ĐẠI HỌC QUỐC GIA TP HCM

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

ĐỒ ÁN LÝ THUYẾT ĐỒ THỊ

"ĐƯỜNG ĐI PACMAN"

SINH VIÊN THỰC HIỆN

1. *1542031 – Phan Huy Hoàng*
2. *1542050 – Bùi Văn Kim*
3. *1542098 – Trần Huyền Trân*

LỚP: 15HCB1

NĂM HỌC: 2015-2016

MỤC LỤC

LÝ THUYẾT ĐỒ THỊ

"Đường đi Pacman"

I. LÝ DO CHỌN ĐỀ TÀI

Trong thực tế, chúng ta thường hay gặp rất nhiều câu hỏi, bài toán về việc tìm đường đi. Và trên thực tế, nếu lựa chọn theo ước lượng của con người thì kết quả dường như không chính xác và mang tính chủ quan. Sự ra đời của nhiều công cụ giúp xác định và tìm vị trí (Google maps, Apple maps, Here maps, …) đường đi đã mang lại rất nhiều hiệu quả.

Các công cụ này khá tốt trong việc xác định đường đi giữa 2 vị trí bất kỳ, nó có khả năng giới thiệu và gợi ý cho người dùng khá nhiều con đường khác nhau. Tuy nhiên việc xác định con đường ngắn nhất giữa **n** vị trí đích khác nhau thì không phải chương trình nào cũng làm tốt.

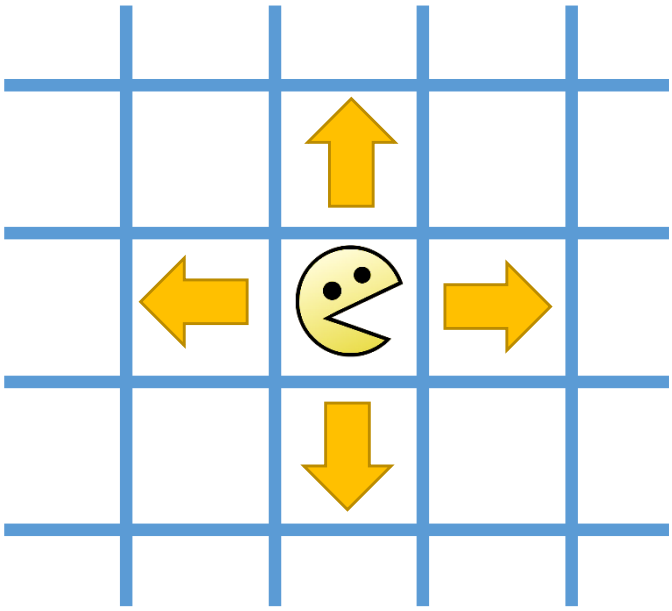
Với đề tài **Đường đi Pacman** nhóm sẽ mang lại 1 phương pháp *(Chiến lược)* để giải quyết bài toán tìm đường đi có nhiều vị trí đích khác nhau bằng cách dựa trên các thuật toán đã học. Cụ thể là thuật toán **Dijsktra** và **A***.

II. BÀI TOÁN PACMAN

Pacman là một trò chơi xuất hiện vào khoảng những năm 80 của thế kỷ 20 tại Nhật Bản, là một trò chơi **arcade** được phát triển bởi **Namco**. Pacman Trở nên nổi tiếng và được ưa thích ngay từ khi được phát hành cho đến ngày nay, Pacman được xem là một trò chơi kinh điển và trở thành một biểu tượng của văn hóa đại chúng những năm 80.

Tuy nhiên trong đề tài này, Pacman không phải là một trò chơi. Nhóm đã chuyển nó thành bài toán tìm đường đi.

Pacman là một nhân vật (Player) được đặt trong bản đồ, và có hình thức như sau:



Dựa vào hình trên, chúng ta nhận thấy:

- Bản đồ đường đi Pacman được thể hiện dưới dạng một bàn cờ caro.
- Nhân vật Pacman sẽ được phép di chuyển theo 4 hướng (Lên – Xuống – Trái – Phải)

III. LỰA CHỌN THUẬT TOÁN

Việc tìm đường đi chúng ta có thể áp dụng nhiều thuật toán khác nhau như: *BFS, DFS, UCS, Dijsktra, A**...

Tuy nhiên ở trường hợp bài toán đường đi Pacman. Chúng ta đang xét trên bản đồ có trọng số, và mục tiêu là tìm đường đi ngắn nhất. Vì thế nhóm cân nhắc việc sử dụng hàm UCS (Thuật toán tìm kiếm có chi phí), Dijsktra hoặc A*… thì sẽ đáp ứng được yêu cầu đề ra.

Trong đề tài này, nhóm quyết định cài đặt và sử dụng 2 thuật toán tìm đường đi, đó là Dijsktra và A*. Đồng thời với thuật toán A* sẽ xây dựng trên nhiều *Heuristic* khác nhau, nhằm đánh giá được tầm quan trọng của chi phí ước lượng (*heuristic*) trong việc tìm kiếm.

1. *Thuật toán Dijsktra:*

Đây là thuật toán tìm kiếm vét cạn tất cả các đỉnh cần tìm, và dừng khi nào tìm thấy được vị trí đích.

