BÀI TOÁN MÊ CUNG

BỘ MÔN: TRÍ TUỆ NHÂN TẠO

Thành viên nhóm:
Nguyễn Thiện Hỷ 20162111
Nguyễn Văn Huy 20161844

I Mô tả bài toán

Mê cung : hệ thống ma trận hai chiều chứa các ô vuông

- Ô bắt đầu
- Ô kết thúc
- Ô lối đi
- Ô tường

Nhiệm vụ: Từ điểm bắt đầu phải tìm được lối đi tới điểm kết thúc với điều kiện:

- Chỉ được đi trên những ô lối đi , ô tường là những ô không thể đi.
- Chỉ có thể đi thẳng, rẽ trái,phải hoặc lui sau .

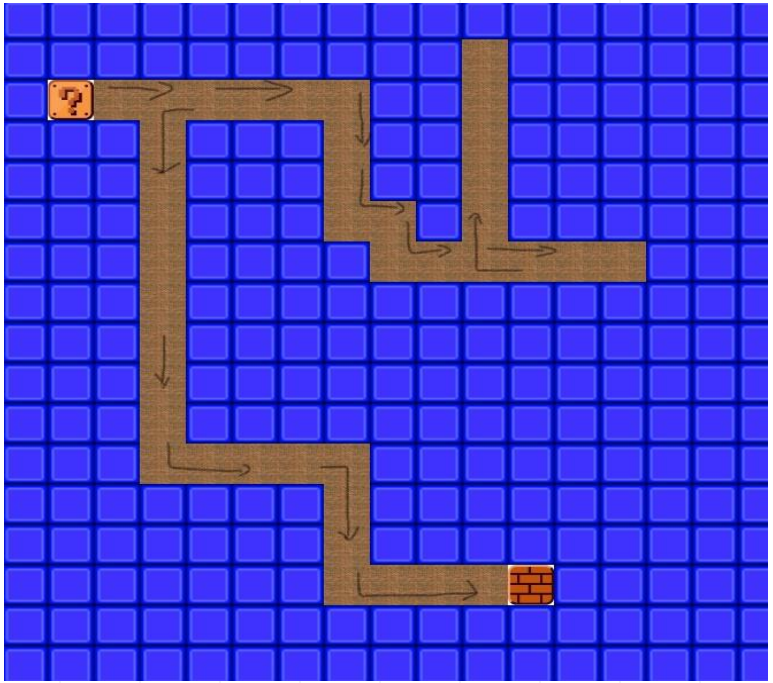
II Các giải pháp

Có rất nhiều thuật giải được đặt ra cho bài toán mê cung như:
-Thuật toán bám tường(đi men theo bờ tường trái hoặc phải)
-Lấp kín đường cụt(lấp kín các lối đi dẫn đến ngõ cụt)
-Duyệt chiều sâu(ưu tiên duyệt tận cùng một lối đi nhất định để tìm đích nếu cụt quay lại duyệt tiếp các lối còn lại)
-Duyệt chiều rộng(ưu tiên loang rộng hết mức có thể để tìm đích)
-Thuật toán A*(sự kết hợp của thuật toán duyệt chiều rộng và hàm lượng giá heuristic)
…

III Chi tiết thuật toán trong bài làm

III.1 Thuật toán DFS

- Thuật toán tìm kiếm xa nhất có thể theo mỗi nhánh (có đánh dấu trên các lối đã đi) cho đến khi gặp được đích hoặc ngõ cụt.
- Nếu là ngõ cụt sẽ quay về ô trước đó để xét các nhánh mà chưa duyệt.



