**Tên đề tài:**

**A\* Pathfinding in Game**

**Danh sác sinh viên thực hiện:**

**1642065 - Phạm Tiến Thành**

**1642037 - Trần Quang Lộc**

**1642001 - Trần Trung An**

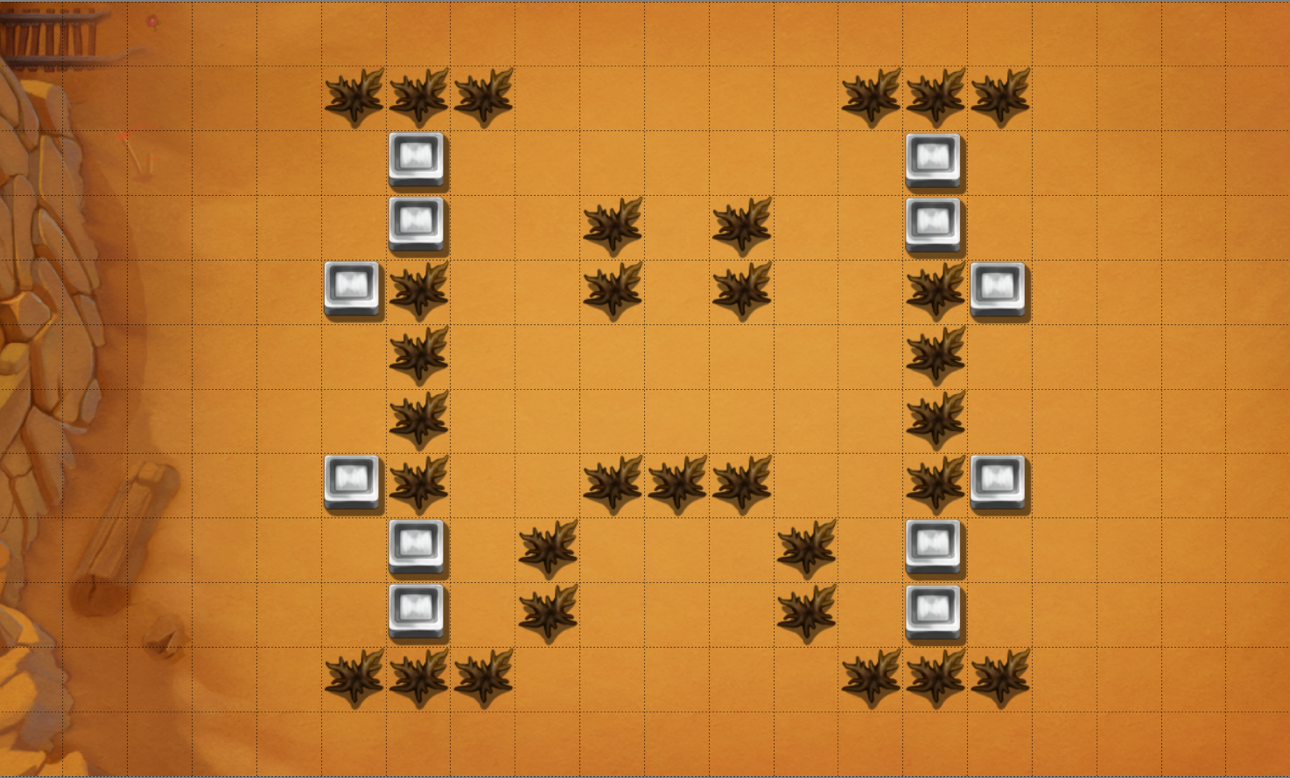
**1642066 - Nguyễn Hữu Thạnh**

**1542285 – Nguyễn Xuân Trường**

# Giới thiệu

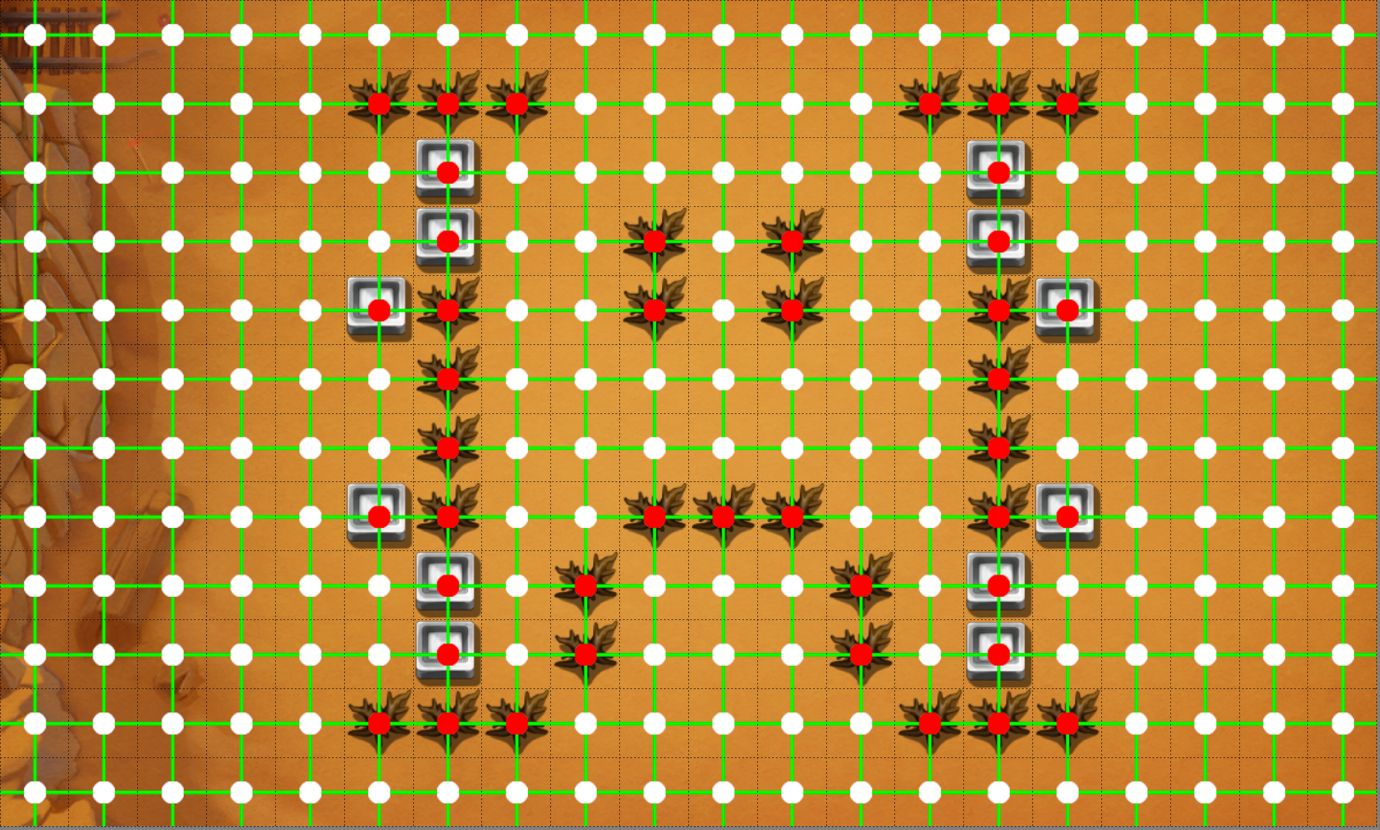
Trong môn lý thuyết đồ thị chúng ta đã học các thuật toán tìm đường trên đồ thị dựa vào tập đỉnh, cạnh và kết nối giữa chúng. Vậy làm thế nào để ứng dụng những thuật toán này trong game ?

Bản đồ game



Mỗi bản đồ trong trò chơi thường được biểu diễn dưới dạng lưới 2 chiều. Nên nó cũng được xem là một đồ thị với mỗi ô là đỉnh và đường ngăn giữa 2 ô là cạnh với trọng số là 1.

Biểu diễn bản đồ game dưới dạng đồ thị



Như vậy chúng ta hoàn toàn có thể áp dụng các thuật toán tìm đường trên đồ thị cho bản đồ game.