Thuật toán:

```
1: procedure DIJKSTRA_FIND_PATH(v_s, v_e)
2:     priority_queue ← v_s (với v_s.d = 0 và v_s.prev = null)
3:     while priority_queue ≠ ∅ do
4:         v ← priority_queue
5:         Duyệt đỉnh v
6:         if v == v_e then
7:             In ra đường và kết thúc
8:         for mỗi đỉnh u kề với đỉnh v do
9:             if đỉnh u chưa duyệt then
10:                if u ∈ priority_queue then
11:                    cập nhật u.d và u.pre nếu tốt hơn
12:                else
13:                    u.pre = v và u.d = v.d + l(v, u)
14:                    priority_queue ← u
```

Việc xây dựng Dijsktra hoàn toàn đơn giản. Tuy nhiên, hàng đợi ưu tiên là vấn đề cần quan tâm nhất trong việc xây đựng thuật toán này.

Hàng đợi ưu tiên: Chứa các phần tử chưa duyệt, và sẽ quyết định chọn đỉnh sẽ đi kế tiếp. Mức độ ưu tiên sẽ tùy thuộc vào lựa chọn và yêu cầu của bài toán cụ thể. Trong trường hợp này chúng ta quyết định ưu tiên dựa vào chi phí đường đi ngắn nhất.

2. *Thuật toán A**

Là thuật toán ra đời sau dijsktra, trước khi có tên A*, nó trải qua 2 lần trước đó là thuật toán A1 và A2.

Trên thực tế, việc cài đặt A* hoàn toàn giống với Dijstra, A* chỉ khác ở việc thay vì tính chi phí tại điểm hiện tại, thì công thêm với chi phí ước lượng.

Nói cách khác, Dijsktra chính là 1 trường hợp đặc biệt của A*, với hằng số **h = 0**.

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 của đường đi cho đến thời điểm hiện tại, nghĩa là tổng trọng số của các cạnh đã đi qua. ***h(x)*** là hàm đánh giá *heuristic* về chi phí nhỏ nhất để đến đích từ ***x***. Ví dụ, nếu "chi phí" được tính là khoảng cách đã đi qua, khoảng cách đường chim bay giữa hai điểm trên một bản đồ là một đánh giá *heuristic* cho khoảng cách còn phải đi tiếp.

Hàm ***f(x)*** có giá trị càng thấp thì độ ưu tiên của ***x*** càng cao (do đó có thể sử dụng một cấu trúc heap tối thiểu để cài đặt hàng đợi ưu tiên này)

Cũng như tìm kiếm theo chiều rộng (breadth-first search), A* là thuật toán đầy đủ (complete) theo nghĩa rằng nó sẽ luôn luôn tìm thấy một lời giải nếu bài toán có lời giải.

Nếu hàm heuristic h có tính chất thu nạp được (admissible), nghĩa là nó không bao giờ đánh giá cao hơn chi phí nhỏ nhất thực sự của việc đi tới đích, thì bản thân A* có tính chất thu nạp được (hay tối ưu) nếu sử dụng một tập đóng. Nếu không sử dụng tập đóng thì hàm h phải có tính chất đơn điệu (hay nhất quán) thì A* mới có tính chất tối ưu. Nghĩa là nó không bao giờ đánh giá chi phí đi từ một nút tới một nút kề nó cao hơn chi phí thực. Phát biểu một cách hình thức, với mọi nút x,y trong đó y là nút tiếp theo của x:

$$h(x) \leq g(y) - g(x) + h(y)$$

A* còn có tính chất hiệu quả một cách tối ưu (optimally efficient) với mọi hàm heuristic h, có nghĩa là không có thuật toán nào cũng sử dụng hàm heuristic đó mà chỉ phải mở rộng ít nút hơn A*, trừ khi có một số lời giải chưa đầy đủ mà tại đó h dự đoán chính xác chi phí của đường đi tối ưu.

Độ phức tạp thời gian của A* phụ thuộc vào đánh giá heuristic. Trong trường hợp xấu nhất, số nút được mở rộng theo hàm mũ của độ dài lời giải, nhưng nó sẽ là hàm đa thức khi hàm heuristic h thỏa mãn điều kiện sau:

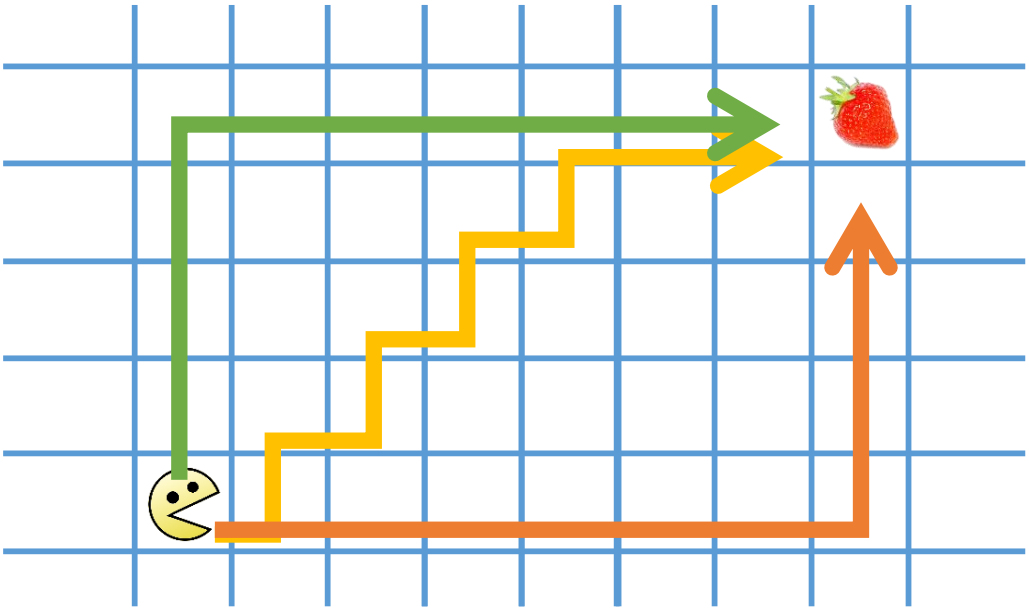
$$|h(x) - h^*(x)| \leq O(\log h^*(x))$$

Trong đó h* là heuristic tối ưu, nghĩa là hàm cho kết quả là chi phí chính xác để đi từ x tới đích.

