**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

🙤🙧🟍🙥🙦



**BÁO CÁO ĐỒ ÁN MÔN TRÍ TUỆ NHÂN TẠO**

**ĐỒ ÁN: GAME PAC-MAN**

**Lớp:** CS106.H21.1

**Giáo viên hướng dẫn:** Huỳnh Thị Thanh Thương.

|  |  |  |
| --- | --- | --- |
| Sinh viên thực hiện: | | |
| STT | Họ tên | MSSV |
| 1 | Nguyễn Thiên Ân | 15520009 |
| 2 | Hồ Đức Trí Mãnh | 15520471 |
| 3 | Võ Duy Thật | 15520821 |

**TP. HỒ CHÍ MINH – 06/2017**

**MỤC LỤC**

Chương I: Giới thiệu bài toán

1.1. Nguồn gốc trò chơi............................................................................................... 2

1.1.1. Xuất xứ…………………………………………………………………………. 2

1.1.2. Nhóm thiết kế……………………………………………………………………2

1.2. Mô tả trò chơi……………………………………………………………………2

1.2.1 Mục tiêu và luật chơi…………………………………………………………….2

1.2.2 Những yếu tố trong game……………………………………………………….3

Chương II: Cơ sở lý thuyết……………………………………………………………..7

Chương III: Phân tích và thiết kế………………………………………………………10

3.1. Phân tích bài toán……………………………………………………………….10

3.1.1. Ý tưởng giải quyết bài toán.................................................................................10

3.1.2. Hướng giải quyết bài toán……………………………………………………..10

3.2. Cấu trúc dữ liệu và cách biểu diễn trạng trái của bài toán……………………..11

3.3. Các vấn đề và thuật giải………………………………………………………..13

3.3.1. Mô hình hóa cơ sở dữ liệu……………………………………………………...13

3.3.2. Thực hiện thuật giải A\*………………………………………………………..14

3.3.3. Vấn đề tìm đường đi cho ghost ở trạng thái Run………………………………20

3.4. Ví dụ về thuật toán/thuật giải…………………………………………………..22

Chương IV: Ứng dụng………………………………………………………………...28

4.1. Giới thiệu chương trình ứng dụng……………………………………………..28

4.1.1. Yêu cầu…………………………………………………………………..…….28

4.1.2. Giao diện trò chơi……………………………………………………………...28

4.1.3. Cách sử dụng trò chơi……………………………………………………….…30

4.2. Cài đặt……………………………………………………………………….…30

4.3. Kết quả chạy chương trình……………………………………………………..30

Chương V: Kết luận…………………………………………………………………...31

5.1. Kết quả đạt được……………………………………………………………….31

5.2. Hạn chế………………………………………………………………………...31

5.3. Hướng phát triển……………………………………………………………….31

Tài liệu tham khảo…………………………………………………………………….32

**CHƯƠNG I: GIỚI THIỆU BÀI TOÁN**

**1.1. Nguồn gốc trò chơi**

**1.1.1. Xuất xứ**

- Được phát triển bởi Namco và phát hành đầu tiên tại Nhật Bản vào 22 tháng 5 năm 1980. Pac-Man được xem là một trò chơi kinh điển và trở thành 1 biều tượng của văn hóa đại chúng những năm 80.

- Tên cũ: Puck-Man.

- Loại trò chơi : arcade (dạng trò chơi chơi rên máy giải trí sử dụng bằng đồng xu). [1]

**1.1.2. Nhóm thiết kế**

- Thiết kế: Toru Iwatani.

- Lập trình: Shigeo Funaki.

- Âm nhạc: Toshio Kai. [2]

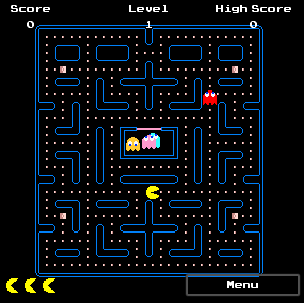
**1.2. Mô tả trò chơi**

**1.2.1. Mục tiêu và luật chơi**

Mục đích của trò chơi rất đơn giản - người chơi điều khiển Pac-Man được đặt trong một mê cung với những “thức ăn” (Pac dots) (thể hiện bởi những chấm tròn) và phải ăn hết chúng để qua level. Điểm đặc biệt là Pac-Man sẽ liên tục di chuyển, người chơi chỉ có thể điều khiển trái, phải, lên, xuống và Pac-Man không thể đứng lại giữa đường. Nhiệm vu này được làm khó bởi 4 ghosts đuổi theo Pac-Man suốt mê cung. Nếu Pac-Man đụng phải ghost thì sẽ mất 1 mạng và vị trí của Pac-Man và ghost sẽ được reset về lúc bắt đầu, dù những chấm “thức ăn” vẫn giữ nguyên. Cách đơn giản hơn để việc Pac-Man phòng thủ ghost, có các viên “năng lượng” ở 4 góc mê cung. Ăn 1 viên sẽ làm cho ghosts bị mất khả năng tấn công Pac-Man trong một khoảng thời gian ngắn và Pac-Man có thể ăn ghosts để có điểm bonus trong khoảng thời gian trên. Một con ghost bị ăn sẽ không biến mất hẳn mà quay lại với vị trí ban đầu trước khi tiếp tục đuổi theo Pac-Man.

Mọi level của Pac-Man đều có nền mê cung giống nhau, gồm 240 chấm “thức ăn” bình thường và 4 viên năng lượng. Dù phần nền luôn giống nhau nhưng độ khó qua mỗi level được thể hiện qua tốc độ của Pac-Man, ghosts.

**1.2.2. Những yếu tố trong game**



**Pac-Man**



Là nhân vật chính duy nhất trong game được điều khiển bằng người chơi. Pac-Man có nhiệm vụ là phải ăn hết tất cả các viên thức ăn trong màn chơi đồng thời phải chạy trốn khỏi sự truy đuổi của Ghosts, tuyệt đối không được va chạm với Ghosts vì mỗi lần va chạm Pac-Man sẽ mất 1 mạng sống. Tuy nhiên, khi ăn được viên năng lượng thì Pac-Man có thể tiêu diệt được ghost bằng cách ăn ghost như ăn một pac-dot và được cộng thêm 200 điểm cho mỗi con ghost. Mỗi level Pac-Man có tổng cộng 3 mạng chơi tất cả.

**Ghosts**

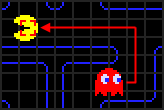


Là các nhân vật đóng vai trò truy đuổi Pac-Man trong game, không để cho Pac-Man ăn hết tất cả viên thức ăn và chạy trốn khi Pac-Man ăn được viên năng lượng. Các ghost được di chuyển tự động dựa vào thuật toán A\* tìm đường đi (sẽ được phân tích sau) để vây bắt Pac-Man tuy nhiên tốc độ di chuyển luôn nhỏ hơn Pac-Man. Trong mỗi level sẽ có bốn con ghost thực hiện nhiệm vụ bao gồm: Ghost Đỏ (Blinky), Ghost Xanh (Inky), Ghost Hồng (Pinky) và Ghost Cam (Clyde). Mỗi ghost sẽ có 3 trạng thái chính đó là:

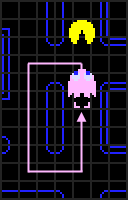
* Phân tán (Scatter): Khi bắt đầu game thì 4 ghosts sẽ được mặc định tản ra 4 góc của mê cung đến một điểm cố định rồi sau đó mới bắt đầu vây bắt Pac-Man.
* Truy đuổi (Chase): Đây là trạng thái chiếm thời gian nhiều nhất trong tổng thời gian tồn tại của ghost cũng chính là lúc ghost thực hiện nhiệm vụ của mình là truy đuổi Pac-Man.
* Chạy trốn (Run): Là trạng thái xuất hiện mỗi khi Pac-Man ăn được viên năng lượng. Ở trạng thái này mỗi cố gắng tìm con đường có khoảng cách xa nhất so với ghost để tránh bị Pac-Man nuốt chửng.

Mỗi ghost sẽ được quy định một tính cách khác nhau nhằm tạo ra sự thú vị khi vây bắt Pac-Man, cụ thể: [3]

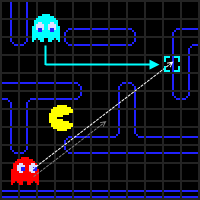
* **Ghost Đỏ**

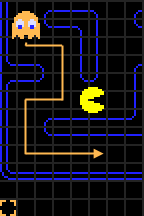
Ghost Đỏ bắt đầu ở ngoài nhà ghost, và là ghost nguy hại đầu tiên, vì nó là ghost đầu tiên truy đuổi Pac-Man sau khi kết thúc trạng thái Phân tán. Và trong trạng thái Truy đuổi, mục tiêu của nó là tọa độ chính xác của Pac-Man, khác với các ghost còn lại sẽ được trình bày bên dưới. Điều này dẫn đến việc ghost Đỏ luôn đi theo ngay sau Pac-Man.

* **Ghost Hồng**

 Được coi như là kẻ phục kích trong tất cả các ghosts. Ghost Hồng thay vì lấy mục tiêu là vị trí chính xác như ghost Đỏ thì nó chọn mục tiêu để tìm đường đi là vị trí đón đầu 4 bước so với hướng đi của Pac-Man. Vì vậy trong một số trường hợp đối đầu với Pac-Man (ví dụ như hình trên) ta thấy cách chọn mục tiêu này không hữu ích, thay vì chạy thẳng để tiêu diệt Pac-Man thì nó lại chuyển hướng đi đường khác. Nhưng tóm lại, với cách phục kích như vậy ghost Hồng tỏ ra khá hiệu quả trong việc vây bắt và tăng thêm sự hấp dẫn cho trò chơi.

* **Ghost Xanh**

  
 Ghost Xanh có biệt danh Inky. Inky khó dự đoán, bởi vì nó là ghost duy nhất sử dụng một yếu tố khác ngoài vị trí/ hướng của Pac-Man khi xác định mục tiêu. Ghost Xanh sử dụng cả vị trí của Pac-Man cũng như vị trí của Đỏ trong tính toán của nó. Để xác định mục tiêu của Xanh, đầu tiên chúng ta sẽ bắt đầu bằng cách chọn vị trí hai gạch ở phía trước của Pac-Man theo hướng hiện tại của Pac-Man, tương tự như phương pháp nhắm mục tiêu của Hồng. Từ đó, vẽ một vector từ vị trí của Đỏ đến ô này, và sau đó tăng gấp đôi độ dài của vector. Các gạch mới này, nơi vector mở rộng kết thúc sẽ là mục tiêu thực tế của Xanh. Kết quả là mục tiêu của ghost Xanh có thể thay đổi rất nhiều khi ghost Đỏ không ở gần Pac-Man, nhưng nếu Đỏ đang theo đuổi chặt chẽ, Xanh cũng sẽ đuổi theo.

* **Ghost Cam**

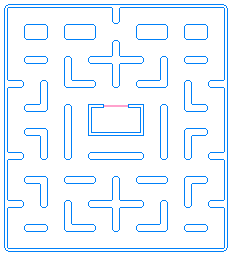
Ghost cam hay còn gọi là Clyde. Bất cứ khi nào Cam cần xác định mục tiêu, Cam sẽ tính khoảng cách giữa nó và Pac-Man. Nếu Cam xa hơn 9, cách chọn mục tiêu sẽ giống Đỏ, dùng vị trí hiện tại của Pac-Man là mục tiêu. Tuy nhiên, khi khoảng cách giữa nó với Pac-Man bé hơn 9, mục tiêu của Cam chính là vị trí tọa độ (1, 2) - góc trái dưới cùng mê cung. Nhìn chung, sự kết hợp của 2 phương pháp này có ảnh hưởng là thay đổi giữa 2 trạng thái : đi thẳng tới Pac-Man, và sau đó đổi ý chạy vào góc của nó khi tới quá gần. Tuy vậy, nhưng nếu khi khoảng cách nhỏ hơn 9 mà Pac-Man lại cũng đang di chuyển trên đường đi trùng với đường đi đến góc trái của ghost Cam thì nó vẫn đuổi theo Pac-Man mà không hề chuyển hướng.

**Nhà Ghost**



Là nơi Ghost được sinh ra mỗi khi bắt đầu game hoặc khi hồi sinh. Pac-Man không bao giờ được đi vào khu vực “nhà” của ghost và ghost cũng vậy, khi ra khỏi nhà thì không thể vào lại.

**Tường**

****

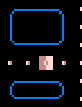
Là các vật thể hình khối được sắp xếp cố định trong mê cung và tạo thành các con đường đi mà ghost, Pac-Man có thể di chuyển được. Cả ghosts và Pac-Man đều không thể đi xuyên qua được tường.

**Pac-dots (Viên thức ăn)**



Là các chấm tròn nhỏ được đặt rải rác trên các đường đi trên mê cung. Các viên thức ăn chỉ biến mất mỗi khi Pac-Man đi ngang qua nó và Pac-Man sẽ được cộng 10 điểm khi ăn 1 viên. Ghosts không ăn được pac-dots và chúng cũng không bị mất đi khi ghosts đi qua chúng.

**Energizers (Viên năng lượng)**

Là các khối hình chữ nhật to hơn pac-dots, gồm tất cả 4 viên trong một màn chơi và chỉ dành riêng cho Pac-Man. Nó có tác dụng giúp cho Pac-Man power-up và có thể ăn được cả ghosts khi va chạm với nhau. Đồng thời, mỗi viên năng lượng cũng mang lại cho người chơi thêm 200 điểm.

**CHƯƠNG II: CƠ SỞ LÝ THUYẾT**

- Bài toán tìm đường đi trong game Pac-Man được giải quyết dựa trên cơ sở lý thuyết thuật toán A\*.

- A\*là một phiên bản đặc biệt của AKT áp dụng cho trường hợp đồ thị. Thuật giải A\* có sử dụng thêm tập Close để lưu trữ những trường hợp đã được xét đến. A\* mở rộng AKT bằng cách bổ sung cách giải quyết trường hợp khi "mở" một nút mà nút này đã có sẵn trong Open hoặc Close. Khi xét đến một trạng thái Ti, bên cạnh việc lưu trữ 3 giá trị cơ bản g, h, f để phản ánh độ tốt của trạng thái đó, A\* còn lưu trữ thêm hai thông số sau :

- Trạng thái cha của trạng thái Ti (ký hiệu là Cha(Ti): cho biết trạng thái dẫn đến trạng thái Ti. Trong trường hợp có nhiều trạng thái dẫn đến Ti thì chọn Cha(Ti) sao cho chi phí đi từ trạng thái khởi đầu đến Ti là thấp nhất, nghĩa là: g(Ti) = g(Tcha) + cost(Tcha, Ti) là thấp nhất.