Các thuật toán thường được dùng để tìm đường:

* Dijkstra’s & Breadth First Search
* Best First Search
* A\* ( được sử dụng phổ biến nhất trong game)

# Nội dung

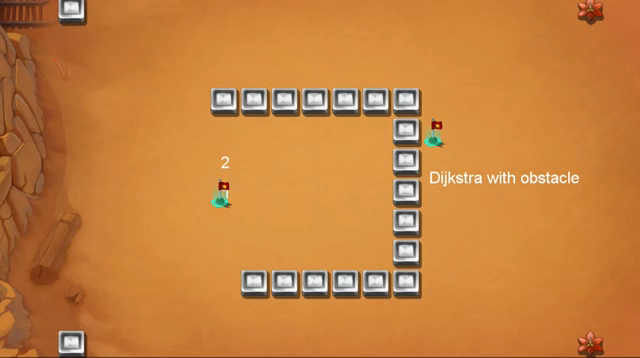
## Thuật toán Dijkstra’s và Thuật toán Breath First Search

Thuật toán Dijkstra’s hoạt động bằng cách duyệt từ đỉnh đầu và thêm các đỉnh gần nhất chưa được duyệt vào một **hàng đợi ưu tiên**\* dựa vào trọng số. Việc duyệt lặp lại cho tới khi gặp đỉnh đích. Thuật toán Dijkstra’s luôn đảm bảo tìm được một đường đi ngắn nhất từ điểm đầu đến điểm đích miễn tồn tại một đường đi giữa chúng. Minh họa thuật toán Dijkstra’s:

Tương tự như Dijkstra’s thuật toán Breadth First Search cũng duyệt bằng các mở rộng về mọi hướng bằng cách thêm các đỉnh vào **hàng đợi** nên chi phí và thời gian tìm kiếm gần như nhau.