IV. ÁP DỤNG THUẬT TOÁN VÀO ĐƯỜNG ĐI PACMAN

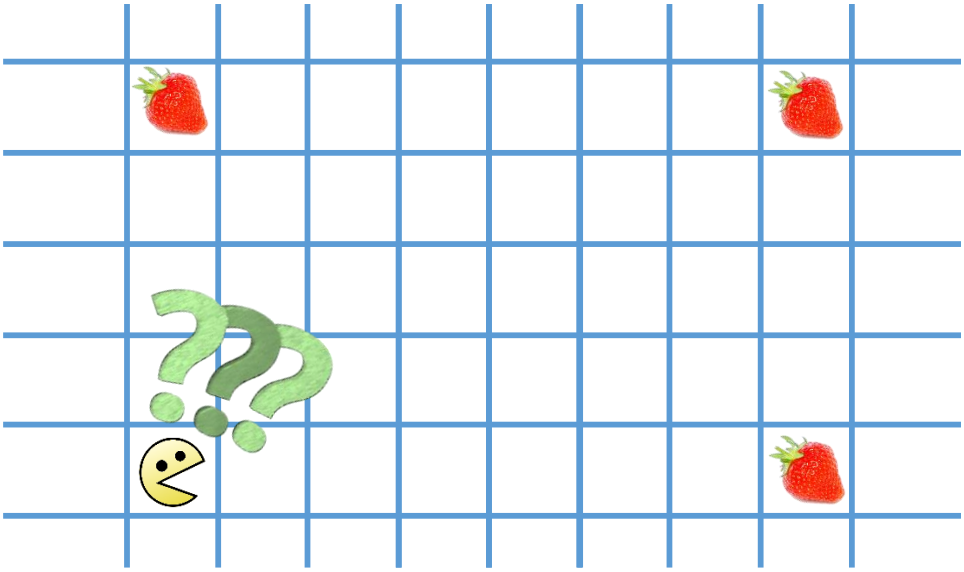
Các yêu cầu cần giải quyết:

Yêu cầu 1: Tìm đường đi ngắn nhất



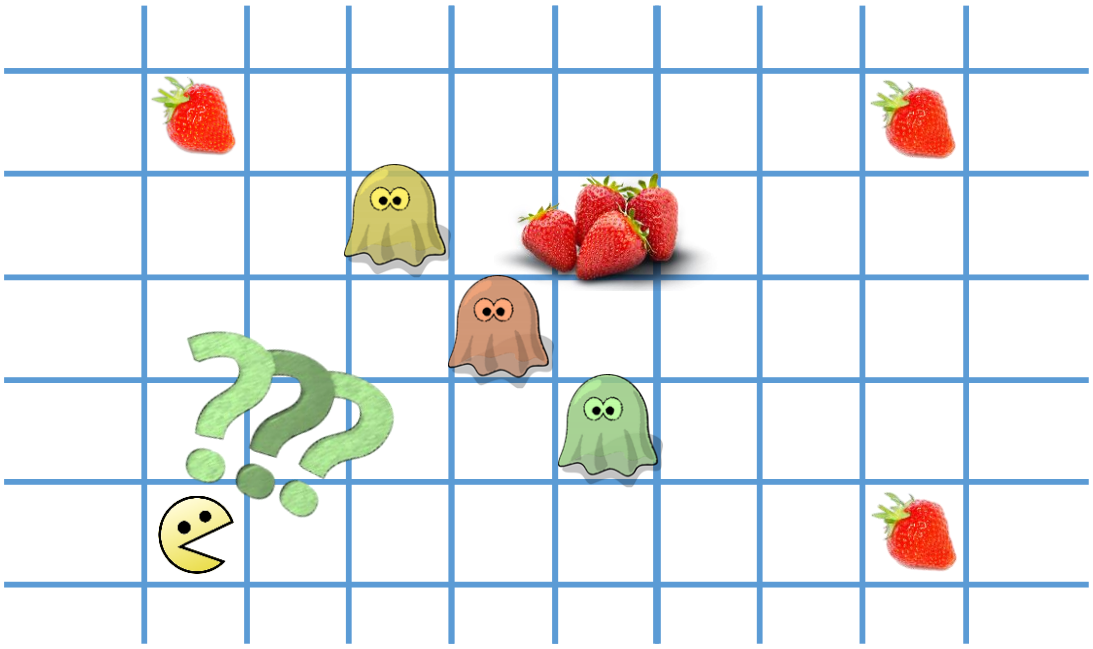
Trong yêu cầu 1 này, bài toán dường như khá đơn giản, chúng ta chỉ việc xác định trạng thái đích và bắt đầu, từ đó thực thi 1 thuật toán tìm đường bất kỳ để đến vị trí đích.