- Danh sách các trạng thái kế tiếp của Ti:danh sách này lưu trữ các trạng thái kế tiếp Tk của Ti sao cho chi phí đến Tk thông qua Ti từ trạng thái ban đầu là thấp nhất. Thực chất thì danh sách này có thể được tính ra từ thuộc tính Cha của các trạng thái được lưu trữ. Tuy nhiên, việc tính toán này có thể mất nhiều thời gian (khi tập Open, Close được mở rộng) nên người ta thường lưu trữ ra một danh sách riêng. Sau đây là mô phỏng cách làm việc của thuật toán mà không có lưu trữ danh sách:

**Bước 1** : Đặt Open chỉ chứa T0. Đặt g(T0) = 0, h(T0) = 0 và f(T0) = 0. Đặt Close là tập hợp rỗng.

**Bước 2** : Lặp lại các bước sau cho đến khi gặp điều kiện dừng :

**Bước 2.1**: Nếu Open rỗng : bài toán vô nghiệm, thoát.

**Bước** **2.2**: Ngược lại, chọn Tmax trong OPEN sao cho f(Tmax) là nhỏ nhất.

**Bước** **2.2.1**: Lấy Tmax ra khỏi OPEN và đưa Tmax vào CLOSE.

**Bước 2.2.2**: Nếu Tmax chính là Tg thì thoát và thông báo lời giải là Tmax.

**Bước 2.2.3 :** Nếu Tmax không phải là Tg. Tạo ra danh sách tất cảcác trạng thái kế tiếp của Tmax. Gọi một trạng thái này là Tk. Với mỗi Tk, làm các bước sau :

**Bước 2.2.3.1**: Tính g(Tk) = g(Tmax) + cost(Tmax, Tk).

**Bước 2.2.3.2**: Nếu tồn tại Tk’ trong OPEN trùng với Tk.

* Nếu g(Tk) < g(Tk’) thì đặt g(Tk’) = g(Tk).

Tính lại f(Tk’).

Đặt Cha(Tk’) = Tmax.

**Bước 2.2.3.3:**  Nếu tồn tại Tk’ trong CLOSE trùng với Tk.

* Nếu g(Tk) < g(Tk’) thì đặt g(Tk’) = g(Tk).

Tính lại f(Tk’).

Đặt Cha(Tk’) = Tmax.

Cập nhật lại sự thay đổi giá trị g, f cho tất cả các trạng thái kế tiếp của Ti (ở tất cả các cấp) đã được lưu trữ trong CLOSE và OPEN.

**Bước 2.2.3.4:** Nếu Tk chưa xuất hiện trong cả OPEN lẫn CLOSE thì thêm Tk vào OPEN

Tính : f(Tk) = g(Tk)+h(Tk).

Có một số điểm cần giải thích trong thuật giải này :

- Đầu tiên là việc sau khi đã tìm thấy trạng thái đích Tg, làm sao để xây dựng lại được "con đường" từ T0 đến Tg. Rất đơn giản, bạn chỉ cần lần ngược theo thuộc tính Cha của các trạng thái đã được lưu trữ trong CLOSE cho đến khi đạt đến T0. Đó chính là "con đường" tối ưu đi từ Tg đến T0 (hay nói cách khác là từ T0 đến Tg).

- Điểm thứ hai là thao tác cập nhật lại g(Tk’), f(Tk’) và Cha(Tk’) trong bước 2.2.3.2 và 2.2.3.3. Các thao tác này thể hiện tư tưởng : "luôn chọn con đường tối ưu nhất". Như chúng ta đã biết, giá trị g(Tk’) nhằm lưu trữ chi phí tối ưu thực sựtính từ T0 đến Tk’. Do đó, nếu chúng ta phát hiện thấy một "con đường" khác tốt hơn thông qua Tk (có chi phí nhỏ hơn) con đường hiện tại được lưu trữ thì ta phải chọn "con đường" mới tốt hơn này. Trường hợp 2.2.3.3 phức tạp hơn. Vì từ Tk’ nằm trong tập CLOSE nên từ Tk’ ta đã lưu trữ các trạng thái con kế tiếp xuất phát từ Tk’. Nhưng g(Tk’) thay đổi dẫn đến giá trị g của các trạng thái con này cũng phải thay đổi theo. Và đến lượt các trạng thái con này lại có thể có các các trạng thái con tiếp theo của chúng và cứ thế cho đến khi mỗi nhánh kết thúc với một trạng thái trong OPEN (nghĩa là không có trạng thái con nào nữa). Để thực hiện quá trình cập nhật này, ta hãy thực hiện quá trình duyệt theo chiều sâu với điểm khởi đầu là Tk’. Duyệt đến đâu, ta cập nhật lại g của các trạng thái đến đó (dùng công thức g(T) = g(Cha(T)) +cost(Cha(T), T)) và vì thế giá trị f của các trạng thái này cũng thay đổi theo.

**CHƯƠNG III: PHÂN TÍCH VÀ THIẾT KẾ**

**3.1. Phân tích bài toán**

**3.1.1. Ý tưởng giải quyết bài toán**

Bài toán: Xây dựng game Pac-Man.

Ý tưởng: Tạo dựng menu game, các chức năng Pause, chức năng Exit game, tạo bản đồ game chính, Pac-Man, các con ghost, các điểm thức ăn cho Pac-Man, các viên năng lượng. Xây dựng các toán tử di chuyển cho cả ghost và Pac-Man, xử cho tường không thể đi qua được, xử lý va chạm khi Pac-Man ăn các hạt thức ăn và khi ghost chạm vào Pac-Man. Giải quyết bài toán tìm đường đi cho Ghost đến vị trí của Pac-Man.

Bài toán: Tìm đường đi cho mỗi Ghost đến Pac-Man

Giải quyết bằng cách xây dựng đồ thị tìm kiếm dựa trên vị trí của Pac-Man và ghosts trên bản đồ. Sau đó, áp dụng thuật giải A\* để tìm đường đi ngắn nhất từ ghosts đến Pac-Man.

Lý do sử dụng thuật giải A\*: Vì đây là bài toán tìm kiếm không có đối thủ, nên không dùng thuật giải Minimax. Hơn nữa, A\* là thuật giải tối ưu nhất để giải quyết bài toán tìm đường đi trong không gian tìm kiếm xác định.

Cách thức áp dụng: xem các giao lộ trên bản đồ là một đỉnh của đồ thị, ta có thể xây dựng đồ thị tìm kiếm, và dựa vào tọa độ các đỉnh ta có thể tính được các giá trị của hàm lượng giá Heuristic. Đồng thời, xây dựng một danh sách các đỉnh cha để tìm ra đường đi ngắn nhất.

Phát biểu bài toán một cách hình thức:

+ Map: Được lưu dưới dạng ma trận 31x28.

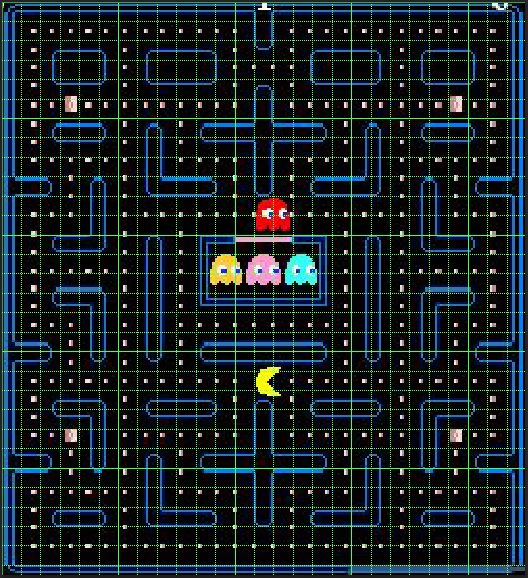
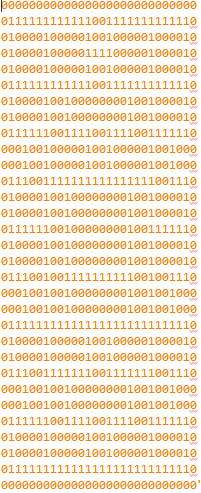
+ Trạng thái ban đầu: Pac-Man và ghost được đặt tại các vị trí có tọa độ xác định trên bản đồ.

+ Trạng thái kết thúc: 1 trong 4 ghosts bắt được Pac-Man hoặc Pac-Man ăn hết các chấm Pac-dots.

+ Toán tử: Lên, xuống, sang trái, sang phải.

**3.1.2. Hướng giải quyết bài toán**

Các giao lộ trên bản đồ sẽ được lưu tọa độ lại và được xem như là một đỉnh của đồ thị tìm kiếm. Áp dụng thuật toán A\* với hàm đánh giá f(n) = g(n) + h(n). Trong đó, g(n) là khoảng cách giữa các giao lộ, h(n) là khoảng cách theo đường chim bay tính từ ghost đến vị trí Pac-Man. Khi ghost đang đứng tại vị trí giao lộ, thuật toán A\* sẽ được thực hiện và tính toán đường đi tốt nhất để đi đến vị trí Pac-Man.

 **3.2. Cấu trúc dữ liệu và cách biểu diễn các trạng thái của bài toán**

Bản đồ trò chơi được chia nhỏ thành các ô với kích thước 31x28 ô. Có 2 loại ô là: ô đường đi (Pac-Man và ghosts có thể di chuyển được), ô tường cản. Mỗi ô sẽ được quản lí thông qua class **Tile.cs**. Hình ma trận 0, 1 bên cạnh là mô phỏng của đường đi và tường trong bản đồ game. Số 0 biểu diễn cho tường cản, số 1 là đường đi.

* **Tile**: Thuộc tính và phương thức chính

+ public int x, y: Tọa độ của Tile trên bản đồ, 0 ≤ x, ≤ 28 ; 0 ≤ y ≤ 31.

+ public float f, g, h: Các thành phần hàm Heuristic, f = g + h.

+ public bool occupied: Xác định 1 Tile có phải là tường hay không, occupied có giá trị true khi là tường, fasle nếu là đường đi

+ public bool isIntersection: Xác định Tile có phải là giao lộ (ngã ba hoặc ngã tư) hay không, nhận giá trị true nếu là giao lộ.

+ public Tile left, right, up, down, before: Thể hiện sự liên kết giữa các Tile, nếu là tường thì nhận giá trị là null. Đối với **before**, đây là thuộc tính mấu chốt để tìm đường đi bằng thuật giải A\*, sẽ được trình bày ở phần dưới.

* **TileManager**: Quản lí các **Tile**, cung cấp các phương thức tính toán cho thuật giải tìm đường đi.

+ private void ReadTiles(): Đọc dữ liệu string 0, 1 (Hình minh họa) và lưu thành mảng các Tile.

+ public float distance(Tile tile1, Tile tile2) : Hàm tính khoảng cách giữa 2 Tile được dùng để tính 2 giá trị g và h trong thuật giải A\*.

+ public int Index(int X, int Y) : Với 2 tham số x, y là tọa độ của 1 đối tượng trên bản đồ, hàm Index sẽ trả về vị trí của Tile tương ứng với tọa độ trong List<Tile>.

* **Ghost**: Mỗi ghost được quản lí thông qua class **GhostMove.cs** và **AI.cs**.
* Class **GhostMove.cs** gồm các thuộc tính và phương thức chính:

+ public float speed: Tốc độ di chuyển của ghost, tăng dần theo level.

+ private Vector3 waypoint: Tọa độ tiếp theo để di chuyển đến của ghost tại một vị trí bất kì

+ public Vector3 direction: Hướng di chuyển của ghost. Tập giá trị của nó bao gồm: left, right, up, down. Thuật giải A\* sẽ quyết định giá trị cho thuộc tính này khi ở trạng thái Chase.

+ State state: Trạng thái của ghost với các giá trị: Init, Wait, Scatter, Chase, Run.

+ void Scatter(): Ứng với trạng thái Scatter. Khi bắt đầu màn chơi, 4 con ghost sẽ mặc định chạy về 4 góc của bản đồ theo một con đường đã được vẽ sẳn.

+ void ChaseAI() – Hàm thực hiện chức năng đuổi bắt Pac-Man, ứng với state Chase.

+ void RunAway() – Hàm thực hiện chức năng chạy trốn Pac-Man khi bị Pac-Man truy đuổi

* Class **AI.cs** gồm các thuộc tính và phương thức chính:

+ public Transform target: Là đối tượng Pac-Man trong game,

dùng để xác định tọa độ của Pac-Man trên bản đồ và sử dụng trong thuật giải tìm đường đi.

+ private List<TileManager.Tile> tiles: Lưu trữ danh sách các Tile nhận giá trị từ biến tiles ở class TileManager.

+ private TileManager manager: Khai báo để sử dụng các phương thức tính khoảng cách (distance), vị trí (Index) các Tile. Sử dụng trong hàm AILogic() và RunLogic().  
 + public GhostMove ghost: Nhận giá trị là 1 trong 4 con ghosts blinky, pinky, inky, clyde.  
 + public TileManager.Tile nextTile: Bước đi tiếp theo của ghost, sử dụng trong 2 hàm AILogic() và RunLogic()  
     + public TileManager.Tile targetTile: Tile mục tiêu (tùy theo mỗi ghost, dựa trên Tile của Pac-Man) hướng đến khi tìm đường đi  
     + TileManager.Tile currentTile: Tile hiện tại của ghost

+ TileManager.Tile GetTargetTilePerGhost(): Tìm targetTile ứng với từng tính cách của mỗi ghost trong quá trình tìm đường đi

+ public void AILogic(): Thực hiện thuật giải tìm đường đi để bắt Pac-Man

+ public void RunLogic(): Thuật hiện thuật giải tìm đường chạy trốn Pac-Man ở trạng thái Run.

* **Pac-Man**: Được quản lí qua class **PlayController.cs** với các thuộc tính và phương thức chính:

+ public float speed: Tốc độ di chuyển của Pac-Man, sẽ tăng dần theo từng level.

+ void ReadInputAndMove(): Đọc hướng đi từ bàn phím cho Pac-Man.