## Thuật toán Best First Search

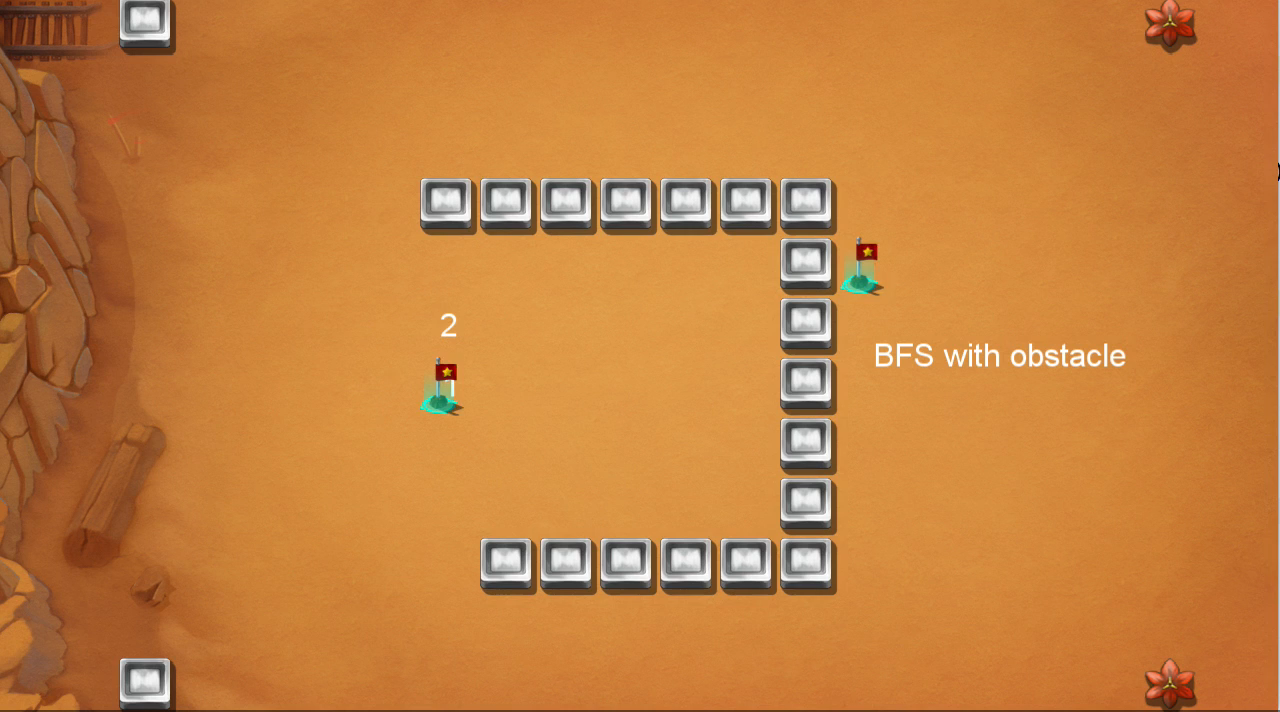
Thuật toán Best First Search cũng hoạt động tương tự nhưng với mỗi đỉnh được duyệt sẽ dựa trên một hàm ước lượng h bằng khoảng cách đến đích. Best First Search chọn đỉnh gần nhất với đỉnh đích thay vì chọn đỉnh gần nhất với đỉnh bắt đầu. Thuật toán Best First Search không đảm bảo tìm thấy đường đi ngắn nhất nhưng luôn tìm được đường đi nhanh hơn thuật toán Dijkstra’s do có hàm h luôn hướng về đỉnh gần với đích. Minh họa:

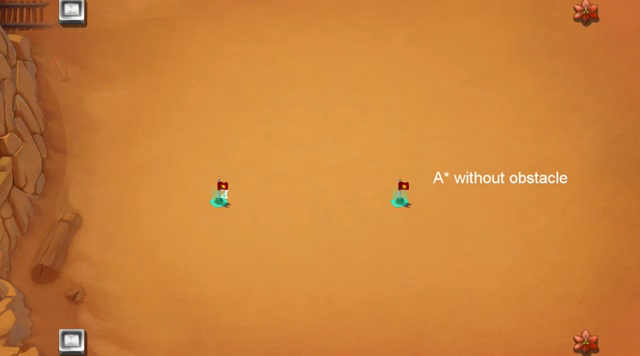


Chúng ta thấy các thuật toán trên hoạt động khá hiệu quả trên bản đồ đơn giản và không có vật cản. Hãy xét trường hợp bản đồ phức tạp hơn.