Yêu cầu 2: Bài toán nhiều trạng thái đích



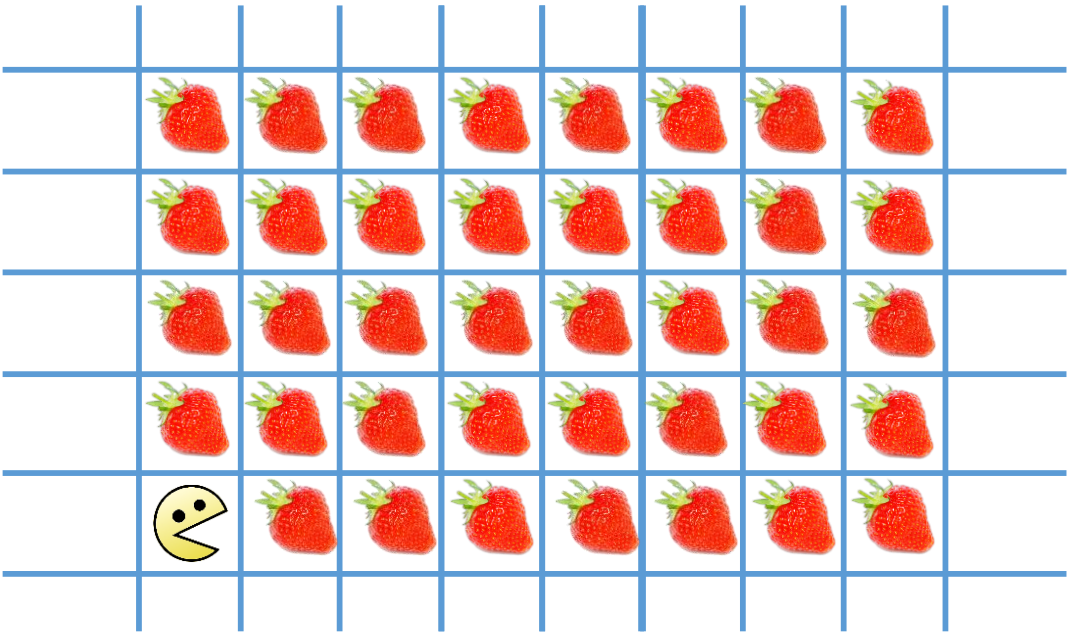
Yêu cầu 2 khó hơn, bài toán đã thay đổi, thay vì 1 trạng thái đích, thì bây giờ bài toán đã chuyển sang nhiều trạng thái đích khác nhau. Như vậy làm sao Pacman có thể đến được tất cả các trạng thái đích với quãng đường có chiều dài tối ưu. (Ở đây mình ghi là tối ưu thay vì ghi là tốt nhất, bởi vì trên thực tế, không chắc sẽ tìm ra được thuật toán tối ưu cho tất cả các dạng bản đồ)

<u>Yêu cầu 3</u>: Tìm kiếm theo tiêu chí



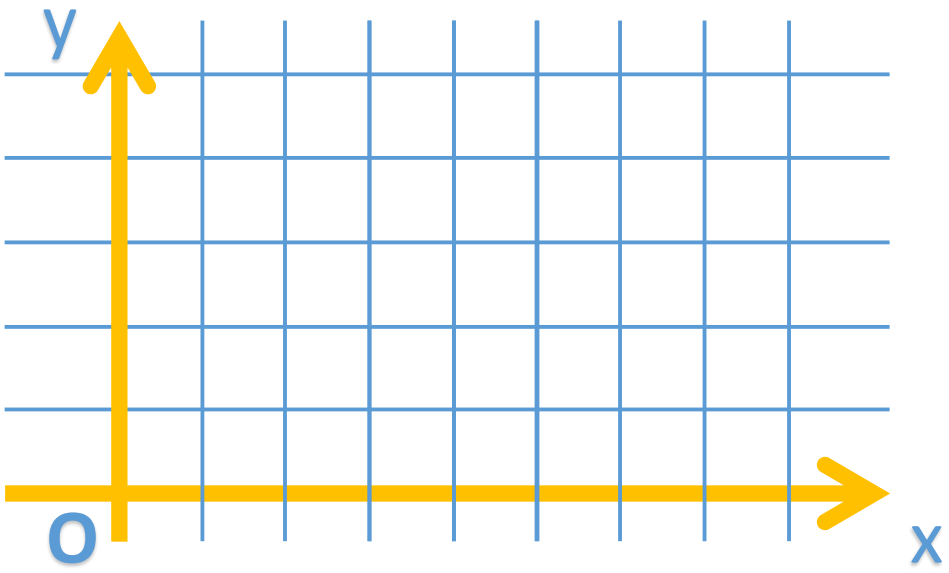
Nâng cấp từ yêu cầu 3, bài toán tìm đường đi có nhiều trạng thái đích. Bây giờ trên bản đồ có nhiều chướng ngại vật khác nhau, và các mẫu thức ăn (vị trí đích) có độ ưu tiên cao hơn. Lúc này nhiệm vụ của Pacman không đơn thuần là tới đích, mà phải làm sao tránh được chướng ngại vật (bóng ma). Đồng thời tới vị trí đích có độ ưu tiên cao hơn trước, và sau đó là các vị trí đích có độ ưu tiên thấp hơn.

Yêu cầu 4: Rất nhiều trạng thái đích khác nhau



Yêu cầu 4 thực tế là tổng hợp yêu cầu 2 và 3, nhằm mục đích vét cạn tất cả các trường hợp vị trí đích. Nhằm nhận biết rõ hơn sự khác nhau giữa các bài toán, và so sánh tương quan các heuristic khác nhau.

V. HƯỚNG GIẢI QUYẾT



Giả sử bàn đồ Pacman được đặt trong hệ trục tọa độ xOy, dựa vào các ô có khả năng đi tới được, chúng ta sẽ lập nên được ma trận kề thể hiện đường đi giữa các đỉnh. Mỗi ô chúng ta sẽ xác định được tọa độ xy.