Mã giả :

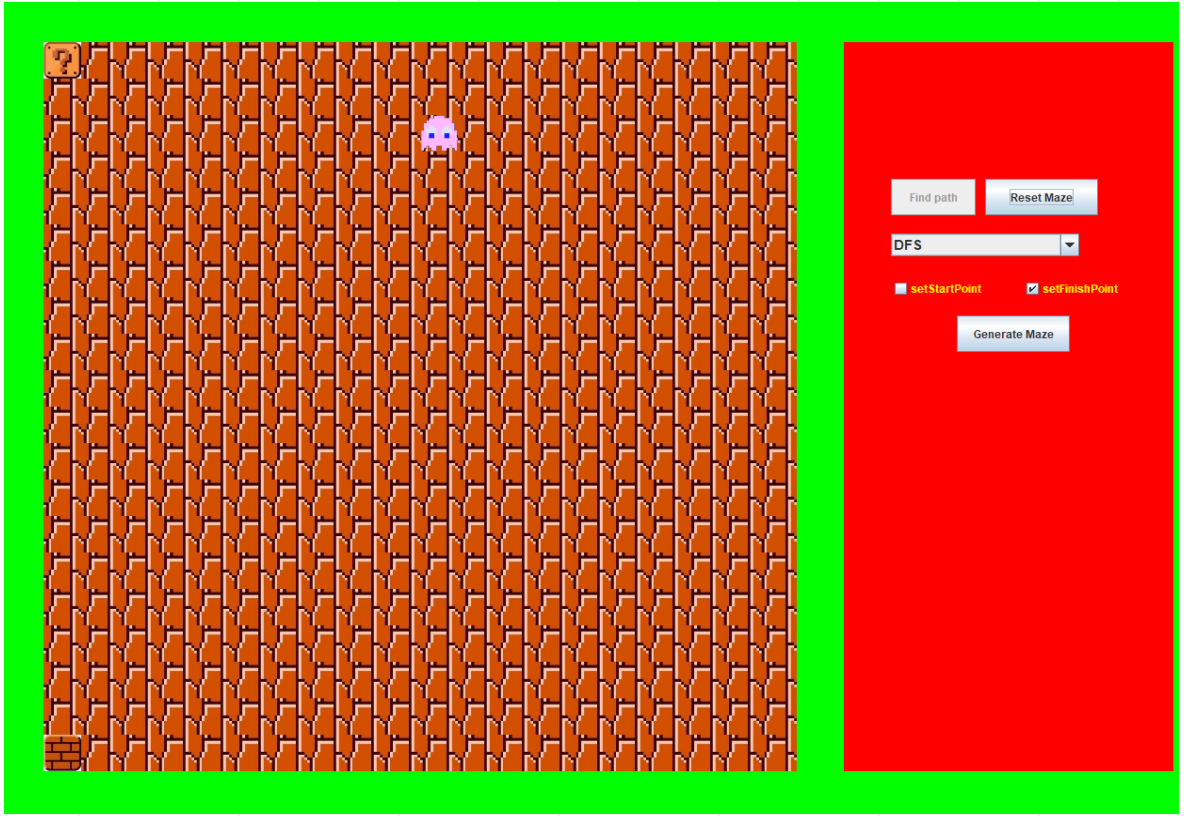
```
Function DFS (x){
        Đánh dấu ô x đã duyệt
        stack St.push(x);
        if(x là đích) then{
                truy vết ngược lại tìm lối đi
                return }
        if(x là lối cụt) St.pop();
        foreach( ô v kề với ô x là lối đi chưa được duyệt){
                v.parent =x
                DFS(v)
        }
}
```

- Mỗi ô sẽ được gọi đệ quy tối đa 1 lần nếu đó là lối đi vì có đánh nhãn các ô đã duyệt, nên không vướng phải sự chồng chéo.
- Trong trường hợp tồi tệ nhất với một mê cung kích thước m*n nó sẽ duyệt qua tất cả các ô là lối đi

⇒ nếu mê cung chỉ gồm lối đi thì nó phải duyệt qua tất cả mê cung

Độ phức tạp thuật toán: O(m*n)

Dùng cấu trúc stack để lưu trữ kết quả lối đi O(m*n)