Thuật toán Dijkstra’s:

Thuật toán Dijkstra’s tốn nhiều chi phí hơn nhưng vẫn trả về đường đi ngắn nhất.

Thuật toán Best First Search:



Thuật toán Best First Search tốn ít chi phí hơn nhưng đường đi không đảm bảo ngắn nhất.

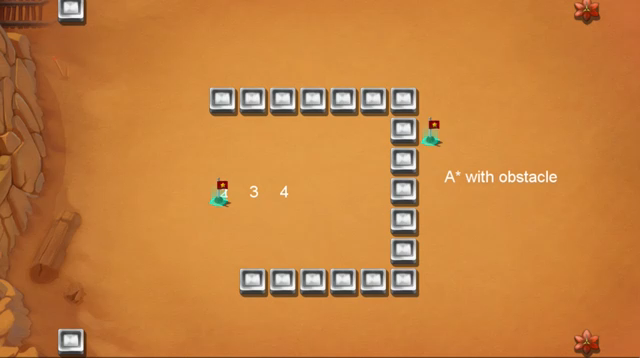
Vậy làm thế nào vừa tìm được đường đi ngắn nhất như Dijkstra vừa đỡ tốn chi phí như Best First Search ?

## Thuật toán A\*

Thuật toán A\* được phát triển năm 1968 bởi Peter Hart, Nils Nilsson, và Bertram Raphael đã giải quyết được vấn đề trên.

Thuật toán A\* giống như thuật toán Dijkstra’s, nó luôn tìm được đường đi ngắn nhất nếu tồn tại một đường như thế. A\* cũng giống như Best First Search vì nó sử dụng hàm heuristic để chọn đường đi tối ưu.

Trong trường hợp bản đồ không có vật cản A\* nhanh như Best First Search:

Trong trường hợp bản đồ có vật cản A\* tốn ít chi phí hơn Dijkstra’s nhưng vẫn trả về đường ngắn nhất.

A\* ước lượng không chỉ dựa trên khoảng cách tới điểm đích mà còn dựa trên khoảng cách đã đi qua. Điều đó làm cho A\* "đầy đủ" và "tối ưu", nghĩa là, A\* sẽ luôn luôn tìm thấy đường đi ngắn nhất nếu tồn tại một đường đi như thế.

Mô tả thuật toán

A\* lưu giữ một tập các lời giải chưa hoàn chỉnh, nghĩa là các đường đi qua đồ thị, bắt đầu từ nút xuất phát. Tập lời giải này được lưu trong một hàng đợi ưu tiên (priority queue). Thứ tự ưu tiên gán cho một đường đi x được quyết định bởi hàm f(x) = g(x) + h(x).

Trong đó g(x) là chi phí đường đi đến hiện tại, h(x) là khoảng cách từ điểm hiện tại tới đích.

Hàm f(x) càng thấp thì x có độ ưu tiên càng cao.

Mô tả thuật toán:

Initialize the open list

initialize the closed list

put the starting node on the open list (you can leave its f at zero)

while the open list is not empty

find the node with the least f on the open list, call it "q"

pop q off the open list

generate q's 8 successors and set their parents to q

for each successor

if successor is the goal, stop the search

successor.g = q.g + distance between successor and q

successor.h = distance from goal to successor

successor.f = successor.g + successor.h

if a node with the same position as successor is in the OPEN list \

which has a lower f than successor, skip this successor

if a node with the same position as successor is in the CLOSED list \

which has a lower f than successor, skip this successor

otherwise, add the node to the open list

end

push q on the closed list

end

## Ứng dụng thực tế

# Tóm tắt

Qua đề tài này chúng ta có thể thấy A\* mang đầy đủ các ưu điểm của các thuật toán tìm đường đi trên đồ thị nên nó được ứng dụng rộng rãi đặc biệt trong phát triển game.

# Tài liệu tham khảo

* <http://web.mit.edu/eranki/www/tutorials/search/>

# Mã nguồn và công cụ

* Cocos2d-x : <http://cocos2d-x.org/>
* Tiled Map Editor : <http://www.mapeditor.org/>
* Mã nguồn: <https://github.com/tienthanh1993/DemoAStar/tree/full>
* Thuật toán: <https://github.com/tienthanh1993/DemoAStar/blob/full/Classes/rpg/PathFinder.cpp#L127>