Trọng số của ma trận kề giả sử quy ước như sau:

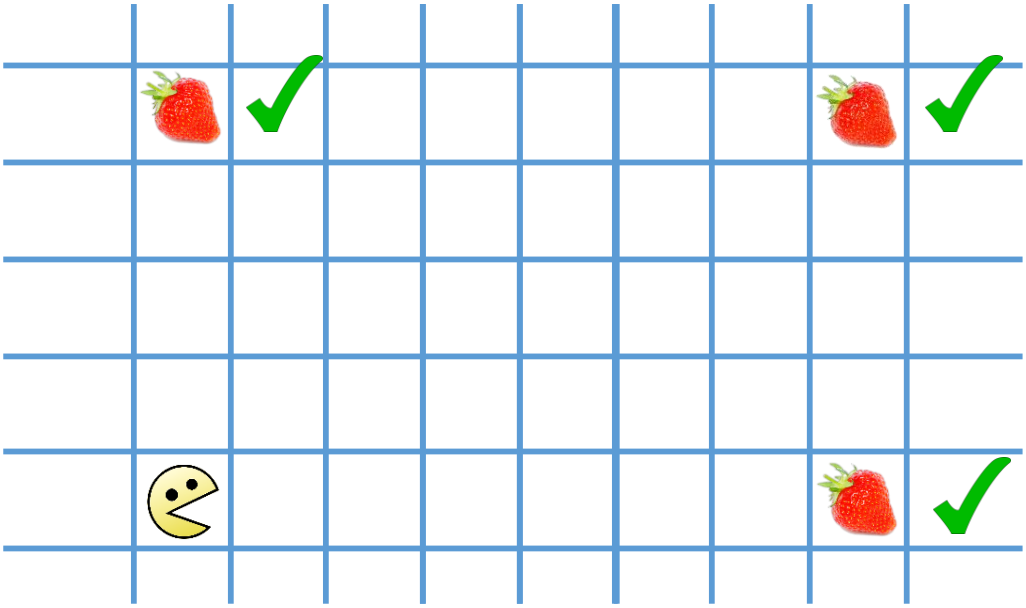
MẪU	TRỌNG SỐ
Thức ăn lớn	1
Thức ăn nhỏ	2
Không có thức ăn	0
Bóng ma	10
Tường (vật cản)	-1 (Không có đường đi)

Chiến lược tìm trang thái đích

- Lưu ý:
  - Chiến lược mang tính chất tương đối.

- Tuy vào từng trường hợp, từng bản đồ khác nhau thì sẽ có ưu và nhược điểm riêng.

- Bước 1:

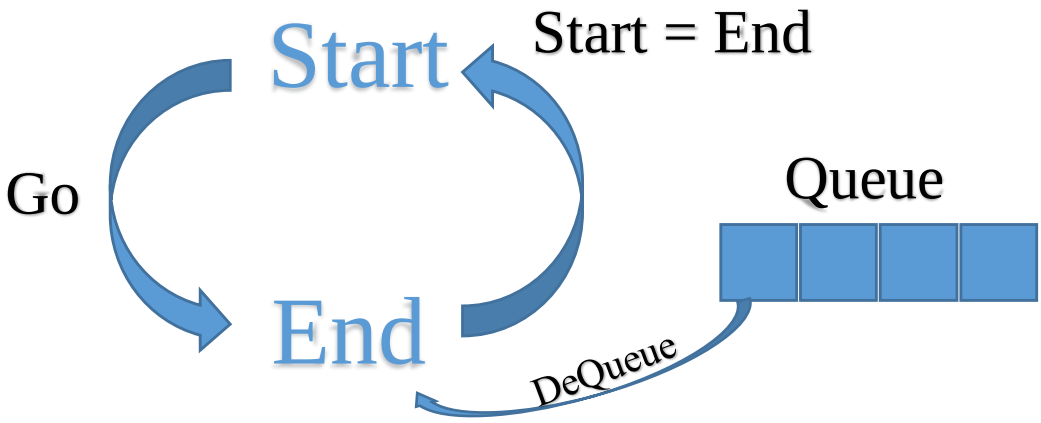


Xác định điểm bắt đầu, liệt kê tất cả các trạng thái đích.

- Bước 2: Tính toán chi phí cho tất cả các trạng thái đích.
- Bước 3: Đưa tất cả các trạng thái đích vào hàng đợi ưu tiên.
- Bước 4: Lấy 1 trạng thái đích từ hàng đợi ưu tiên để thực hiện tìm đường đi.

  Trong bước này:

  Nếu trên đường đi, vô tình đi qua trạng thái đích nào khác, thì lập tức đánh dấu là đã đi qua, và loại trừ trạng thái đích này ra khỏi hàng đợi ưu tiên.
- Bước 5: Sau khi tìm thấy đích, chuyển trạng thái đích thành điểm bắt đầu và quay lại bước 4.



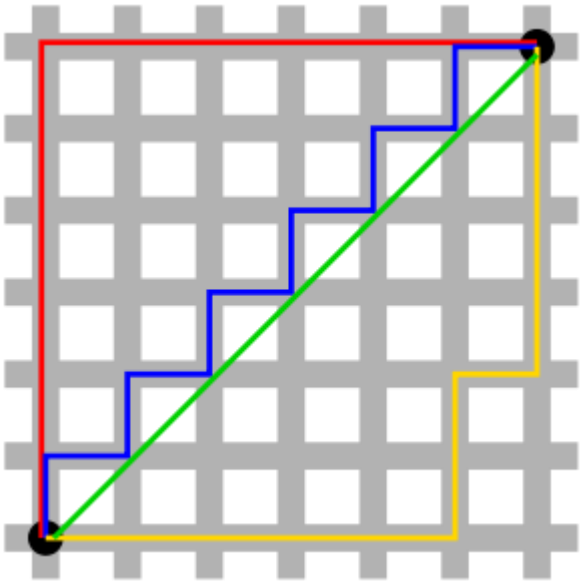
VI. TỐI ƯU THUẬT TOÁN

Để tìm được cách tối ưu thuật toán này, việc đầu tiên cần phải biết nó hoạt động như thế nào. Với A*, đặc trưng của nó là heuristic, như vậy với heuristic tốt sẽ giúp tìm đường đi nhanh và tối ưu hơn.

