Thuật toán tìm đường thường sử dụng trong game.

Tổng hợp các thuật toán tìm đường thường được sử dụng trong phát triển game.

Tổng quan

Trong phát triển game thuật toán tìm đường được ứng dụng rất phổ biến từ các game AAA cho đến cả những game puzzle trên mobile.

Như cái tên của nó. Thuật toán tìm đường giúp tìm ra con đường từ điểm A tới điểm B tránh khỏi các chướng ngại vật với một lộ trình tối ưu nhất.

Ví dụ trong game moba. Thuật toán tìm đường giúp nhân vật tự động di chuyển đến điểm người chơi đã chọn, giúp creep chạy về phía nhà chính đối phương và né khỏi các chướng ngại vật trên đường đi như trụ, nhà lính, tường. Trong pikachu, thuật toán tìm đường được sử dụng để để kiểu tra 2 con pikachu có match được với nhau không.

Các thu?t toán tìm ???ng th??ng g?p.

	Dijkstra	**A***	**BFS (Breadth-first search)**
Định nghĩa	Tìm đường đi **với chi phí thấp nhất** bằng cách duyệt qua tất cả các node trên bản đồ và tính toán khoảng cách ngắn nhất từ gốc đến mỗi node.	Tìm đường đi **tối ưu nhất** (khoảng cách và chi phí) dựa trên hằng số heuristic để dự đoán khoảng cách từ node đến đích.	Thuật toán **tìm kiếm theo chiều rộng**. Tìm kiếm theo từng lớp bắt đầu từ node gốc sau đó chuyển sang các lớp kế tiếp.
Ưu điểm	Đảm bảo tìm ra đường đi ngắn nhất.	- Tìm đường nhanh, chính xác - Hiệu quả với bản đồ lớn.	- Đơn giản, dễ triển khai. - Hiệu quả với bản đồ không có trọng số
Nhược điểm	Tốn kém (bộ nhớ và CPU) với bản đồ lớn.	Phụ thuộc vào hàm heuristic (nếu hàm không tốt có thể gây tốn tài nguyên).	- Không tối ưu cho tìm đường đi ngắn nhất - Tốc độ chậm với các bản đồ lớn.

Ghi chú:

- Hàm heuristic là một hàm ước lượng khoảng cách từ một node dến đích.
- Bản đồ ở đây là một tập hợp các node và các cạnh nối giữa chúng.

Các yếu tố ảnh hưởng đến lựa chọn thuật toán

Lo?i game

Thể loại game ảnh hưởng lớn đến việc lựa chọn thuật toán. Ví dụ trong một game nhập vai thế giới mở với môi trường rộng lớn và phức tạp thì thuật toán A* hoặc Jump Point Search có thể là lựa chọn tốt hơn Dijkstra. Trong khi với một game puzzle thì BFS có thể hiệu quả hơn.

Môi tr??ng trong game

Cấu trúc của môi trường trong game cũng là một yếu tố quan trọng. Nếu môi trường game là một lưới đơn giản (grid-based) thì Jump Point Search có thể là lựa chọn tối ưu. Nếu môi trường phức tạp bao gồm cả địa hình thì NavMesh có thể là lựa chọn.

Hi?u n?ng

Thuật toán tìm đường sẽ ảnh hưởng đến hiệu năng của game. Đặc biệt với những game có nhiều đối tượng cần tìm đường cùng lúc. Cần cân nhắc giữa độ chính xác của thuật toán và tốc độ tính toán để đảm bảo game có thể chạy chính xác và mượt mà.

S? l??ng nhân v?t

Như đã nói ở trên thì hiệu năng bị ảnh hưởng khi có nhiều nhân vật cùng tìm đường. Thế nên cần cân nhắc lựa chọn thuật toán phù hợp với số lượng nhân vật trong game.

Các y?u t? khác

Ngoài ra còn cần xem xét đến nhiều yếu tố khác như khả năng di chuyển của nhân vật, loại chướng ngại vật, yêu cầu về tính linh hoạt của đường đi...

Ví d?

Trong game chiến thuật thời gian thực (RTS) với hàng trăm đơn vị quân di chuyển trên bản đồ, việc sử dụng thuật toán Dijkstra có thể gây ra hiện tượng giật lag do yêu cầu tính toán lớn. Trong trường hợp này A* hoặc Jump Point Search là lựa chọn phù hợp hơn.