* **Viên thức ăn**: Được quản lí bởi class **Pacdot.cs** gồm 1 phương thức chính

+ void OnTriggerEnter2D(Collider2D other) : Xử lý va chạm giữ Pac-Man và pacdot

* **Viên năng lượng**: Được quản lí bởi class **Energizer.cs**, gồm 1 phương thức

+ void OnTriggerEnter2D(Collider2D other): Xử lý va chạm giữ Pac-Man và viên năng lượng

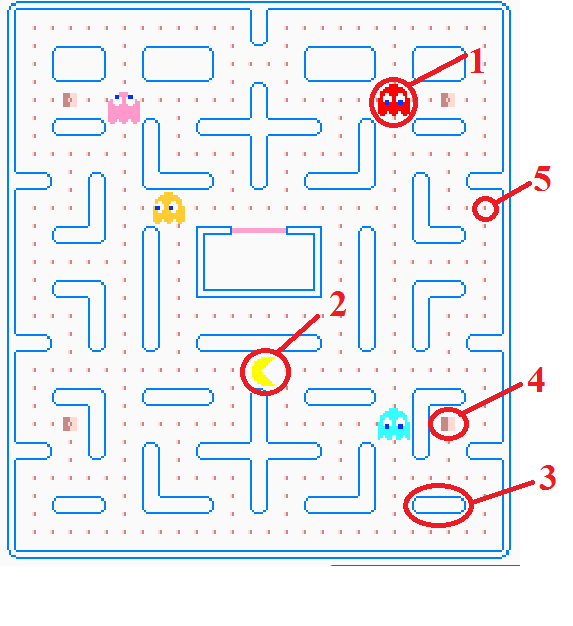
* Trạng thái bắt đầu:



* Trạng thái kết thúc:



* Ví dụ: Trạng thái Ghost truy đuổi Pac-Man và các thông số của các đối tượng



* Ghost (Số 1):
* Speed: 0.15
* Ghost : Ghost Đỏ (blinky)
* Target: Pac-Man
* Direction: Hướng xuống (down)
* State: Chase
* CurrentTile:

+ x = 22, y = 26

+ occupied = false

+ isIntersection = true

+ left, right, up, down != null (Có giá trị là 1 Tile xác định)

* TartgetTile (vị trí Pac-Man):

+ x = 15, y = 11

+ occupied = false;

+ isIntersection = false;

+ left = right != null, up = down = null;

* Pac-Man (Số 2)
* Speed: 0.2
* Tile hiện tại:

+ x = 15, y = 11

+ occupied = false;

+ isIntersection = false;

+ left = right != null, up = down = null;

* Tường cản (Số 3): Thuộc Tile có thuộc tính

+ x = 25, y = 4

+ occupied = true;

+ isIntersection = false;

+ left = right = up = down = null;

* Viên năng lượng (Số 4)

+ x = 25, y = 8;

+ occupied = false;

+ isIntersection = false;

+ right = down != null, left = up = null;

* Viên thức ăn (Số 5):
* + x = 27, y = 20;
* + occupied = false;
* + isIntersection = false;
* + left = down != null, right = up = null;

**3.3. Các vấn đề và thuật giải**

**3.3.1. Bài toán lớn: Tìm đường đi cho từng ghost đến vị trí xác định**

**Ghost đỏ**

Bài toán: Tìm đường đi ngắn nhất từ vị trí của ghost đỏ đến vị trí Pac-Man

Ý tưởng: Input: tọa độ của ghost (currentPos)

Vector3 currentPos = new Vector3(transform.position.x + 0.499f, transform.position.y + 0.499f);

Tọa độ PacMan (targetPos)

targetPos = new Vector3 (target.position.x + 0.499f, target.position.y + 0.499f);

Output: Một trong số các hướng đi có thể của ghost (left, right, up, down)

Thuật giải:

Bước 1: Xác định Tile hiện tại (currentTile) và Tile mục tiêu (targetTile)

currentTile = tiles[manager.Index ((int)currentPos.x, (int)currentPos.y)];

targetTile = tiles[manager.Index((int)targetPos.x, (int)targetPos.y)

];

Bước 2: Xác định Tile tiếp theo của ghost (nextTile) theo hướng đi

if(ghost.direction.x > 0)

nextTile = tiles[manager.Index ((int)(currentPos.x+1), (int)currentPos.y)];  
if(ghost.direction.x < 0)

nextTile = tiles[manager.Index ((int)(currentPos.x-1), (int)currentPos.y)];  
if(ghost.direction.y > 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y+1))];  
if(ghost.direction.y < 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y-1))];

Bước 3: Nếu nextTile không phải là tường và currentTile không phải là giao lộ thì chọn hướng đi tiếp theo cho ghost là hướng đi hiện tại và kết thúc thuật giải.

ghost.direction = ghost.direction;

Bước 4: Kiểm tra vị trí tiếp theo có phải là tường và vị trí hiện tại không phải là giao lộ hay không

if(nextTile.occupied && !currentTile.isIntersection)

Nếu đúng, thực hiện xử lý ngã 2 và dừng bài toán

Bước 5: Kiểm tra xem ghost có đang đứng ở giao lộ hay không, nếu điều kiện trả về là đúng thì thực hiện thuật giải A\* và kết thúc bài toán.

if(currentTile.isIntersection)

**Ghost hồng**

Bài toán: Tìm đường đi ngắn nhất từ vị trí của ghost đỏ đến vị trí đón đầu trực diện 4 bước trước Pac-Man

Ý tưởng: Input: tọa độ của ghost (currentPos)

Vector3 currentPos = new Vector3(transform.position.x + 0.499f, transform.position.y + 0.499f);

Tọa độ PacMan (targetPos)

targetPos = new Vector3 (target.position.x + 0.499f, target.position.y + 0.499f);

Output: Một trong số các hướng đi có thể của ghost (left, right, up, down)

Thuật giải:

Bước 1: Xác định Tile hiện tại (currentTile) và Tile mục tiêu (targetTile)

currentTile = tiles[manager.Index ((int)currentPos.x, (int)currentPos.y)];

dir = target.GetComponent<PlayerController>().getDir();  
targetPos = new Vector3 (target.position.x + 0.499f, target.position.y + 0.499f) + 4\*dir;  
targetTile = tiles[manager.Index((int)targetPos.x, (int)targetPos.y)];

Bước 2: Xác định Tile tiếp theo của ghost (nextTile) theo hướng đi

if(ghost.direction.x > 0)

nextTile = tiles[manager.Index ((int)(currentPos.x+1), (int)currentPos.y)];  
if(ghost.direction.x < 0)

nextTile = tiles[manager.Index ((int)(currentPos.x-1), (int)currentPos.y)];  
if(ghost.direction.y > 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y+1))];  
if(ghost.direction.y < 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y-1))];

Bước 3: Nếu nextTile không phải là tường và currentTile không phải là giao lộ thì chọn hướng đi tiếp theo cho ghost là hướng đi hiện tại và kết thúc thuật giải.

ghost.direction = ghost.direction;

Bước 4: Kiểm tra vị trí tiếp theo có phải là tường và vị trí hiện tại không phải là giao lộ hay không

if(nextTile.occupied && !currentTile.isIntersection)

Nếu đúng, thực hiện xử lý ngã 2 và dừng bài toán

Bước 5: Kiểm tra xem ghost có đang đứng ở giao lộ hay không, nếu điều kiện trả về là đúng thì thực hiện thuật giải A\* và kết thúc bài toán.

if(currentTile.isIntersection)

**Ghost cam**

Bài toán: Tìm đường đi ngắn nhất từ vị trí của ghost cam đến vị trí Pac-Man, nếu khoảng các từ ghost cam đến Pac-Man nhỏ hơn 9 ô gạch, nó sẽ chuyển hướng đến một vị trí khác có tọa độ cố định.

Ý tưởng: Input: tọa độ của ghost (currentPos)

Vector3 currentPos = new Vector3(transform.position.x + 0.499f, transform.position.y + 0.499f);

Tọa độ PacMan (targetPos)

targetPos = new Vector3 (target.position.x + 0.499f, target.position.y + 0.499f);

Output: Một trong số các hướng đi có thể của ghost (left, right, up, down)

Thuật giải:

Bước 1: Xác định Tile hiện tại (currentTile) và Tile mục tiêu (targetTile)

currentTile = tiles[manager.Index ((int)currentPos.x, (int)currentPos.y)];

targetTile = tiles [manager.Index ((int)targetPos.x, (int)targetPos.y)];  
 if (manager.distance (targetTile, currentTile) < 9) {  
                targetTile = tiles[manager.Index (0, 2)];  
         }

Bước 2: Xác định Tile tiếp theo của ghost (nextTile) theo hướng đi

if(ghost.direction.x > 0)

nextTile = tiles[manager.Index ((int)(currentPos.x+1), (int)currentPos.y)];  
if(ghost.direction.x < 0)

nextTile = tiles[manager.Index ((int)(currentPos.x-1), (int)currentPos.y)];  
if(ghost.direction.y > 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y+1))];  
if(ghost.direction.y < 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y-1))];

Bước 3: Nếu nextTile không phải là tường và currentTile không phải là giao lộ thì chọn hướng đi tiếp theo cho ghost là hướng đi hiện tại và kết thúc thuật giải.

ghost.direction = ghost.direction;

Bước 4: Kiểm tra vị trí tiếp theo có phải là tường và vị trí hiện tại không phải là giao lộ hay không

if(nextTile.occupied && !currentTile.isIntersection)

Nếu đúng, thực hiện xử lý ngã 2 và dừng bài toán

Bước 5: Kiểm tra xem ghost có đang đứng ở giao lộ hay không, nếu điều kiện trả về là đúng thì thực hiện thuật giải A\* và kết thúc bài toán.

if(currentTile.isIntersection)

**Ghost xanh**

Bài toán: Tìm đường đi ngắn nhất từ vị trí của ghost xanh đến một vị trí trên bản đồ được xác định bằng công thức:

Ý tưởng: Input: tọa độ của ghost (currentPos)

Vector3 currentPos = new Vector3(transform.position.x + 0.499f, transform.position.y + 0.499f);

Hướng đi của Pac-Man (dir )

dir = target.GetComponent<PlayerController> ().getDir ();  
 Tọa độ ghost Đỏ (blinkyPos)

Vector3 blinkyPos = GameObject.Find ("blinky").transform.position;

Output: Một trong số các hướng đi có thể của ghost (left, right, up, down)

Thuật giải:

Bước 1: Xác định Tile hiện tại (currentTile) và Tile mục tiêu (targetTile)

currentTile = tiles[manager.Index ((int)currentPos.x, (int)currentPos.y)];

Vector3 ambushVector = target.position + 2 \* dir - blinkyPos;  
targetPos = new Vector3 (target.position.x + 0.499f, target.position.y + 0.499f) + 2 \* dir + ambushVector;  
targetTile = tiles[manager.Index((int)targetPos.x, (int)targetPos.y)];

Bước 2: Xác định Tile tiếp theo của ghost (nextTile) theo hướng đi

if(ghost.direction.x > 0)

nextTile = tiles[manager.Index ((int)(currentPos.x+1), (int)currentPos.y)];  
if(ghost.direction.x < 0)

nextTile = tiles[manager.Index ((int)(currentPos.x-1), (int)currentPos.y)];  
if(ghost.direction.y > 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y+1))];  
if(ghost.direction.y < 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y-1))];

Bước 3: Nếu nextTile không phải là tường và currentTile không phải là giao lộ thì chọn hướng đi tiếp theo cho ghost là hướng đi hiện tại và kết thúc thuật giải.

ghost.direction = ghost.direction;

Bước 4: Kiểm tra vị trí tiếp theo có phải là tường và vị trí hiện tại không phải là giao lộ hay không

if(nextTile.occupied && !currentTile.isIntersection)

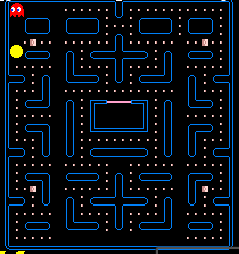
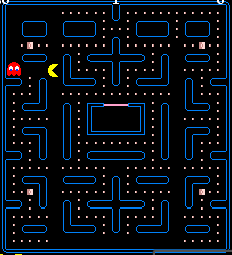
Nếu đúng, thực hiện xử lý ngã 2 và dừng bài toán

Bước 5: Kiểm tra xem ghost có đang đứng ở giao lộ hay không, nếu điều kiện trả về là đúng thì thực hiện thuật giải A\* và kết thúc bài toán.

if(currentTile.isIntersection)

**3.3.2 Giải quyết đụng tường, ngã 2**

Trường hợp ghost đang đứng ở ngã 2(ô tiếp theo là tường và ô hiện tại không phải là giao lộ) chỉ có thể một lựa chọn duy nhất để đi bước tiếp theo. Để dễ hình dung ta có hình minh họa sau:

H1 H2

Giải quyết vấn đề khi chạm vào tường

*//di chuyen qua trai, qua phai va dung wall*  
if(ghost.direction.x != 0)  
{  
 if(currentTile.down == null)    ghost.direction = Vector3.up;  
       else                          ghost.direction = Vector3.down;                      
}  
*// di chuyen len , xuong dung wall*  
else if(ghost.direction.y != 0)  
{  
     if(currentTile.left == null) ghost.direction = Vector3.right;   
       else                  ghost.direction = Vector3.left;                      
}

H1 thể hiện ghost đang di chuyển từ phải sang trái và đụng phải ngã 2, do vậy chỉ có thể đi xuống hoặc lên, nhưng đi lên là tường nên chỉ có thể đi xuống. H2 cũng tương tự. Do vậy trong trường hợp này ta không cần dùng A\*.

**3.3.3. Thực hiện thuật giải A\***

Mã giả chi tiết thuật giải A\*:

Bước 1: Khởi tạo 2 List<Tile> open và close.