Có nhiều cách xác định chi phí h:

Ví dụ:

- Hàm tính **Heuristic** nhóm sử dụng đó chính là khoảng cách **Mahattan** giữa 2 đỉnh:



+ Có thể dễ dàng xác định được công thức tính chi phí ước lượng **heuristic** là:

Ví dụ, khoảng cách Manhattan giữa hai điểm: $P1$ có tọa độ $(x1, y1)$ và điểm $P2$ có tọa độ $(x2, y2)$ là: $|x1 - x2| + |y1 - y2|$

Ngoài ra còn có thể tính khoảng cách **Euclid, MaxDxDy …**

Ngoài chi phí h, chúng ta còn cài đặt 1 hàng đợi ưu tiên, hàng đợi ưu tiên có nhiều cách cài, tuy nhiên nếu dùng theo cấu trúc heap, thì chi phí sẽ giảm đi đáng kể. Trong đề tài lần này, nhóm không đi sâu về cấu trúc heap, nhưng đây cũng là 1 gợi ý, để thực hiện tối ưu thuật toán.

VII. XÂY DỰNG DEMO

- Theo đề bài chúng ta sẽ có các dạng mê cung trong các file plain text với các ký hiệu và quy ước như sau:

KÝ TỰ	Ý NGHĨA
%	Bức tường mê cung
P	Vị trí ban đầu của Pacman
.	Mẫu thức ăn nhỏ
O	Mẫu thức ăn lớn
G	Con ma ăn thịt Pacman
‘ ’	Khoảng trắng

- Xét với các ký tự quy ước như trên, chúng ta có thể dễ dàng nhận thấy rằng (**‘P’, ‘.’ , ‘o’, ‘G’**) là những quy ước cho phép Pacman đi qua, tập hợp các đỉnh này với nhau sẽ tạo nên con đường đi của Pacman.
- Với mỗi bước đi của Pacman trong mê cung sẽ là 1 Đỉnh. Các đỉnh là khoảng trắng chúng ta có thể quy ước là: chi phí, khoảng cách là như nhau, tuy nhiên các đỉnh đặc biệt khác *(Thức ăn nhỏ, Thức ăn lớn, bóng ma)* thì chúng ta có thể đánh chi phí khác nhau tùy vào mục đích của trò chơi.
- Ma trận kề cũng có thể dễ dàng xây dựng với kích thước (**n** Đỉnh x **n** Đỉnh).

- Với mỗi đỉnh của Pacman, để cho thuận tiện, chúng ta cần tập hợp những nội dung của 1 đỉnh để tiện cho quá trình sử dụng. Mỗi đỉnh sẽ bao gồm các thông tin như sau:

```
class Dinh
{
private:
    // Giá trị Value
    // 0 -> Pacman
    // 1 -> Mẫu thức ăn nhỏ
    // 2 -> Mẫu thức ăn lớn
    // 10 -> Bóng ma
    int Value;
    int Vertex; // Tên đỉnh (Đỉnh số thứ mấy)
    int x;      // Tọa độ x
    int y;      // Tọa độ y
    int cost; // Chi phí từ Start tới đỉnh hiện tại
    int previous; // Đỉnh cha
```

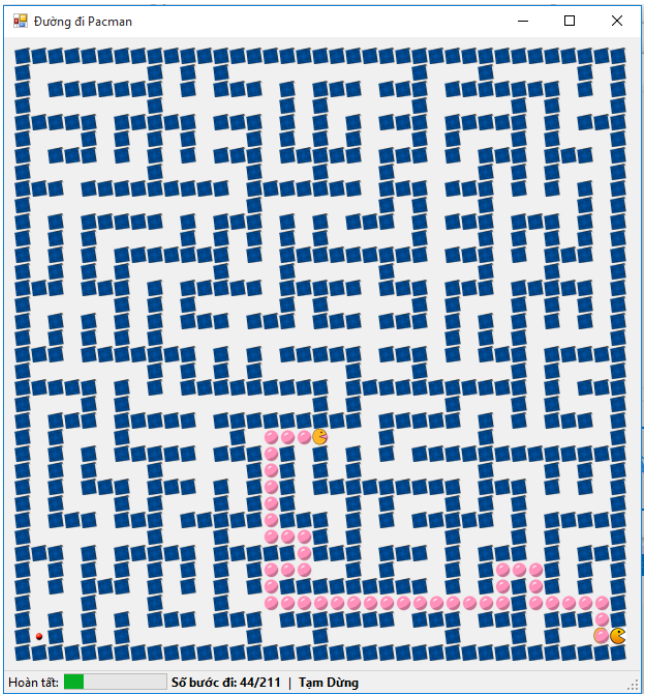
- Vì các bản đồ đều xây dựng có đặc điểm chung, nên quá trình đọc file, chúng ta có thể sử dụng chung hàm, và áp dụng xây dựng ma trận khả nghịch. Với từng yêu cầu khác nhau thì đồ thị **Graph** thể hiện khác nhau, tuy nhiên, Những thông tin chính mà **Graph** cần phải có là:

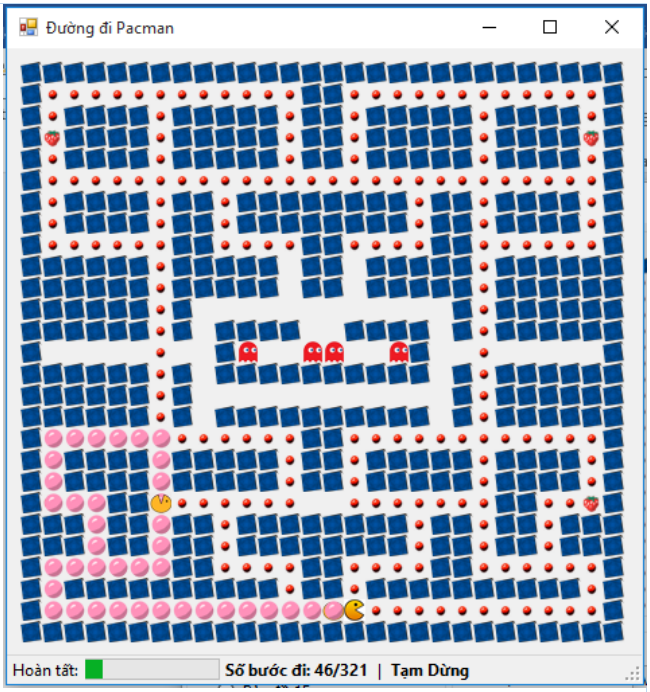
```
class MyGraph
{
private:
    int N; // Số lượng đỉnh trong đồ thị
    int start, end; // Đỉnh bắt đầu và kết thúc
    vector< vector<int> > MATRIX; // Ma trận kề thể hiện trạng thái của bản đồ
    vector<int> lbl; // Mảng lưu nút đã đi qua chưa
    vector<Dinh> arrDinh; // Mảng chứa danh sách các đỉnh
```

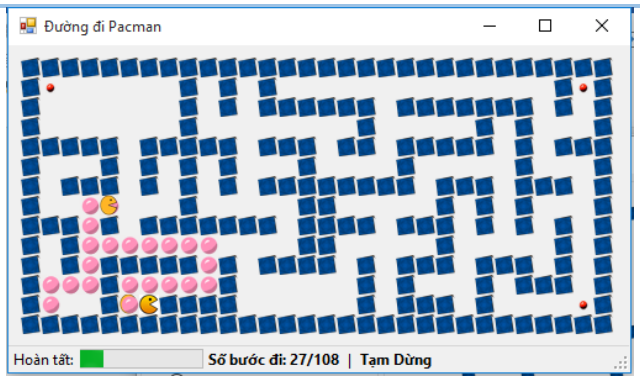
- Xử lý đọc file:
    - Tiến hành đọc từng dòng trong file **.lay**
    - Với mỗi loại ký tự Pacman có thể đi qua thì tạo các **DINH** tương ứng và đẩy vào mảng chứa, sau đó dựa vào mảng này để tiến hành xây dựng ma trận kề.

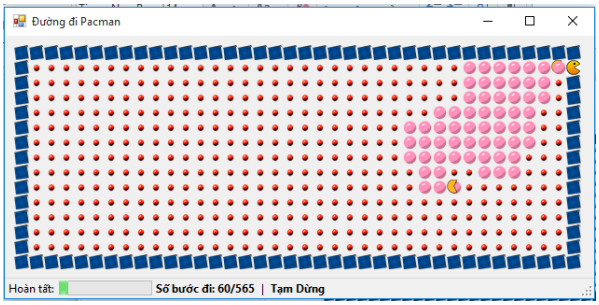
VIII. HÌNH ẢNH CÁC MA TRẬN SỬ DỤNG TRONG ĐỒ ÁN

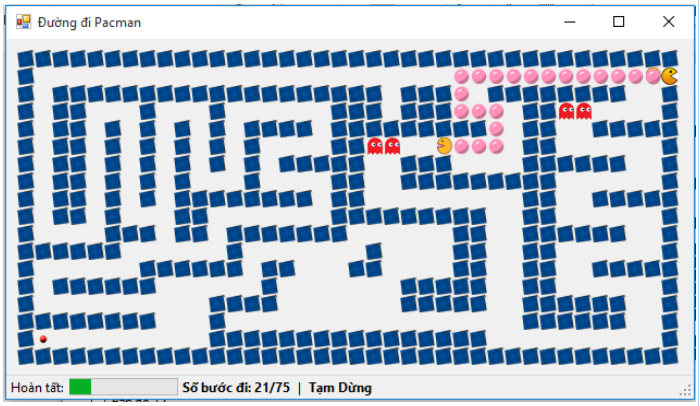
(Đây là hình ảnh bản đồ nhóm xây dựng trên C#)

