nếu không sẽ ra nghiệm cục bộ và ngừng tại đó

* + 1. **Thuật toán Beam-Search**

Beam Search là thuật toán lấy ý tưởng từ BFS nếu khi trước kia thay vì duyệt

theo chiều rộng con nào ta cũng lấy nhưng đối với Beam Search sau khi duyệt hết con, ta sẽ chỉ lấy k con có heristic tốt nhất giữ lại để duyệt điều này khiến bộ nhớ được giảm , có thể nhanh hơn nhưng vẫn sẽ xót lời giải tối ưu

* + 1. **Thuật toán Backtracking**

X is a set of variables, {X1,... ,Xn}.

D is a set of domains, {D1,... ,Dn}, one for each variable.

C is a set of constraints that specify allowable combinations of values.

Đây là nhóm thuật toán CSP tức có ràng buộc mãn điều kiện ta có một

tập các biến, tập các giá trị của từng biến, tập các ràng buộc. Sử dụng đệ quy để làm

Thuật toán lặp đi lặp việc thử từng giá trị của tập giá trị tương ứng của biến Xi, nếu biến Xi đó vi phạm ràng buộc sẽ chọn giá trị khác, nếu thử hết tập giá trị mà biến Xi vẫn không thỏa tập ràng buộc thì quay lui lên Xi-1 để chọn lại giá trị cho biến Xi-1

* + 1. **Thuật toán Forward Checking**

Forward Checking là cải tiến của Backtracking để giảm bớt không gian tìm kiếm

Khi gán giá trị cho một biến, thuật toán sẽ:

Loại bỏ các giá trị không hợp lệ khỏi miền giá trị (domain) của các biến chưa gán. Nếu miền giá trị của một biến trống → dừng và quay lui sớm. So với backtracking nó sẽ giảm số lần quay lui. Hiệu quả cao hơn Backtracking thông thường

* 1. **Khái quát về các thư viện cần dùng**
     1. **Pygame**

Pygame là một thư viện mã nguồn mở, đa số dùng để thiết kế game 2D, có các hàm tiện lợi để xử lý các thao tác chuột, âm thanh, hình ảnh, video,…

Để sử dụng ta cần pygame, sau khi cài ta có thể sử dụng bằng cách gọi: import pygame

Trong đề tài Pygame được sử dụng để làm giao diện Background, các nút

bấm, giao diện ma trận 2D (lưới) để các ô chứa pikachu, hình ảnh các Pokemon,…

Các hàm mà trong đề tài thường dùng:

Pygame.display.set\_mode((width,height)) # để hiển thị màn hình chính chưa nội dung cần hiển thị với hai tham số chiều rộng , chiều cao tính bằng pixel

pygame.display.update() hoặc pygame.display.flip() # Cập nhật nội dung hiển thị lên màn hình.

pygame.Surface((w, h)) # để tạo 1 vùng vẽ (panel) riêng biệt

pygame.image.load("ten\_anh.png") # để load ảnh

surface.blit(image, (x, y)) # vẽ ảnh lên panel

pygame.font.SysFont("Arial", size) # tạo ra chữ

font.render("Nội dung", True, color) # Tạo hình ảnh văn bản lên màn hình

* + 1. **NumPy**

Đây là thư viện để xử lý tính toán số học và xử lý mảng một hay

nhiều chiều với hiệu suất, xử lý cao cho các hàm

Các hàm dùng trong đề tài:

np.arange(a, b) # sinh ma trận từ a đến b-1

np.zeros((m,n)) # sinh ra ma trận toàn 0 từ m hàng , n cột

* + 1. **Random**

Random là thư viện chuẩn của Python, dùng để sinh số ngẫu nhiên, chọn

ngẫu nhiên phần tử, hoặc trộn dữ liệu

Các hàm dùng trong đề tài:

random.shuffle(list) # Tạo ra 1 danh sách khác danh sách gốc với các giá trị được xáo trộn

**CHƯƠNG 2: THUẬT TOÁN BFS VÀ DFS TRONG**

**TRÒ CHƠI NỐI THÚ PIKACHU**

**CHƯƠNG 3: THUẬT TOÁN GREEDY VÀ A SAO TRONG**

**TRÒ CHƠI NỐI THÚ PIKACHU**

**CHƯƠNG 4: THUẬT TOÁN HILL-CLAMBING VÀ**

**BEAM-SEARCH TRONG TRÒ CHƠI NỐI THÚ PIKACHU**

**CHƯƠNG 5: THUẬT TOÁN BACKTRACKING VÀ FORWARD-CHECKING TRONG TRÒ CHƠI NỐI THÚ PIKACHU**