List<TileManager.Tile> open = new List<TileManager.Tile> ();  
List<TileManager.Tile> close = new List<TileManager.Tile> ();

Bước 2: Tính toán giá trị g, h, f cho currentTile. Thêm currentTile vào open. Ban đầu currentTile chưa đi qua ô nào nên g = 0; f = h = khoảng cách từ currentTile tới targetTile, đồng thời lưu lại vị trí ban đầu bằng biến saveCurrentTile (mục đích sẽ đề cập ở B8).

currentTile.h = currentTile.f = manager.distance (currentTile, targetTile);  
TileManager.Tile saveCurrentTile = currentTile;  
open.Add (currentTile);

Bước 3: Thực hiện vòng lặp đến khi open rỗng thì kết thúc tìm kiếm, không tìm được lời giải.

while (open.Count != 0)

Bước 4: Tìm Tile có giá trị f nhỏ nhất trong open. Gán currentTile bằng Tile vừa tìm được.

for(int i = 0; i < open.Count; i++){  
   if (open [i].f < MIN) {  
      MIN = open [i].f;  
      currentTile = open [i];  
                        }

}

Bước 5: Kiểm tra nếu tọa độ của currentTile bằng tọa độ targetTile thì break vòng while. Ở đây ta dùng so sánh tọa độ thay vì currentTile == targetTile là vì currentTile các đại lượng g, h, f đều khác null còn targeTile thì ngược lại nêu việc so sánh không chính xác dẫn đến lặp vô cùng.

if (targetTile.x == currentTile.x && targetTile.y == currentTile.y) {  
                        *//Debug.Log ("Break");*  
                        break;  
}

Bước 6: Xóa currentTile khỏi open rồi thêm nó vào close.

open.Remove (currentTile);  
close.Add (currentTile);

Bước 7: Tìm tất cả các Tile kề (left, right, up, down) khác null cho currentTile để kiểm tra. Ở đây ta lấy ví dụ cho Tile left, các Tile kề còn lại làm tương tự.

+ Vì có 4 trường hợp nên ta tạo biến p kiểu Tile bằng với currentTile để dùng chung.

TileManager.Tile p = currentTile;

Bước 7.1: Nếu p.left != null, tức là bên trái currentTile có thể di chuyển được, không phải tường thì gán p = p.left và thực hiện các kiểm tra.

if (p.left != null)

     p = p.left;

Bước 7.2.1: Kiểm tra nếu p không tồn tại trong cả open và close (Hàm Contains() trả về true nếu có tồn tại)

if (!open.Contains (p) && !close.Contains (p))

Gán p.before = currentTile. Như đã mô tả ở trên, mỗi Tile sẽ có thuộc tính là before kiểu Tile để lưu vết lại “cha” của 1 Tile và hữu ích cho việc truy ngược tìm đường đi.

p.before = currentTile;

Tính đại lượng g – chi phí đường đi thực tế đã đi qua từ vị trí bắt đầu tới vị trí đang xét.

p.g = manager.distance(currentTile, p) + p.before.g;

Tính đại lượng h – chi phí đường đi tính theo đường chim bay từ Tile p đến targetTile.

p.h = manager.distance(targetTile, p);

Tính f = g + h rồi thêm p vào tập open.

p.f = p.g + p.h;  
open.Add (p);

Bước 7.2.2: Nếu p đã có trong open và đại lượng g đã lưu lớn hơn g hiện tại thì ta tiến hành cập nhật lại before, g, f

if (open.Contains (p) && p.g > currentTile.g + manager.distance(currentTile, p )) {  
     p.before = currentTile;  
     p.g = currentTile.g + manager.distance (currentTile, p);  
     p.f = p.g + p.h;

}

Bước 7.2.3: Nếu p tồn tại trong close và đại lượng g đã lưu lớn hơn g hiện tại thì ta tiến hành:

Xóa p khỏi close.

Cập nhật lại before, g, h, f cho p.

Thêm p vào lại tập open.

if (close.Contains (p) && p.g > currentTile.g + manager.distance(currentTile, p)) {  
              close.Remove (p);  
              p.before = currentTile;  
              p.g = manager.distance(currentTile, p) + p.before.g;  
              p.h = manager.distance(targetTile, p);  
              p.f = p.g + p.h;  
              open.Add (p);  
              }

Tương tự như vậy đối với các Tile kề right, up, down thì ta xây dựng được đầy đủ các trường hợp để thêm vào tập open.

Bước 8: Xác định hướng di chuyển

Sau khi thuật giải A\* kết thúc thì ta có được currentTile, bây giờ cũng chính là targetTile. Và do mỗi lần ghost gặp giao lộ thì nó lại thực hiện lại giải thuật A\* để tìm đường đi cho nên ta chỉ cần xác định 1 hướng di chuyển tiếp theo so với vị trí hiện tại cho ghost là đủ.

Bước 8.1: Xác định bước đi đầu tiên của thuật giải là Tile nào bằng cách dùng Tile before và quay lui.

while (saveCurrentTile.h != currentTile.before.h) {  
       currentTile = currentTile.before;  
}

Bước 8.2: Sau khi vòng while hoàn tất thì currentTile chính là 1 trong 4 Tile left, right, up, down. Ta cần xác định xác định currentTile là loại nào bằng cách dựa vào

tọa độ x, y và hướng đi cho ghost cũng chính là loại đó.

if (saveCurrentTile.x == currentTile.x) {  
        if (saveCurrentTile.y < currentTile.y) {  
             ghost.direction = Vector3.up;  
        } else  
             ghost.direction = Vector3.down;  
} else {  
        if (saveCurrentTile.x < currentTile.x) {  
             ghost.direction = Vector3.right;  
        } else {  
             ghost.direction = Vector3.left;  
        }  
}

Vậy, sau 8 bước thực hiện thuật giải A\* trên cơ sở dữ liệu đã được mô hình hóa, ta đã hoàn thành được mục đích tìm đường đi cho Ghost.

**3.3.4. Vấn đề tìm đường đi cho ghost ở trạng thái Run (Chạy trốn)**

Để tăng độ thông minh cho ghosts ở trạng thái Chạy trốn, ghost cũng cần có một cơ chế nào đó để có thể biết được con đường nào là tốt nhất để tránh bị Pac-Man tiêu diệt. Không giống như trạng thái Truy đuổi là ghost có mục tiêu rõ ràng để tìm đường đi, ở trạng thái Run ghost không có một cột mốc cụ thể nào để hướng đến vì vậy việc áp dụng thuật giải A\* trong trường hợp này là không khả thi. Tuy nhiên, ta vẫn có thể dùng đến một đại lượng trong thuật giải A\* để giải quyết vấn đề này, đó là đại lượng h (khoảng cách theo đường chim bay từ ghost đến Pac-Man).

Không giống như đại lượng h trong thuật giải A\*, h còn nhỏ thì độ tốt của hướng đi càng lớn. Đối với trường hợp này thì ngược lại, h càng lớn (càng cách xa Pac-Man) thì độ tốt càng của hướng đi đó sẽ được ưu tiên. Vì vậy ta có mã giả của thuật toán như sau:

**Bước 1**: Xác định Tile hiện tại của ghost và Pac-Man. Tile currentTile, targetTile lần lượt là Tile hiện tại của ghost và Pac-Man.

Vector3 currentPos = new Vector3(transform.position.x + 0.499f, transform.position.y + 0.499f);  
currentTile = tiles[manager.Index ((int)currentPos.x, (int)currentPos.y)];  
Vector3 targetPos = new Vector3 (target.position.x + 0.499f, target.position.y + 0.499f);  
targetTile = tiles[manager.Index((int)targetPos.x, (int)targetPos.y)];

**Bước 2** : Xác định Tile tiếp theo dựa vào hướng di chuyển hiện tại của ghost rồi lưu vào nextTile.

if(ghost.direction.x > 0)

nextTile = tiles[manager.Index((int)(currentPos.x+1), (int)currentPos.y)];  
if(ghost.direction.x < 0)

nextTile = tiles[manager.Index ((int)(currentPos.x-1), (int)currentPos.y)];  
if(ghost.direction.y > 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y+1))];  
if(ghost.direction.y < 0)

nextTile = tiles[manager.Index ((int)currentPos.x, (int)(currentPos.y-1))];

**Bước 3** : Nếu nextTile không phải là tường cản và cũng không phải giao lộ thì ghost tiếp tục hướng đi như hiện tại.

**Bước 4** : Nếu nextTile là tường cản và currentTile không phải giao lộ

if(nextTile.occupied && !currentTile.isIntersection)

thì thực hiện xử lý ngã 2

**Bước 5** : Kiểm tra nếu currentTile là giao lộ

if(currentTile.isIntersection)

**Bước 5.1** : Khởi tạo 4 giá trị float để lưu khoảng cách có thể có từ left, right, up, down của currentTile đến Pac-Man. Mặc định các giá trị bằng 0 và dist1, dist2, dist3, dist4 lần lượt để lưu khoảng cách cho up, down, left, right.

float dist1, dist2, dist3, dist4;  
dist1 = dist2 = dist3 = dist4 = 0f;

**Bước 5.2** : Tính khoảng cách theo 4 hướng nếu có bằng hàm distance() trong lớp TileManager.

if (currentTile.up != null && !currentTile.up.occupied && !(ghost.direction.y < 0) )           
dist1 = manager.distance(currentTile.up, targetTile);  
if(currentTile.down != null && !currentTile.down.occupied && !(ghost.direction.y > 0) )       
dist2 = manager.distance(currentTile.down, targetTile);  
if(currentTile.left != null && !currentTile.left.occupied && !(ghost.direction.x > 0) )       
dist3 = manager.distance(currentTile.left, targetTile);  
if(currentTile.right != null && !currentTile.right.occupied && !(ghost.direction.x < 0) )      
dist4 = manager.distance(currentTile.right, targetTile);

**Bước 5.3**: So sánh 4 giá trị dist1, dist2, dist3, dist4 để tìm giá trị lớn nhất lưu vào biến max

    float max = Mathf.Max(dist1, dist2, dist3, dist4);

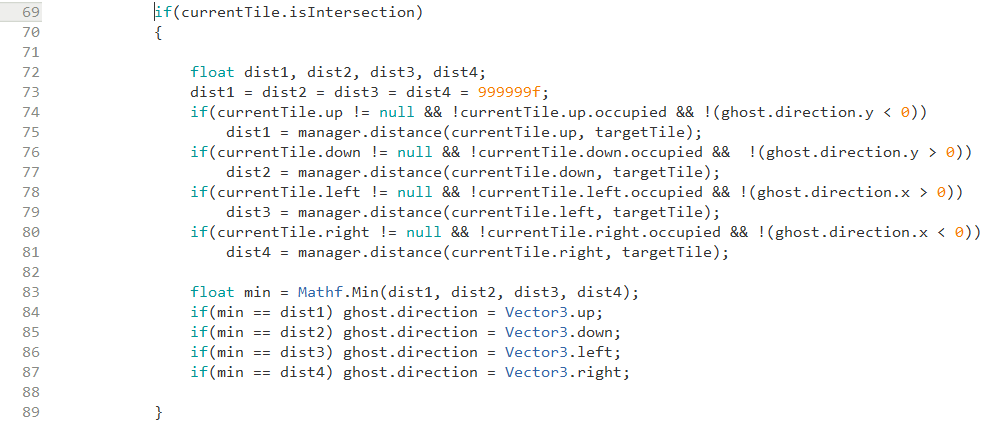
So sánh max với 4 khoảng cách để chọn hướng đi và kết thúc thuật toán.

if(max == dist1) ghost.direction = Vector3.up;  
if(max == dist2) ghost.direction = Vector3.down;

if(max == dist3) ghost.direction = Vector3.left;  
if(max == dist4) ghost.direction = Vector3.right;

Như vậy với việc sử dụng đại lượng h của giải thuật A\* ta đã có thể dễ dàng tìm hướng đi tốt nhất cho ghost khi chạy trốn, góp phần tối ưu sự thông minh cho ghosts.

**3.3.5 So sánh thuật giải cũ với thuật giải A\* tự thiết kế**

Thuật giải tìm đường đi cho ghosts cũ (Có sẳn trong source, giờ đã được thay bằng A\*) thực chất là một thuật toán đơn giản dựa trên đại lượng h trong thuật giải Heuristic, giống như thuật toán Chạy trốn đã trình bày ở trên. Nhưng thay vì chọn giá trị lớn nhất trong bốn giá trị dist1, dist2, dist3, dist4 như trong thuật toán chạy trốn, ở đây ta tìm giá trị nhỏ nhất, vì mục tiêu là tìm đường đường đi ngắn nhất tới Pac-Man.

\**Code cũ trong source (hiện tại đã được thay thế bằng A\*)*

Ta xét trường hợp khi ghost ở giao lộ, vì ở các vị trí khác, ghost di chuyển giống y hệt thuật giải A\* đã trình bày ở trên. Ta thấy, mỗi khi tại giao lộ, ghost chỉ xét 4 vị trí gần kề (left, right, up, down nếu có), sau đó tính và so sánh khoảng cách từ bốn vị trí đó đến vị trí của Pac-Man, từ 4 vị trí này rồi chọn hướng đi theo giá trị nhỏ nhất và di chuyển ghost ngay theo hướng đó

Đánh giá, so sánh với giải thuật A\*:

\* Ưu điểm:

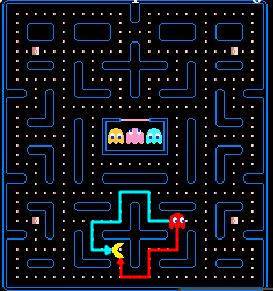
- Cài đặt code đơn giản

- Thời gian thực hiện giải thuật nhanh

\* Khuyết điểm:

- Không thông minh, không tìm được đường đi ngắn nhất trong nhiều trường hợp

**\*Ví dụ:** Đặt trường hợp Pac-Man và ghost đứng ở 2 vị trí như hình dưới, Pac-Man sẽ đứng yên một chỗ cho đến khi bị ghost Đỏ tấn công

****

* Tọa độ ghost Đỏ: (19, 8), đang ở giao lộ
* Tọa độ Pac-Man: (13, 5)

Vị trí ghost đỏ là ngã ba nên có 3 Tile kề là: left(18, 8) ; right(20, 8); down(19, 7) , tính toán các khoảng cách ta có:

+ Khoảng cách từ Tile left tới Pac-Man: 34

+ Khoảng cách từ Tile down tới Pac-Man: 40

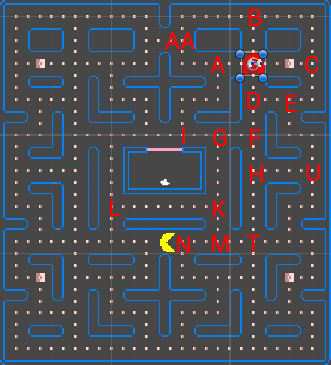
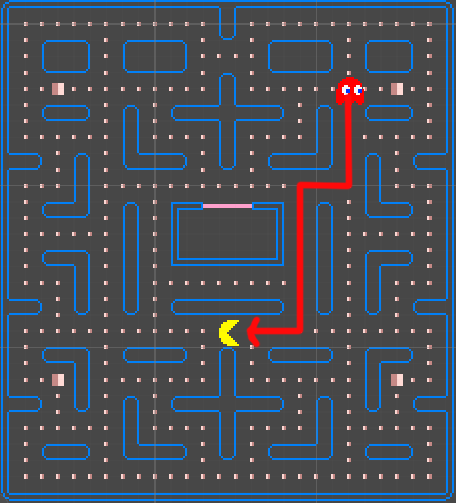
+ Khoảng cách từ Tile right tới Pac-Man: 58

Theo như thuật giải cũ thì ghost sẽ chọn hướng đi qua trái (khoảng cách 34), và tiếp tục như vậy, ta có đường đi màu xanh như hình vẽ là đường đi của ghost. Ngược lại, với thuật giải A\* thì ghost sẽ đi xuống, và đường đi là đường màu đỏ trong hình. Rõ ràng bằng mắt thường ta cũng có thể nhận ra được, đường đi màu đỏ là ngắn hơn so với đường đi xanh. Đây là 1 trong những trường hợp cho thấy khuyết điểm của thuật giải cũ.