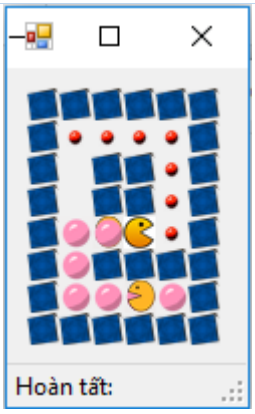

Đường đi Pacman
Hoàn tất:
Số bước đi: 44/211 | Tạm Dừng

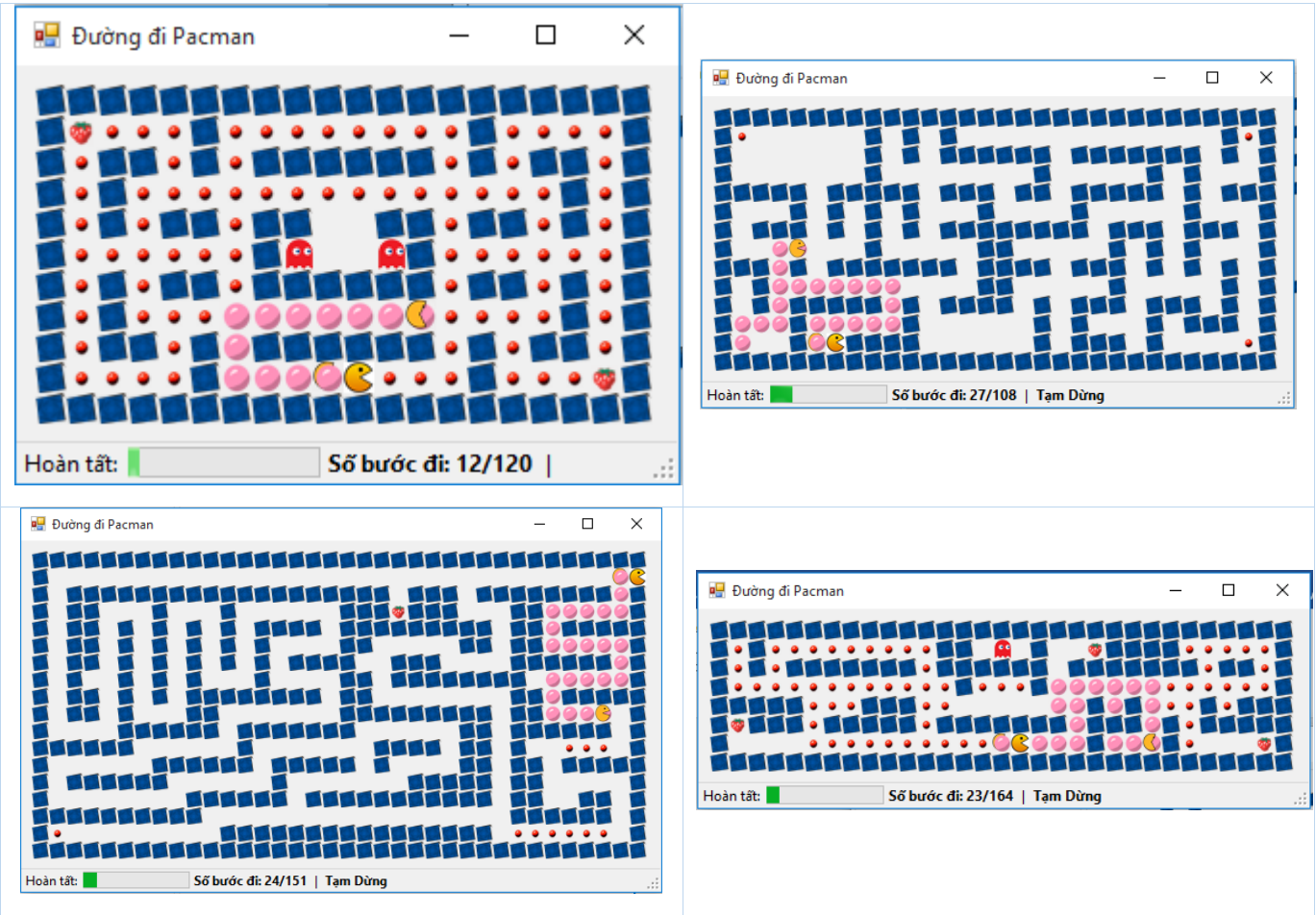

Đường đi Pacman
Hoàn tất:
Số bước đi: 46/321 | Tạm Dừng


Đường đi Pacman
Hoàn tất:
Số bước đi: 27/108 | Tạm Dừng


Đường đi Pacman
Hoàn tất:
Số bước đi: 60/565 | Tạm Dừng


Đường đi Pacman
Hoàn tất:
Số bước đi: 21/75 | Tạm Dừng


Hoàn tất:



IX. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

Thực hiện hoàn hiện thuật toán Dijsktra và A*, với các hàm ước lượng chi phí Khác nhau: Manhattan, Euclid, MaxDxDy…

Thể hiện đường đi Pacman trên giao diện đồ họa.

Tìm được giải pháp tìm đường đi tối ưu cho Pacman.

Thể hiện được tính ứng dụng thực tế của bài toán vào tìm được đi thực thế, vượt chướng ngại vật, ưu tiên trạng thái đích …

X. TỔNG KẾT

Với bài toán Pacman có thể áp dụng nhiều thuật toán tìm đường đi BFS, DFS, UCS, Dijsktra, A*

Đôi khi với cả A* và một heuristic tốt thì Pacman cũng khó mà tìm được đường đi tối ưu để lấy hết các mẩu thức ăn. Chúng ta có một giải pháp thay thế là tìm một đường tương đối tốt trong thời gian ngắn. Cụ thể Pacman sẽ luôn tiến đến mẩu thức ăn gần nó nhất.

Ứng dụng bài toán thực tế (Tìm đường đi cho nhiều trạng thái đích)

Tìm đường đi an toàn, vượt chướng ngại vật (Tìm đường theo tiêu chí)

XI. CÁC NGUỒN TÀI LIỆU THAM KHẢO

- Bài tập hướng dẫn hàng tuần (Tham khảo thuật toán)
- Slide bài giảng lý thuyết.
- Wikipedia.
- Demo PathFinding.js (https://github.com/qiao/PathFinding.js)
- Các slide hướng dẫn GVTH cung cấp
- Cảm ơn các bạn trong lớp đã góp ý, cũng như giúp nhóm xử lý 1 số các lỗi trong quá trình xây dựng.