❖ Đánh giá thuật toán

-Độ phức tạp thuật toán O(m*n) kích thước mê cung

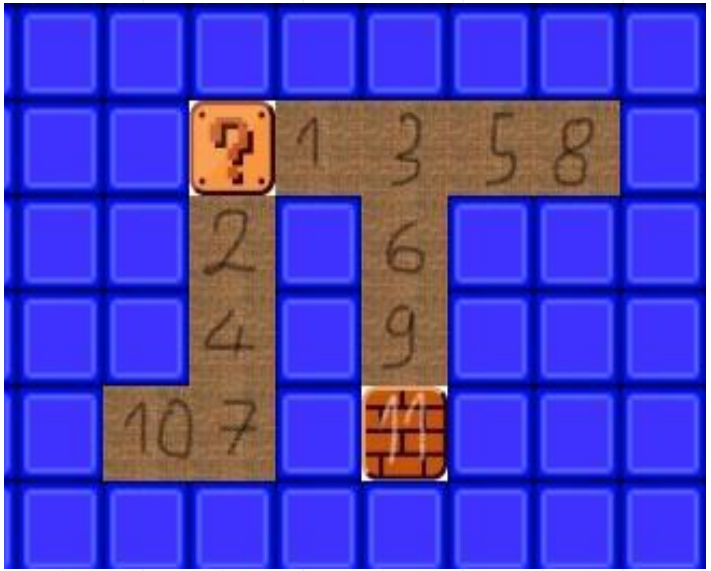
-Tính hoàn chỉnh (tìm thấy lời giải) : luôn luôn có thể tìm được lời giải nếu mê cung có đường đi tới đích tuy nhiên không hẳn là lời giải tốt nhất.

- Thời gian tìm thấy lời giải phụ thuộc vào kích thước mê cung, và độ phức tạp mê cung , chiều duyệt ưu tiên.

- Thuật toán không tối ưu(không đảm bảo tìm được tuyến đường có chi phí thấp nhất).

III.2 Thuật toán duyệt chiều rộng(BFS)

- Thuật toán bao gồm thao tác là kiểm tra đỉnh hiện tại, thêm các đỉnh kề với điểm hiện tại vào hàng đợi.
- Đánh dấu các đỉnh đã duyệt và lấy các đỉnh từ hàng đợi để duyệt.



Mã giả:

```
Function BFS(x){
        if(x là đích) return
        foreach(ô v kề với x chưa xét){
                v.parent =x
                queue Q.push(v)
        }
        if(Q.empty()) then return
        y=Q.front()
        Q.pop()
        BFS(y)
}
```

Tương tự với trường hợp xấu nhất của dfs thì độ phức tạp bfs cũng như thế.
Tuy nhiên độ phức tạp không gian bộ nhớ thì bfs thực sự không tốt vì sử dụng queue để lưu trữ các ô lân cận nên có độ phức tạp O(n^3)

❖ Đánh giá thuật toán

- Tương tự với trường hợp xấu nhất của dfs thì độ phức tạp bfs cũng như thế.
- Tuy nhiên độ phức tạp không gian bộ nhớ thì bfs thực sự không tốt vì sử dụng queue để lưu trữ các ô lân cận nên có độ phức tạp O(n^3)

- Độ phức tạp thuật toán O(n*m)
- Tính hoàn chỉnh : Hoàn toàn có thể tìm thấy lời giải, lời giải còn có thể là tốt nhất.
- Thời gian tìm thấy lời giải phụ thuộc vào mê cung, cũng như độ sâu của đích so với điểm xuất phát (độ sâu càng lớn thời gian duyệt càng lâu).
- Thuật toán là tối ưu nếu chi phí cho mỗi bước đi là 1.

III.3 Thuật toán A*

- Là thuật toán tìm kiếm 1 đường đi từ nút khởi đầu tới nút kết thúc trong đồ thị.
- Thuật toán sử dụng hàm đánh giá heuristic để ước lượng về tuyến đường tốt nhất có thể khi đi qua nút đó.
- Ý tưởng ban đầu của thuật toán: bắt nguồn từ việc tối ưu thuật toán Dijkstra để xây dựng được một tập các đường đi được sắp xếp tăng dần theo chi phí.
- Hàm heuristic là hàm đánh giá thô , giá trị của hàm phụ thuộc vào trạng thái hiện tại của bài toán tại mỗi bước giải.

$\Rightarrow$ Ta thu được tuyến đường có lời giải tốt nhất.