**3.4. Ví dụ minh họa thuật toán/thuật giải**

***\*****Nếu coi mỗi Tile là một đỉnh đồ thị thì quá trình chạy tay thuật toán rất là dài dòng, vì vậy, ta xem mỗi giao lộ là một đỉnh đồ thị và xét vị trí của Pac-Man cũng là ở giao lộ (Vẫn đảm bảo tính đúng đắn của thuật giải giống trong game).*

Ví Dụ 1:

Tọa độ liên quan, hàm lượng giá:

(chú thích: với o là đỉnh bắt đầu, N là đỉnh đích)

O(22; 26) g(O) = 0; h(O) = 16.2; F(O) = 16.2

A(19; 26) g(A) = 3; h(A) = 15.3; f(A) = 18.3

AA(16; 28) g(AA) = 8; h(AA) = 17; f(AA) = 25

I(16; 20) g(I) = 12; h(I) = 9; f(I) = 21

B (22; 30) g(B) = 4; h(B) = 19.9; f(B) = 23.9

C(27; 26) g(C) = 5; h(C) = 18.6; f(C) = 23.6

D(22; 23) g(D) = 3; h(D) = 13.4; f(D) = 16.4

E(25; 23) g(E) = 6; h(E) = 15; f(E) = 21

F(22; 20) g(F) = 6; h(F) = 10.8; f(F) = 16.8

G(19; 20) g(G) = 9; h(G) = 9.5; f(G) = 18.5

H(22; 17) g(H) = 9; h(H) = 8.5; f(H) = 17.5

U(27; 17) g(U) = 14; h(U) = 12.5; f(U) = 26.5

T(22; 11) g(T) = 15; h(T) = 6; f(T) = 21

K(19; 14) g(K) = 13; h(K) = 4.2; f(K) = 17.2

L(10; 14) g(L) = 22; h(L) = 6.7; f(L) = 28.7

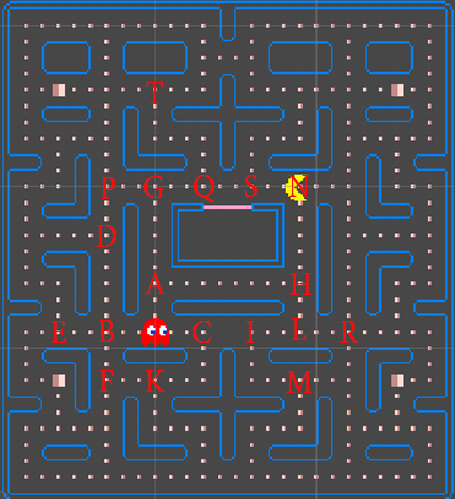
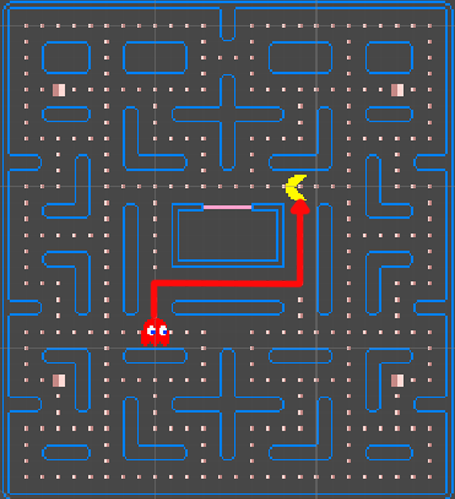
M(19; 11) g(M) = 16; h(M) = 3; f(M) = 19

N(16; 11) g(N) = 19; h(N) = 0; f(N) = 19

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lần lặp | Đỉnh đang xét P | Các đỉnh kề với đỉnh đang xét | Open | Close |
| 0 |  |  | O(16.2) |  |
| 1 | O | A, B, C, D | A(18.3), B(23.9), C(23.6), D(16.4) | O |
| 2 | D | E, F | A(18.3), B(23.9), C(23.6), E(21), F(16.8) | D |
| 3 | F | G, H | A(18.3), B(23.9), C(23.6), E(21), G(18.5), H(17.5) | F |
| 4 | H | U, T | A(18.3), B(23.9), C(23.6), E(21), G(18.5), U(26.5), T(21) | H |
| 5 | A | AA, I | B(23.9), C(23.6), E(21), G(18.5), U(26.5), T(21), AA(25), I(21) | A |
| 6 | G | I, K | B(23.9), C(23.6), E(21), U(26.5), T(21), AA(25), I(21), K(17.2) | G |
| 7 | K | L, M | B(23.9), C(23.6), E(21), U(26.5), T(21), AA(25), I(21), L(28.7), M(19) | K |
| 8 | M | N, T | B(23.9), C(23.6), E(21), U(26.5), T(21), AA(25), I(21), L(28.7), N(19) | M |
| 9 | N |  |  |  |

Đồ thị VD1:

Ví Dụ 2:

Tọa độ các đỉnh liên quan, hàm lượng giá:

O(10; 11) g(O) = 0; h(O) = 12.7; f(O) = 12.7

A(10; 14) g(A) = 3; h(A) = 10.8; f(A) = 13.8

B(7; 11) g(B) = 3; h(B) = 9.2; f(B) = 12.2

C(13; 11) g(C) = 3; h(C) = 10.8; f(C) = 13.8

D(7; 17) g(D) = 9; h(D) = 12.4; f(D) = 21.4

E(4; 11) g(E) = 6; h(E) = 17.5; f(E) = 23.5

F(7; 8) g(F) = 6; h(F) = 17; f(F) = 23

G(10; 20) g(G) = 9; h(G) = 9; f(G) = 18

H(19; 14) g(H) = 12; h(H) = 6; f(H) = 18

I(16; 11) g(I) = 6; h(I) = 9.5; f(I) = 15.5

K(10; 8) g(K) = 9; h(K) = 15; f(K) = 24

L(19; 11) g(L) = 9; h(L) = 9; f(L) = 18

M(19; 8) g(M) = 12; h(M) = 12; f(M) =24

P(7; 20) g(P) = 12; h(P) = 12; f(P) = 24

Q(13; 20) g(Q) = 12; h(Q) = 6; f(Q) = 18

N(19; 20) g(N) = 18; h(N) = 0; f(N) = 18

R(22; 11) g(R) = 12; h(R) = 9.5; f(R) = 21.5

S(16; 20) g(S) = 15; h(S) = 3; f(S) = 18

T(10; 26) g(T) = 21; h(T) = 10.8; f(T) = 31.8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lần lặp | Đỉnh đang xét P | Các đỉnh kề với đỉnh đang xét | Open | Close |
| 0 |  |  | O(12.7) |  |
| 1 | O | A, B, C | A(13.8), B(12.2), C(13.8) | O |
| 2 | B | D, E, F | A(13.8), C(13.8), D(21.4), E(23.5), F(23) | O, B |
| 3 | A | G, H | C(13.8), D(21.4), E(23.5), F(23), G(18), H(18) | O, B, A |
| 4 | C | K, I | D(21.4), E(23.5), F(23), G(18), H(18), K(24), I(15.5) | O, B, A, C |
| 5 | I | L, M | D(21.4), E(23.5), F(23), G(18), H(18), K(24), L(18), M(24) | O, B, A, C, I |
| 6 | G | P, Q | D(21.4), E(23.5), F(23), H(18), K(24), L(18), M(24), P(24), Q(18) | O, B, A, C, I, G |
| 7 | H | A, N | D(21.4), E(23.5), F(23), K(24), L(18), M(24), P(24), Q(18), N(18) | O, B, A, C, I, G, H |
| 8 | L | H, R | D(21.4), E(23.5), F(23), K(24), M(24), P(24), Q(18), N(18), R(21.5) | O, B, A, C, I, G, H, L |
| 9 | Q | S, T | D(21.4), E(23.5), F(23), K(24), M(24), P(24), N(18), R(21.5), S(18), T(31.8) | O, B, A, C, I, G, H, L, Q |
| 10 | N |  |  |  |

Đồ thị VD2:

**CHƯƠNG IV: ỨNG DỤNG**

**4.1. Giới thiệu chương trình ứng dụng**

**4.1.1. Yêu cầu:**

* OS: Window XP/Vista/7/8/10
* CPU: Intel Pentium D or AMD Athlon 64 (K8) 2.6 GHz.
* RAM: 128MB - GPU: HD Graphic
* Free Disk space: 200MB

**4.1.2. Giao diện trò chơi:**

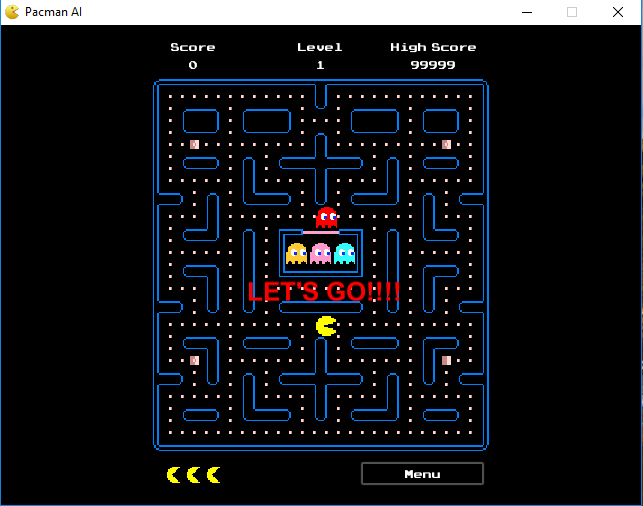
- Màn hình Home khi vào game. Play để bắt đầu chơi game, Exit thoát khỏi ứng dụng: 

- Giao diện chính để bắt đầu chơi game:

+ Phần trên là hiển thị các thông tin về điểm (Score), cấp độ (Level), điểm cao nhất (High Score).

+ Góc trái bên dưới gồm 3 icon Pac-Man thể hiện số lượt chơi còn lại.

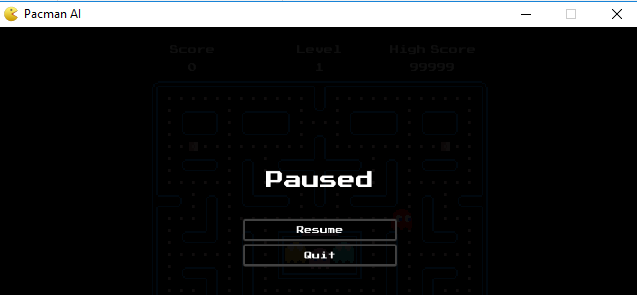
+ Góc phải bên dưới là nút Menu dùng để tạm dừng trò chơi hoặc muốn thoát khỏi game đang chơi.



- Giao diện khi nhấn nút Menu góc phải:

+ Chọn Resume nếu muốn quay lại trò chơi, mọi đối tượng trong game đều như cũ trước khi tạm dừng.

+ Chọn Quit để trở về màn hình Home.



**4.1.3. Cách sử dụng trò chơi.**

Người chơi dùng 4 phím mũi tên trên bàn phím để di chuyển Pac-Man. Tương ứng với mỗi phím là mỗi hướng đi cho Pac-Man và chỉ cần nhấn thả 1 lần duy nhất để Pac-Man di chuyển không cần phải nhấn giữ. Pac-Man sẽ chỉ dừng lại khi gặp tường cản.

**4.2 Cài đặt**

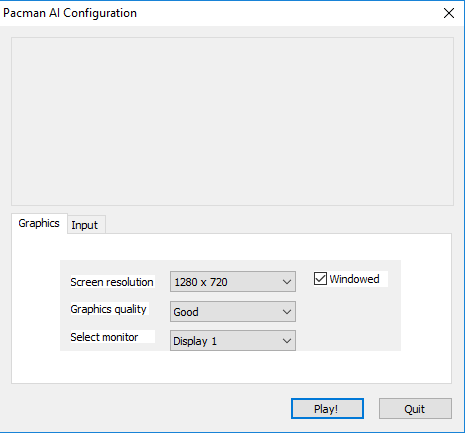
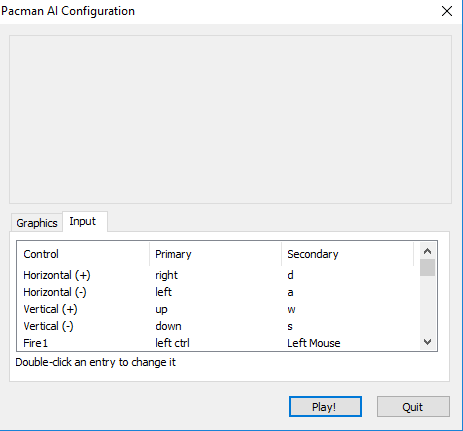
**-** Ngôn ngữ sử dụng : C#.

- Game Engine: Unity v5.6.0.f3 Personal (64 bit)

- IDE: MonoDevelop (Tích hợp sẵn trong bộ Unity).