Thuật toán A*

Định nghĩa

Thuật toán A* là một thuật toán tìm kiếm đường đi ngắn nhất giữa 2 điểm, sử dụng hàm heuristic để ước lượng khoảng cách từ một node đến đích. Kết hợp ưu điểm của thuật toán Dijkstra và thuật toán tìm kiếm tham lam.

Các khái niệm cơ bản

Node: Mỗi vị trí trên bản đồ được coi là một node. Ví dụ trên một grid 10x10 thì mỗi ô được coi là một node.

Cạnh (Edge): Đường nối giữa 2 node được gọi là cạnh. Cạnh thể hiện mối liên hệ giữa các vị trí ví dụ như đường đi giữa 2 ô trên bản đồ.

Chi phí đường đi (G cost): Chi phí để di chuyển từ điểm node ban đầu đến node hiện tại. Giá trị này có thể được tính dựa trên khoảng cách thực tế, thời gian di chuyển hoặc các yếu tố khác. G cost được tính bằng tổng G cost của các step trước đó.

Hàm heuristic (H cost): Hàm ước lượng khoảng cách từ node hiện tại đến node đích. hàm heuristic đóng vai trò quan trọng trong việc hướng dẫn A* tìm đường đi hiệu quả. Có nhiều loại hàm heuristic khác nhau (Khoảng cách Manhattan, khoảng cách Euclid hoặc khoảng cách đường chim bay).

Tổng chi phí ước tính (F cost): Tổng chi phí ước tính để di chuyển từ node bắt đầu đến node đích đi qua điểm hiện tại. **F = G + H.**

Hàm heuristic (H cost)

??nh ngh?a

Hàm heuristic là hàm ước lượng khoảng cách từ một nút bất kỳ đến nút đích. Nó đưa ra một **dự đoán** về chi phí tối thiểu để di chuyển từ điểm node hiện tại đến node đích mặc dù không biết chính xác đường đi thực tế.

Vai trò

Hàm heuristic có vai trò như một "la bàn" chỉ dẫn thuật toán A* tìm đường đi hiệu quả hơn. Bằng cách ước lượng khoảng cách đến đích, nó giúp thuật toán ưu tiên khám phá các node có khả năng nằm trên đường đi ngắn nhất, tránh lãng phí thời gian tìm kiếm những hướng không hiệu quả.

Các lo?i hàm heuristic th??ng g?p

1. **Khoảng cách Manhattan**
   - **Định nghĩa**: tính tổng số bước di chuyển theo chiều ngang và chiều dọc giữa 2 node
   - **Công thức**: **$H = Abs(x_1 - x_2) + Abs(y_1 - y_2)$**
     - x_1, x_2 là toạ độ node hiện tại.
     - y_1, y_2 là toạ độ node đích.
   - **Ưu điểm**: Đơn giản, dễ tính toán, phù hợp với bản đồ dạng lưới. Nơi chỉ có thể di chuyển theo 4 hướng (lên xuống trái phải).
   - **Nhược điểm**: Không chính xác khi có thể di chuyển theo đường chéo hoặc trong môi trường phức tạp.
2. **Khoảng cách Euclid**
   - **Định nghĩa**: tính khoảng cách đường thẳng giữa 2 node.
   - **Công thức**: **$H = Sqrt((x_1 - x_2)_2 + (y_1 - y_2)_2)$**
     - x_1, x_2 là toạ độ node hiện tại.
     - y_1, y_2 là toạ độ node đích.
   - **Ưu điểm**: chính xác hơn khoảng cách Manhattan khi có thể di chuyển theo mọi hướng.
   - **Nhược điểm**: tốn kém hơn về mặt tính toán.
3. **Khoảng cách đường chim bay.**
   - **Định nghĩa**: Ước lượng khoảng cách thực tế giữa 2 điểm trên bản đồ tính đến các chướng ngại vật.
   - **Ưu điểm**: Chính xác nhất trong các loại hàm heuristic, phù hợp với các ứng dụng thực tế như định vị GPS.
   - **Nhược điểm**: Khó tính toán, thường yêu cầu các thuật toán phức tạp để xác định đường đi tránh các chướng ngại vật.
4. **Hàm heuristic đường chéo**
   - **Định nghĩa**: là một biến thể của khoảng cách Manhattan, cho phép di chuyển theo đường chéo với chi phí lớn hơn di chuyển theo chiều ngang hoặc chiều dọc.
   - **Ưu điểm**: Đa dụng hơn khoảng cách Manhattan khi có thể di chuyển theo chiều dọc.
   - **Nhược điểm**: Không chính xác bằng khoảng cách Euclid trong môi trường không gian 3 chiều.