Mô tả thuật toán

Độ ưu tiên đường đi được xét bằng chi phí x qua hàm f(x)=g(x)+h(x). Chi phí càng nhỏ độ ưu tiên càng cao.
-f(x):hàm ước lượng đường đi khi đi qua nút hiện tại.
-g(x): chi phí quãng đường khi đi từ điểm xuất phát tới điểm hiện tại.
-h(x): ước lượng chi phí từ điểm hiện tại tới đích.

-Tạo hai list open (chứa các điểm chuẩn bị xét)
Close (chứa điểm đã xét).
-Đưa điểm xuất phát vào open làm rỗng close
- while(open !=rỗng){
điểm hiện tại = điểm có f min trong open
if(điểm hiện tại trùng điểm kết thúc) truy vấn lại đường đi và kết thúc.
xóa điểm hiện tại khỏi open và đưa vào close
đưa các điểm kề với điểm hiện tại mà không có trong close vào open
sắp xếp lại list open theo hàm heuristic f(x) (sắp xếp tăng dần).
}

Lựa chọn hàm heuristic

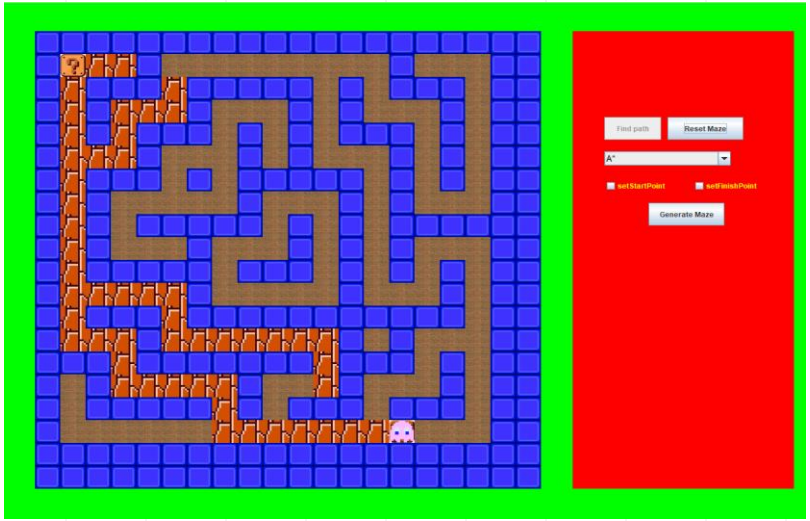
Độ phức tạp thuật toán hoàn toàn phụ thuộc vào đánh giá heuristic
Trong trường hợp tồi tệ nhất thì số điểm được mở rộng thêm sẽ là hàm mũ .
Để xác định được hàm heuristic chúng ta cần ước lượng được chi phí từ hai ô trong mê cung (ví dụ như khoảng cách Euclid,…) sao cho chi phí thực tế của đường đi không nhỏ hơn hàm ước lượng hay nói cách khác 0<h(n)<=h*(n).
Hàm Heuristic càng lớn sẽ càng tốt nếu thỏa mãn điều kiện trên.
Qua đó chọn hàm cho

h(n)=(|xđích-xcurrent|+|yđích-ycurrent|)

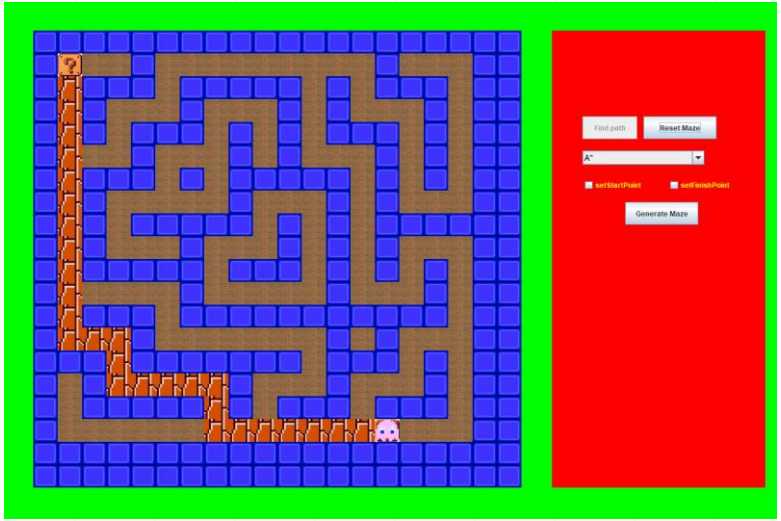
Hàm heuristic : f(n)=g(n)+ (|xĐích-xCurrent|+|yĐích-yCurrent|)