* Hướng dẫn cài đặt chương trình từ file cài đặt

Bộ cài đặt game gồm 2 phần: File cài đặt pacman.exe và folder pacman\_Data

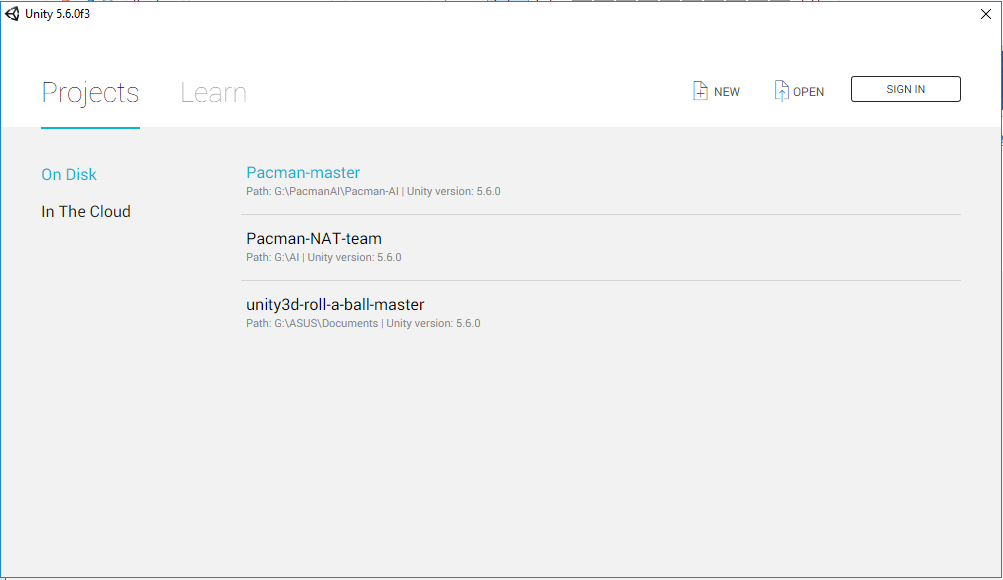
 

- Double-click vào file pacman.exe sẽ xuất hiện hộp thoại cài đặt như bên trên. Nhấn Play để bắt đầu vào game. Tùy vào cấu hình máy sẽ chọn lại option cho phù hợp ở các mục Screen Solution, Graphics quality hoặc monitor. Nếu muốn lựa chọn phím di chuyển khác thì có thể thay đổi ở tab Input.

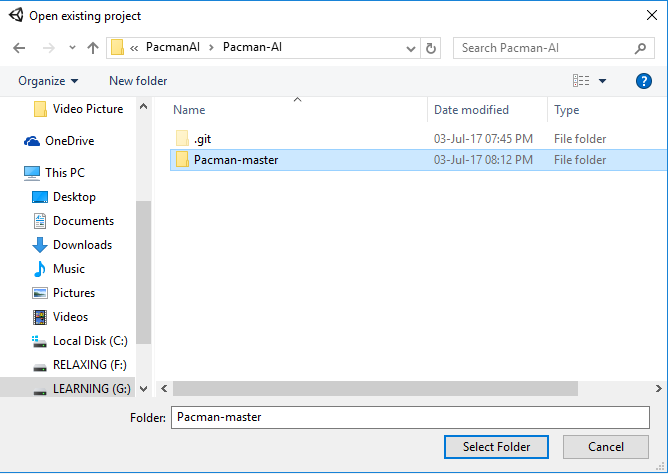
* Hướng dẫn chạy game từ source code

Bước 1: Cài đặt bộ công cụ Unity v5.6.0.f3 Personal (64 bit)

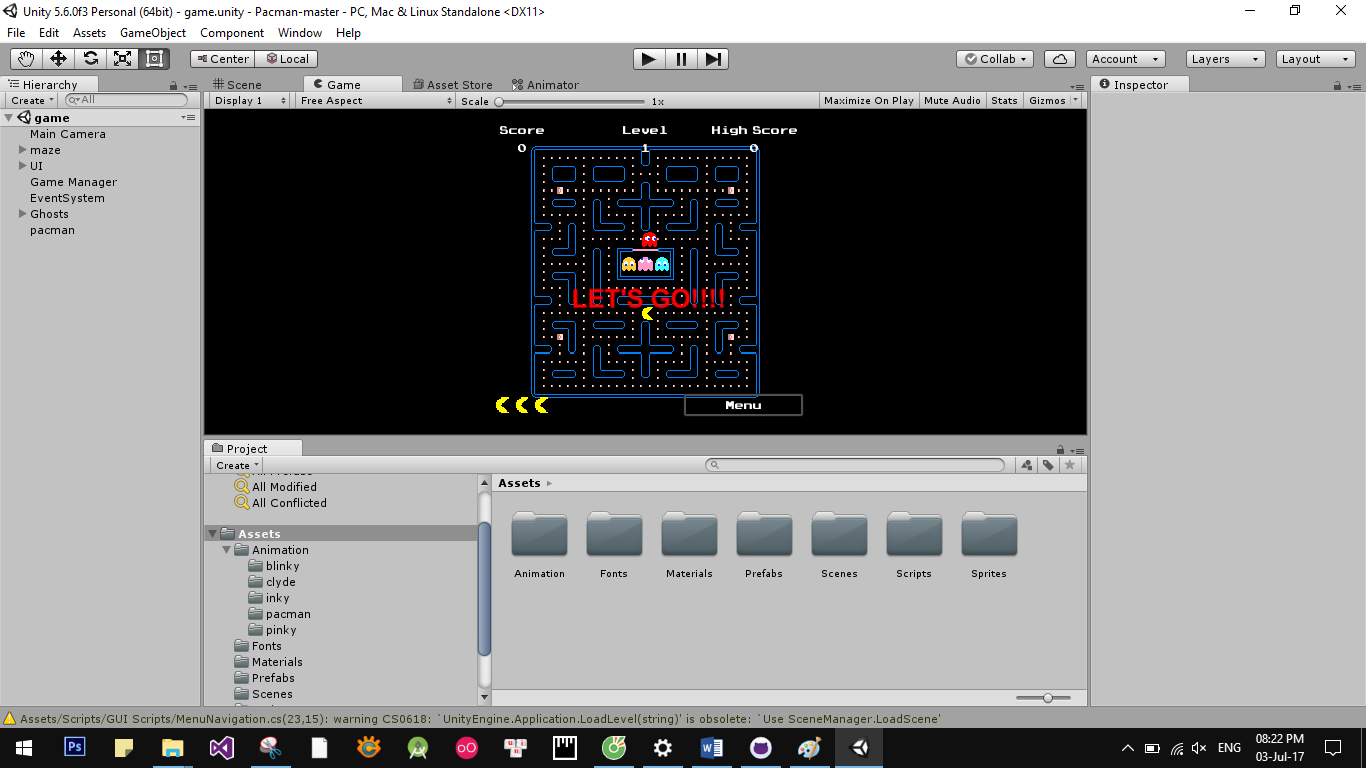
Bước 2: Khởi động Unity, từ màn hình chính chọn OPEN



Bước 3: Trỏ đường dẫn tới thư mục chứa source code có tên Pacman-master. Chọn Select folder để mở project.

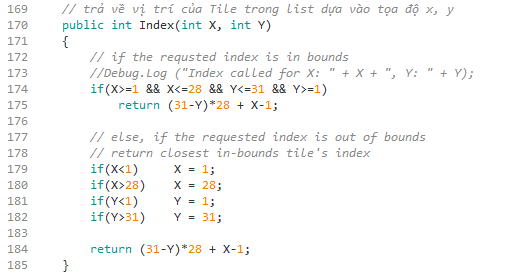


Bước 4: Sau khi Unity load xong project nhấn Play để chơi game

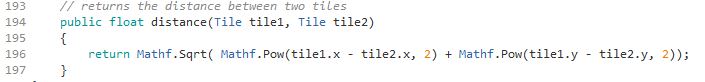


**4.3. Một số hàm xử lý chính**

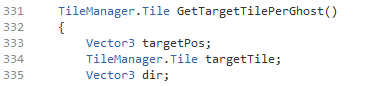
* Cài đặt class **TileManager.cs**
* **Phương thức Index(int X, int Y):** Trả về vị trí của 1 Tile trong List<Tile> đã khởi tạo dựa vào tọa độ x, y truyền vào. Đây là method để tìm Tile hiện tại cho ghosts và Pac-Man. Vì tọa độ x, y các đối tượng trong game chỉ giới hạn trong khoảng 1 ≤ x ≤ 28, 1 ≤ y ≤ 31 vì vậy ta bắt điều kiện cho 2 tham số x, y truyền vào, tránh bị lỗi “Out of Bound” .



* **Phương thức distance(Tile tile1, Tile tile2):** Trả về khoảng cách giữa 2 Tile dựa vào tọa độ x, y. Công thức được tính dựa trên công thức tính khoảng cách 2 điểm trên mặt phẳng tọa độ Oxy.



* Cài đặt class **AI.cs**
* **Phương thức GetTargetTilePerGhost ()**: Xác đinh targetTile cho từng ghost dựa trên tính cách của mỗi ghost đã mô tả. Hàm trả về giá trị kiểu Tile.



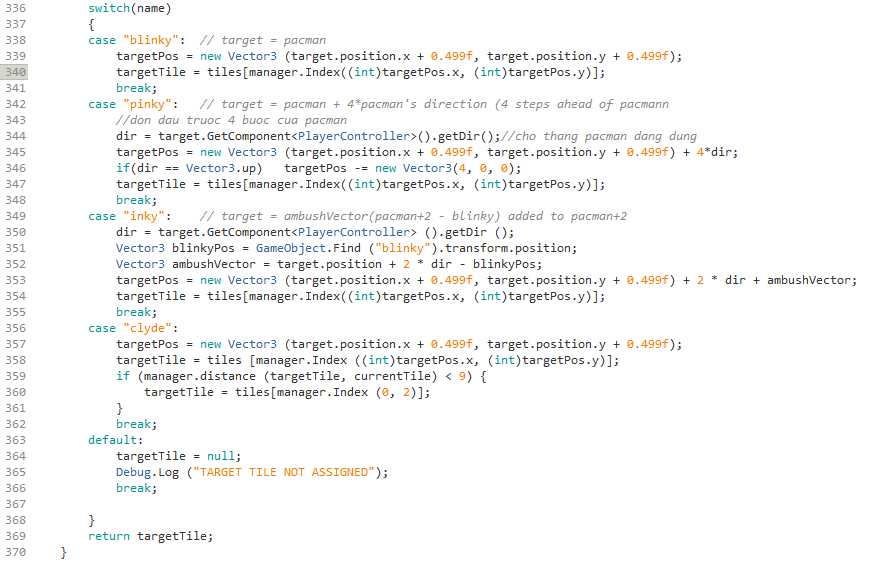
Đầu tiên ta có 3 biến local:

+ targetPos: Lưu vị trí tọa độ hiện tại của Pac-Man, từ đó có thể tìm ra được Tile hiện tại dựa vào phương thức Index(int x, int y) của class TileManager.

+ targetTile: Giá trị Tile sẽ trả về sau khi function được thực hiện

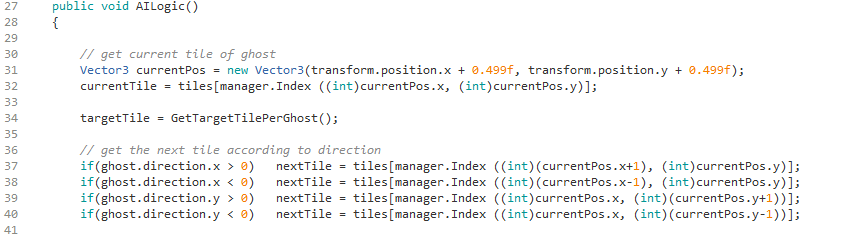
+ dir: Biến lưu trữ hướng đi hiện tại của Pac-Man, dùng để xác định targetTile cho ghost Xanh và ghost Hồng.

Vì phương thức GetTargetTilePerGhost () được dùng chung cho 4 ghost, nên ta dùng switch(name) để tính toán targetTile.

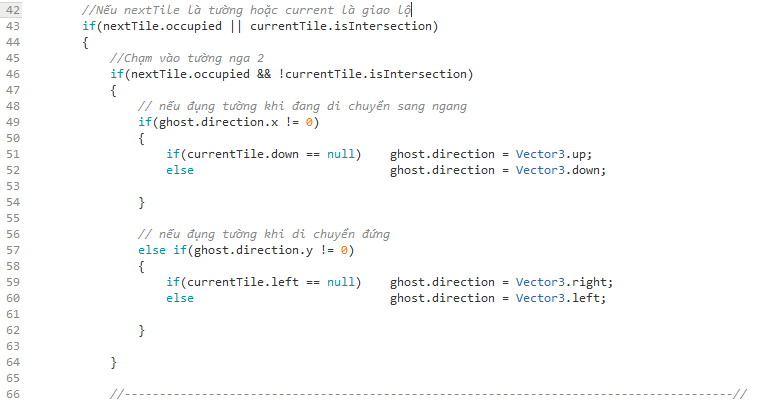


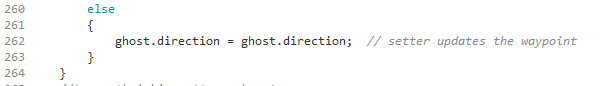
Ứng với mỗi ghost ta có cách tính toán riêng để có được những tính cách như đã mô tả. Ta chỉ cần chú ý tới con số 0.499f. Mục đích cộng thêm 0.499f mỗi khi xác định tọa độ Pac-Man là để làm tròn lên.

* **Phương thức AILogic():** Phương thức tìm đường đi cho ghost ở trạng thái Chase có sử dụng thuật giải A\*, giá trị trả về kiểu void. Hàm được gọi trong class GhostMove



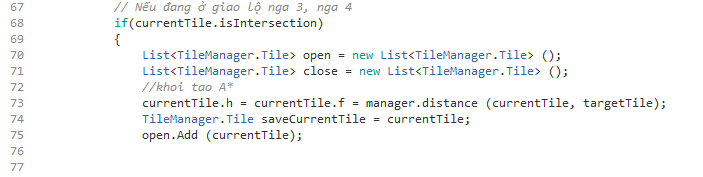
Từ dòng 31 🡪 40: Xác định Tile hiện tại của ghost, targetTile mà ghost đang hướng đến (sử dụng hàm GetTargetTilePerGhost() đã nói ở trên) và nextTile – Tile tiếp theo của ghost dựa trên hướng di chuyển.