Lưu ý: Việc lựa chọn hàm heuristic phù hợp có thể ảnh hưởng đáng kể đến hiệu suất của thuật toán A*. Một hàm heuristic tốt sẽ giúp A* tìm ra đường đi ngắn nhất một cách nhanh chóng và hiệu quả.

Tính ch?t c?a hàm heuristic

- **Tính chấp nhận được (Admissible)**: Hàm heuristic không được đánh giá quá cao chi phí thực tế đến đích. Nó phải luôn đưa ra ước lượng thấp hơn hoặc bằng khoảng cách thực tế.

- **Tính nhất quán (Consistent)**: Chi phí ước tính để di chuyển từ A đến B cộng với chi phí ước tính từ B đến đích phải nhỏ hơn hoặc bằng chi phí ước tính từ A đến đích.

?nh h??ng ??n thu?t toán A*

- **Hàm heuristic tốt:** Có thể hướng dẫn A* tìm kiếm hiệu quả, nhanh chóng tìm ra đường đi ngắn nhất.
- **Hàm heuristic xấu**: Có thể dẫn đến A* duyệt qua nhiều node không cần thiết làm giảm hiệu quả của thuật toán.
- **Hàm heuristic = 0**: A* sẽ hoạt động giống Dijkstra, tìm kiếm tất cả các đường đi có thể.

Cách thức hoạt động

A* hoạt động bằng cách duyệt qua các node trên bản đồ, bắt đầu từ node bắt đầu và kết thúc tại node đích. Thuật toán sử dụng một danh sách các node mở (open list) và một danh sách node đóng (closed list) để theo dõi quá trình tìm kiếm.

Kh?i t?o

Thêm node bắt đầu vào open list.

Tìm ki?m

- Lấy node có F cost thấp nhất từ open list. Gọi node này là node hiện tại.
- Chuyển node hiện tại từ open list sang closed list.
- Xét tất cả các node kề với node hiện tại.
- Nếu node đã có trong closed list hoặc là chướng ngại vật thì **bỏ qua**
- Nếu node chưa có trong open list hoặc có G cost thấp hơn giá trị G Cost hiện tại của nó thì:
  - Cập nhật G cost của node kề.
  - Tính toán H cost và F cost của node kề.
  - Thêm node kề vào open list.
  - Lưu node hiện tại là node cha của node kề.
- Lặp lại các bước trên cho đến khi open list rỗng hoặc tìm thấy node đích.

Xây d?ng ???ng ?i.

Nếu tìm thấy node đích. Truy ngược từ node đích về node đầu bằng cách sử dụng thông tin node cha để xây dựng đường đi ngắn nhất.

Ưu và nhược điểm của thuật toán A*

?u ?i?m

- **Hiệu quả**: A* thường tìm ra đường đi ngắn nhất một cách nhanh chóng và hiệu quả, đặc biệt là khi sử dụng hàm heuristic tốt. Nó kết hợp ưu điểm của cả thuật toán Dijkstra (thuật toán tìm kiếm chi phí thấp nhất) và thuật toán tìm kiếm tham lam (ước lượng khoảng cách đến đích).
- **Linh hoạt**: A* có thể sử dụng cho nhiều bài toán tìm đường khác nhau. Từ game đến robot di động. Nhờ khả năng sử dụng hàm heuristic khác nhau để thích ứng với từng môi trường cụ thể.
- **Tối ưu**: Trong nhiều trường hợp, A* đảm bảo tìm ra đường đi ngắn nhất, miễn là hàm heuristic thoả mãn các điều kiện về tính chấp nhận được và tính nhất quán.

Nh??c ?i?m

- **Độ phức tạp**: A* có thể trở nên phức tạp khi xử lý các bài toán không gian tìm kiếm lớn hoặc hàm heuristic phức tạp.
- **Bộ nhớ**: A* có thể tốn nhiều bộ nhớ, đặc biệt là khi phải lưu trữ một lượng lớn các node trong open list và closed list.
- **Hiệu năng**: Hiệu quả của A* phụ thuộc rất nhiều vào chất lượng của hàm heuristic. Nếu hàm heuristic không được thiết kế tốt A* có thể kém hiệu quả hơn các thuật toán khác.