MỘT SỐ VÍ DỤ CHỨNG MINH ƯỚC LƯỢNG ƯU THẾ

f(n)=g(n)+√((xđích-xcurrent)^2+ (yđích-ycurrent)^2)

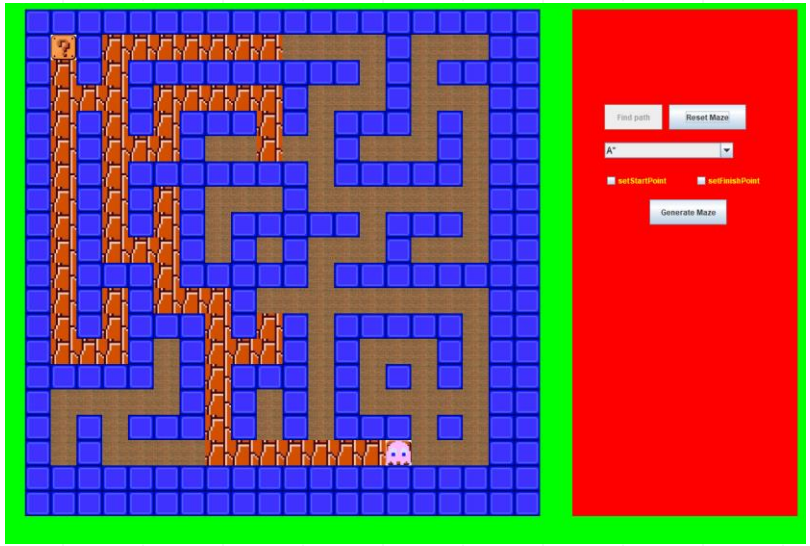


f(n)=g(n)+|xđích-xcurrent|+|yđích-ycurrent|

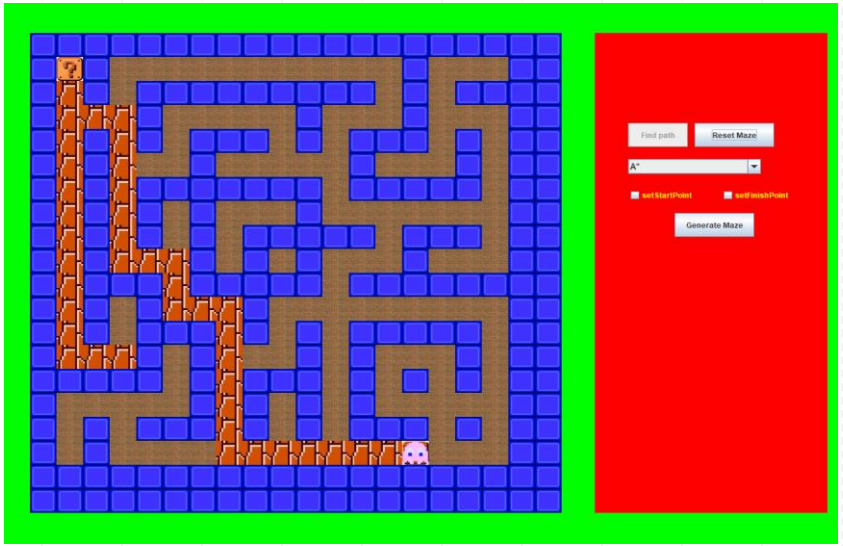


MỘT SỐ VÍ DỤ CHỨNG MINH ƯỚC LƯỢNG ƯU THẾ

f(n)=g(n)+√((xđích-xcurrent)^2+ (yđích-ycurrent)^2)