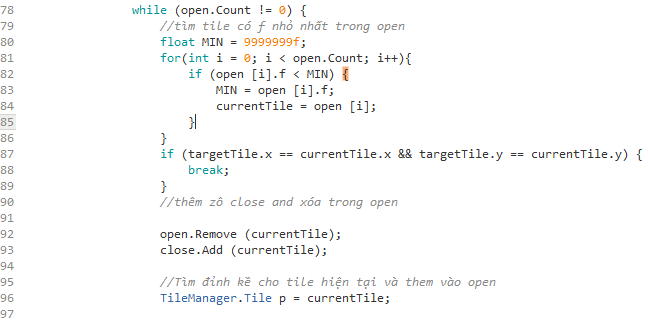


Dòng 43 kiểm tra nếu nextTile là tường hoặc currentTile là giao lộ thì mới thực hiện các bước tìm đường đi. Ngược lại, thì hướng đi tiếp theo là hướng đi hiện tại và kết thúc function (Dòng 260).

Dòng 46 🡪 64, nếu va vào ngã 2 ghost sẽ chọn hướng đi dựa vào hướng đang di chuyển và ngã rẽ sẳn có ở ngã 2.

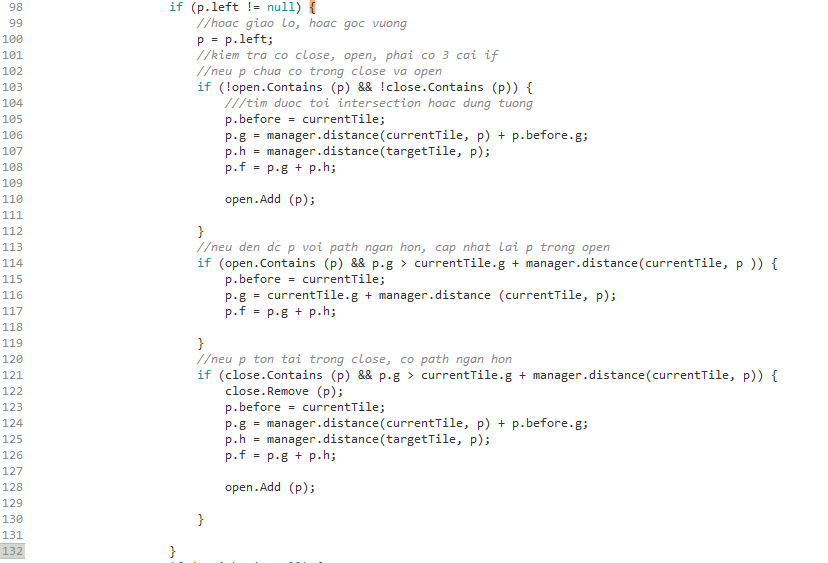
 Nếu ghost đúng giao lộ thì thực hiện giải thuật A\* tìm đường đi theo đúng như mã giả đã trình bày. Lưu ý dùng một biến saveCurrentTile lưu lại currentTile hiện tại để tìm hướng đi sau khi chạy xong thuật giải A\*

Giải thuật sẽ dừng nếu tập open rỗng (không tìm được đường đi) hoặc khi tọa độ currentTile và targetTile trùng nhau (Dòng 87 🡪 89, tìm được đường đi).



Gán Tile p = currentTile để tìm tất cả các Tile kề (left, right, up, down) có thể có, rồi thực hiện các bước kiểm tra theo đúng giải thuật A\*.

Kiểm tra Tile p.left nếu khác null, tức là currentTile có 1 Tile tồn tại ở bên trái, ta gán p = p.left rồi kiểm tra 3 điều kiện if và thực hiện theo đúng giải thuật A\* ứng với 3 điều kiện đó nếu đúng:

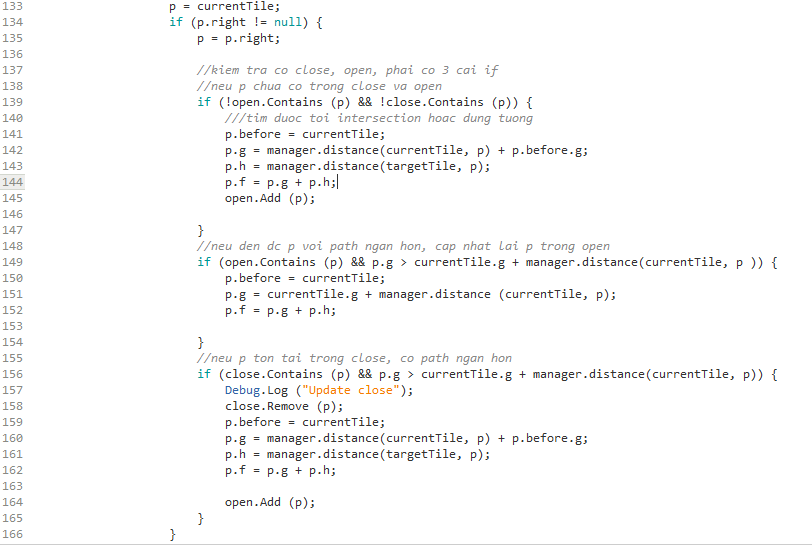


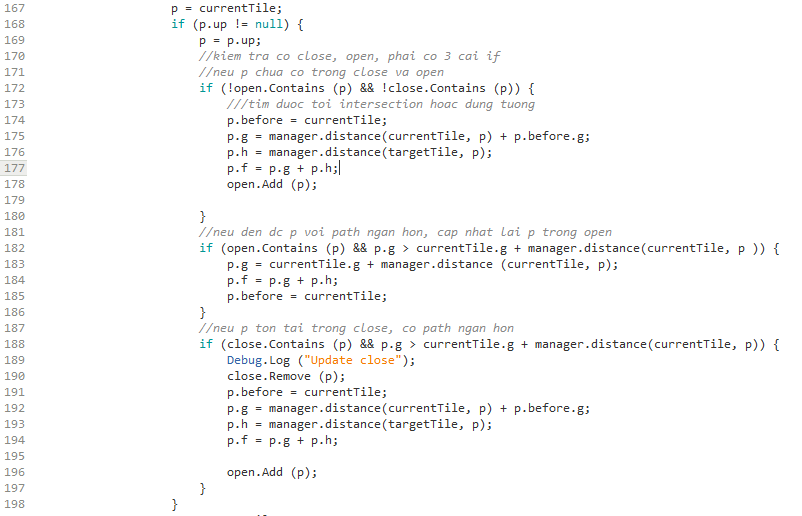
**+** Nếu p chưa có trong close và open (Dòng 103)

+ Nếu p đã có trong open nhưng đại lượng g hiện tại có giá trị nhỏ hơn giá trị cũ(Dòng 114)

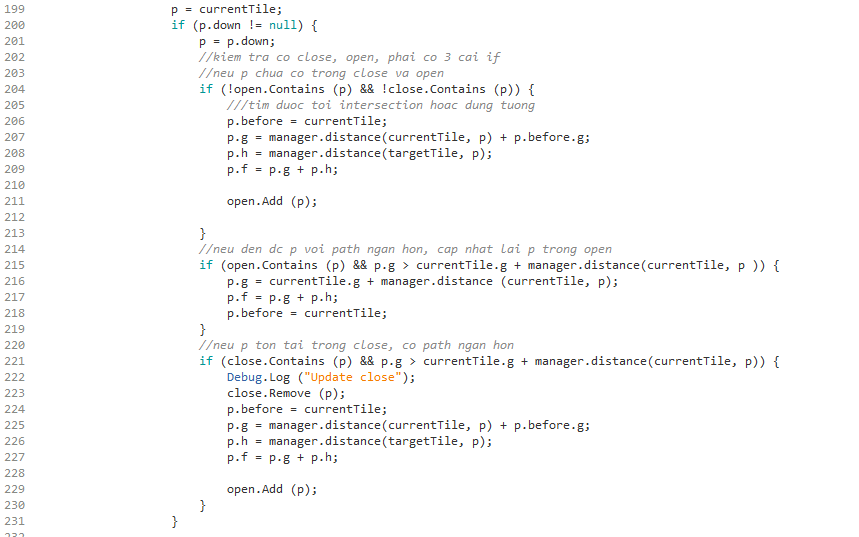
+ Nếu p đã có trong close nhưng đại lượng g hiện tại có giá trị nhỏ hơn giá trị cũ (Dòng 121)

Tương tự như vậy, đối với right, up, down của currentTile, tuy nhiên ta phải gán lại giá trị cho p (Dòng 133) vì giá trị của p đã thay đổi sau khi kiểm tra p.left (Dòng 100)

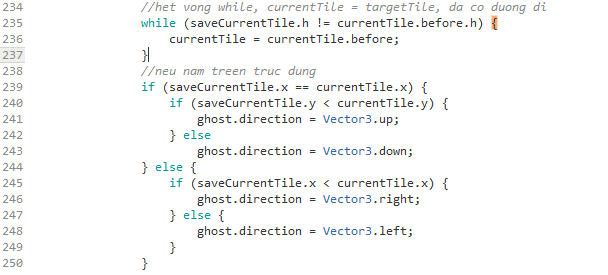


 Kiểm tra Tile up:

Kiểm tra Tile down:



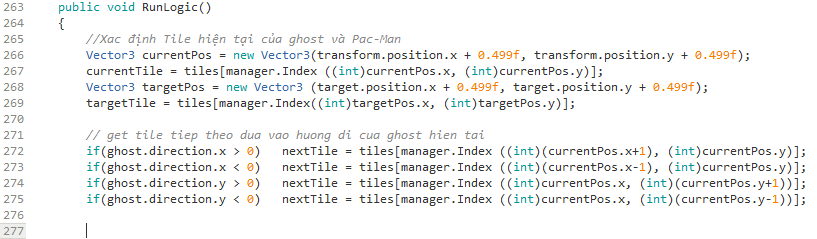
Kết thúc vòng while



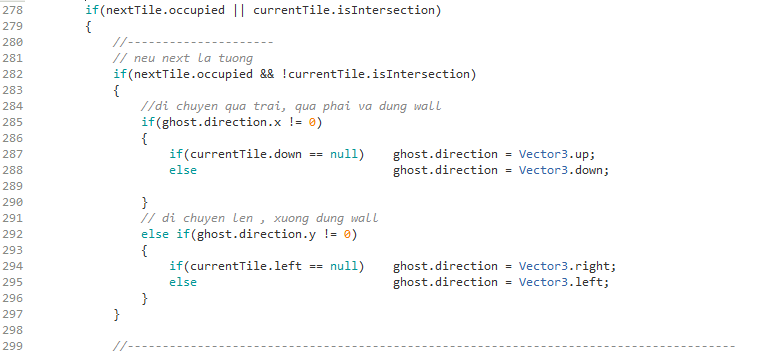
Sau khi kết thúc vòng while thì currentTile chính là targetTile. Vòng while dòng 235 có nhiệm vụ truy ngược từ targetTile đến bước đi đầu tiên so với vị trí ban đầu. Mục đích là để so sánh bước đi đó nằm bên trái hay phải hay trên hay dưới so với saveCurrentTile để chọn hướng đi tiếp theo cho ghost (Dòng 239 🡪 249)

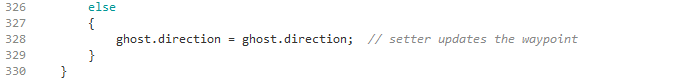
* **Phương thức RunLogic():** Thuật toán tìm đường chạy trốn Pac-Man khi bị truy đuổi, mục tiêu của function là tìm đường đi tránh Pac-Man càng xa càng tốt. Hàm được gọi trong class GhostMove

Cũng như phương thức AILogic(), bước đầu tiên là xác định currentTile , targetTile và nextTile

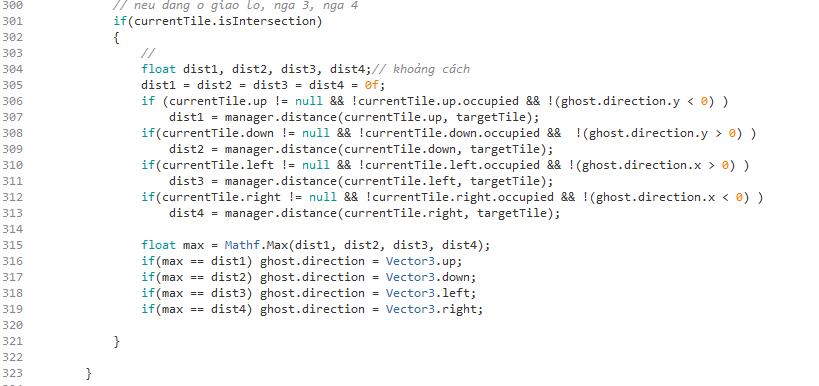
Tương tự, ta cũng kiểm tra nextTile có phải là tường cản, currentTile có phải là giao lộ hay không(Dòng 278) và cũng chọn hướng đi là hướng hiện tại (Dòng 328) nếu điều kiện if dòng 278 không đúng.

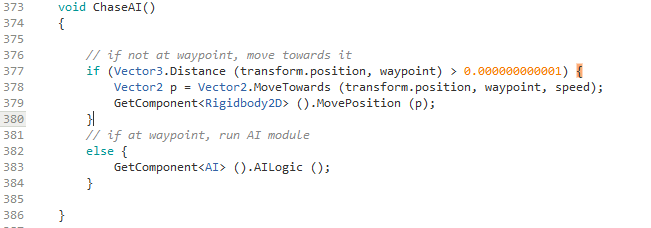
Nếu là ngã 2 thì thực hiện tương tự như AILogic() (Dòng 282 🡪 297).





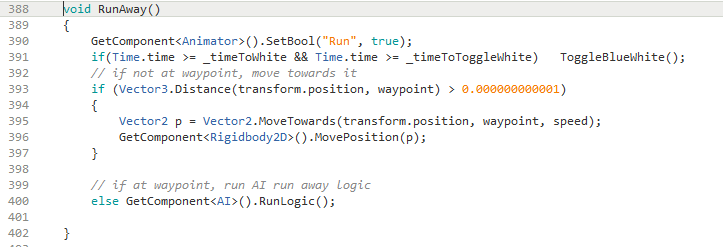
Nếu là giao lộ thì thực hiện tìm đường chạy trốn theo như mã giả



* Cài đặt class **GhostMove.cs**
* **Phương thức ChaseAI():** Được gọi khi ghost ở trạng thái Chase.

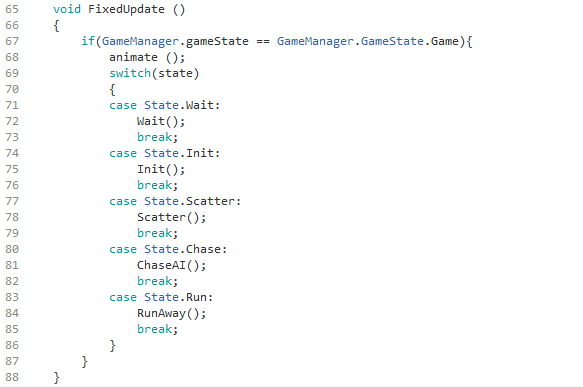
Ở điều kiện if đầu tiên, tham số tranform.position chính là tọa độ của ghost đang đứng và tọa độ này không phải lúc nào cũng là số nguyên cũng như nằm trùng với waypoint. Vì vậy, chỉ gọi hàm AILogic() khi tọa độ ghost nằm trùng với waypoint, ngược lại thì di chuyển vị trí ghost đến waypoint (Dòng 379)

* **Phương thức RunAway():** Được gọi khi ghost ở trạng thái Run



Ở trạng thái Run cũng tương tự như trạng thái Chase. Tuy nhiên ta có xử lý thêm hiệu ứng của ghost ở trạng thái này, đó là biến tất cả ghost thành màu xanh (Dòng 390) cũng như đổi trạng thái chớp nháy sang màu trắng khi sắp hết trạng thái Run (Dòng 393).

* **Phương thức FixedUpdate():** Là phương thức được gọi liên tục, như một vòng lặp while trong game. Đây là nơi bắt nguồn gọi hàm để xử lý từng trạng thái của ghost.



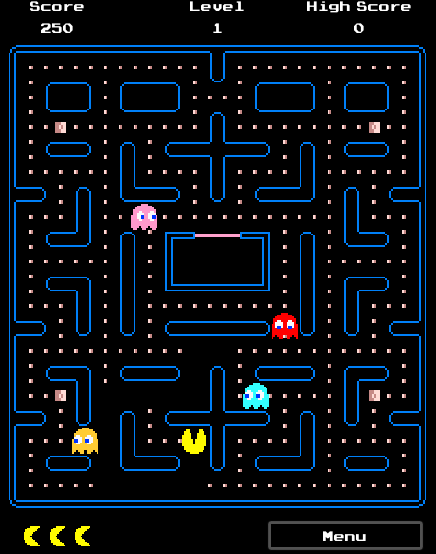
**4.4. Kết quả chạy chương trình**

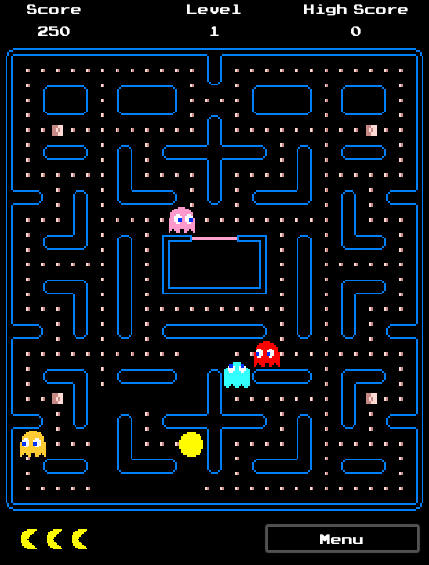
**-** Kết quả:

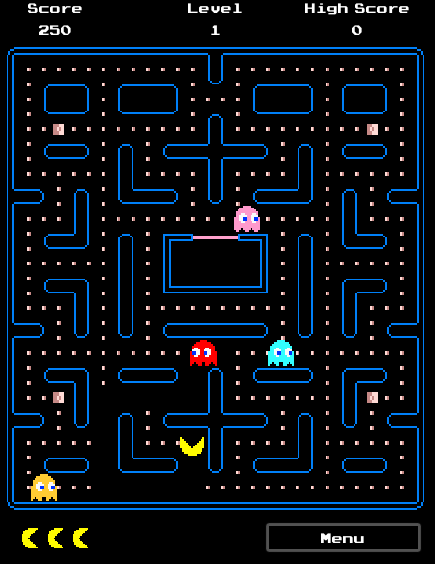
+ Ghost luôn tìm được đường đi ngắn nhất để bắt Pac-Man

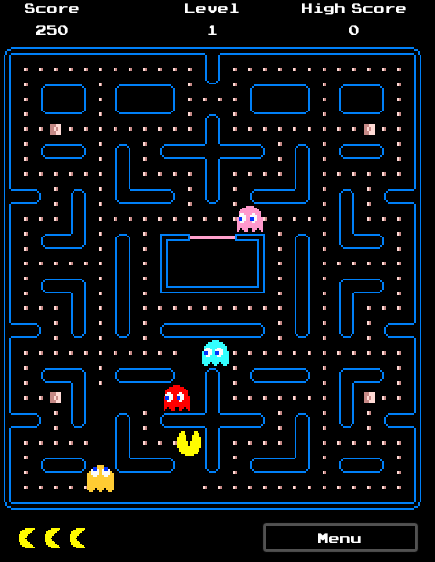
+ Ghost luôn tìm được đường đi xa nhất để trốn Pac-Man

* Ví dụ trường hợp ghost đuổi bắt Pac-Man

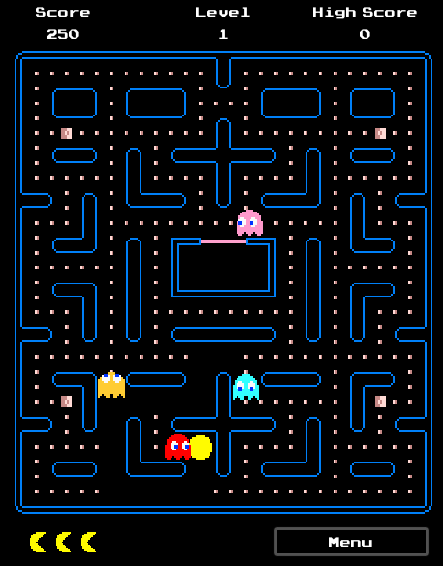


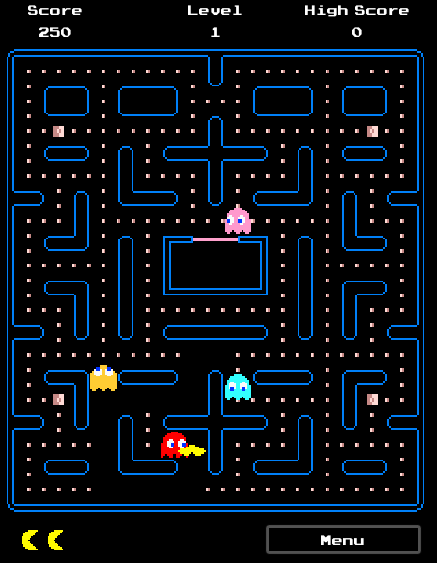




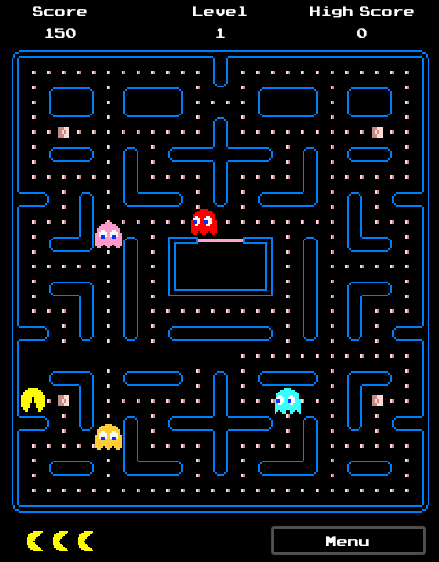


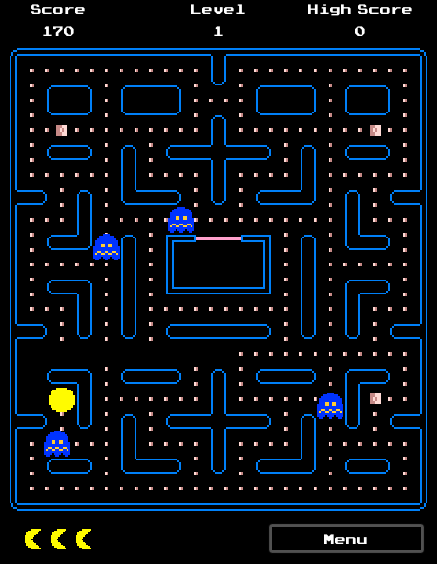


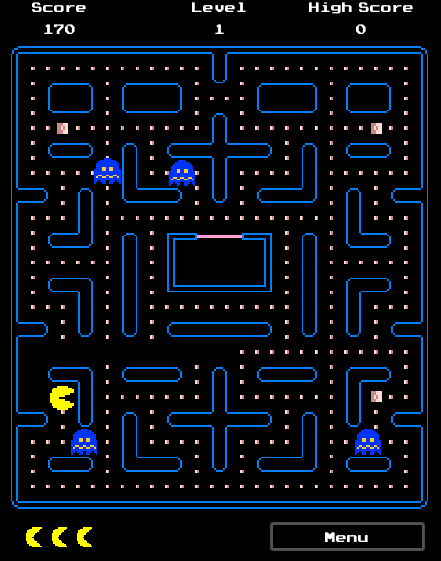


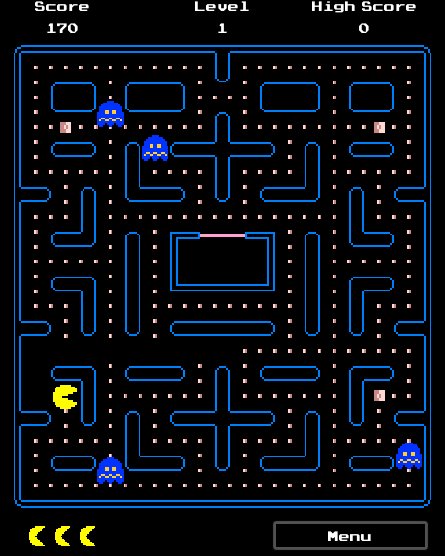


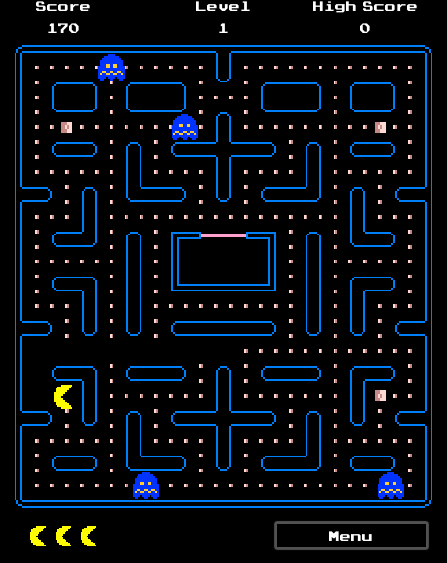
* Ví dụ trường hợp Ghost chạy trốn Pac-Man

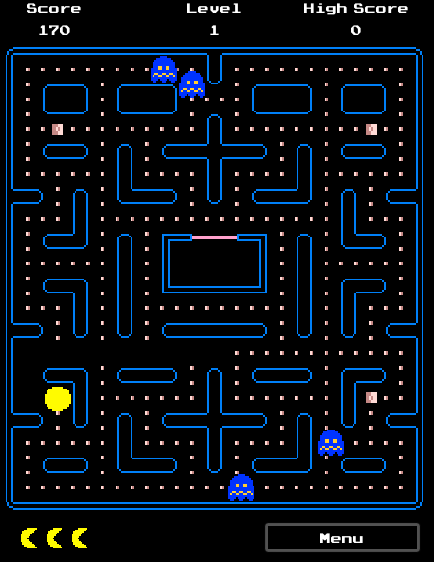


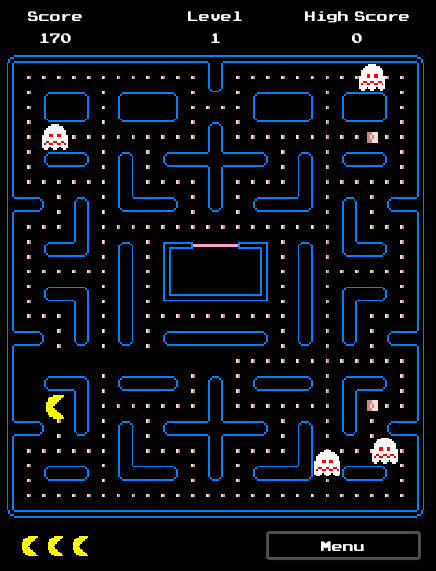














- Nhận xét:

+ Ưu điểm: Giao diện được kế thừa nên khá đẹp, mang tính chất cổ điển, thân thiện, dễ chơi.

+ Khuyết điểm: Chưa có tính năng lưu điểm cao nhất, chỉ hỗ trợ tốt trên môi trường window

**CHƯƠNG V: KẾT LUẬN**

**5.1. Kết quả đạt được**

**-** Tự nghiên cứu hoàn thành được thuật giải A\*.

- Game chạy đúng như yêu cầu, không bị giật lag do thuật giải.

- Thiết kế được thuật toán đơn giản để hổ trợ ghost trong trạng thái Chạy trốn.

**5.2. Hạn chế**

**-** Game chỉ mới chạy ổn định trên môi trường window, các nền tảng khác như android, ios, linux chưa hoàn chỉnh.

- Game chỉ có duy nhất 1 map áp dụng cho tất cả level

**5.3. Hướng phát triển**

- Phát triển thêm đa nền tảng.

- Thêm nhiều map hơn nữa, mỗi level là mỗi map.

**TÀI LIỆU THAM KHẢO**

[1], [2]:Tham khảo từ Wikipedia

Link: https://vi.wikipedia.org/wiki/Pac-Man

[3]: Tham khảo từ GameInternals.

Link: <http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior>

\*Môi trường source code về giao diện được kế thừa từ: [https://github.com/vilbeyli/Pac-Man](https://github.com/vilbeyli/Pacman)

\*\* Ý tưởng giải quyết bài toán bằng A\* và thuật toán chạy trốn hoàn toàn do nhóm tự thảo luận và tìm cách giải.