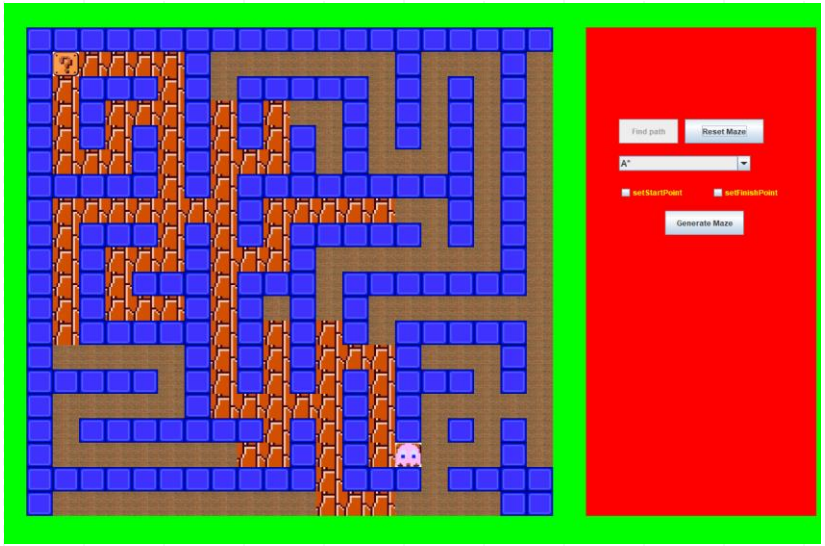


f(n)=g(n)+|xđích-xcurrent|+|yđích-ycurrent|

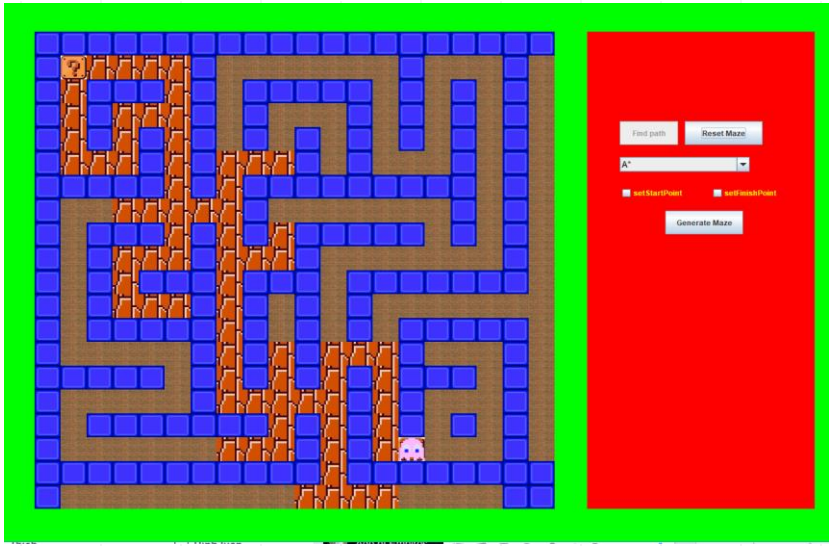


MỘT SỐ VÍ DỤ CHỨNG MINH ƯỚC LƯỢNG ƯU THẾ

f(n)=g(n)+√((xđích-xcurrent)^2+ (yđích-ycurrent)^2)



f(n)=g(n)+|xđích-xcurrent|+|yđích-ycurrent|



Nhận xét: Tuy nhiên, cũng như tất cả các thuật toán tìm kiếm có thông tin , nó chỉ xây dựng các tuyến đường "có vẻ" dẫn về phía đích. Với bài toán mê cung thường lối đi sẽ vòng từ ngoài vào đích, khiến cho khả năng duyệt từ điểm gần đích trước của A* gây mất thời gian hơn.

IV So sánh các thuật toán

Thuật toán	Tính khả giải	Tính thực tế	Phụ thuộc	Bộ nhớ	Lời giải
DFS	Có	Có	Người thực hiện	Nhiều	Không tốt
BFS	Có	Mất thời gian	Mê cung	Ít	Tốt
A*	Có	Không thể thực hiện trong trường hợp không biết trước mê cung	Mê cung	Nhiều với hàm f(x) tồi và ngược lại	Tốt

THANKS!