Tài liệu hướng dẫn thực hành Buổi 1
Môn : Trí tuệ nhân tạo – CT332

Khoa KHMT Trường CNTT&TT T. Đại học Cần Thơ
Giảng viên: Nguyễn Bá Diệp

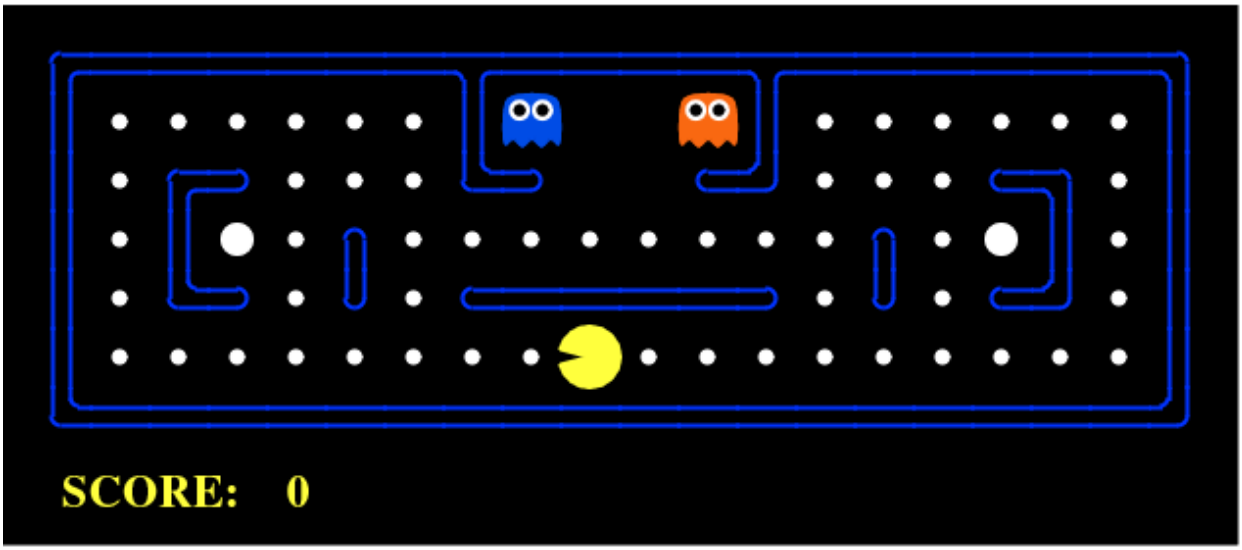
Phần 1: Giới thiệu

Trong bài thực hành này, bạn sẽ làm quen với thế giới Pac-Man. Trong các bài thực hành tiếp theo, bạn sẽ cài đặt một tác tử Pac-Man để tìm đường đi trong mê cung nhằm đến một vị trí cụ thể và thu thập thức ăn một cách hiệu quả. Bạn cũng sẽ xây dựng các thuật toán tìm kiếm tổng quát và áp dụng chúng vào các tình huống trong trò chơi Pac-Man.

Bài tập này bao gồm nhiều tập tin Python. Một số tập tin bạn cần đọc và hiểu để hoàn thành bài, số còn lại bạn có thể bỏ qua. Mã nguồn Pac-Man được phát triển bởi John DeNero và Dan Klein tại Đại học UC Berkeley cho lớp học CS188 – Trí tuệ nhân tạo.

Bước 1: Tải mã nguồn

Bạn có thể tải mã nguồn **`search`** dưới dạng file zip. Giải nén mã nguồn này, bạn sẽ thấy thư mục **`search`** chứa các tập tin cần thiết để chạy nền tảng này. Bạn có thể chạy thử với dòng lệnh từ terminal: python pacman.py . 1 cửa số trò chơi như hình 1 hiện lên tức là hệ thống hoạt động tốt.



Hình 1: giao diện trò chơi Pac-Man

Trong cửa sổ trò chơi, đây là phiên bản cơ bản của Pac-Man và bạn có thể sử dụng các phím mũi tên trên bàn phím để di chuyển nhân vật chính Pac-Man (màu vàng – hình bán nguyệt). Mục tiêu của trò chơi là ăn hết số đậu (trắng) trong thời gian ngắn nhất trong khi phải né các con ma có nhiều màu sắc khác. Các con ma không ăn đậu mà sẽ truy đuổi và ăn Pac-Man và khi đó trò chơi cũng sẽ kết thúc.

Trong thế giới Pac-Man môi trường được thiết lập với 1 mê cung (thay đổi theo tuỳ chỉnh) với hành lang tạo từ các bức tường màu xanh dương và PacMan sẽ di chuyển trong các hành lang đó. Một số hành lang có đặt các viên đậu trắng tròn nhỏ, một số hành lang có đặt các viên đậu trắng lớn là 1 cách tăng sức mạnh cho Pac-Man, khi Pac-Man ăn các viên đậu lớn thì sẽ không còn sợ các con ma trong 1 khoảng thời gian.

Bước 2: Điều khiển tác nhân (agent) Pac-Man

Bạn có thể tuỳ biến mê cung cho tác nhân Pac-Man bằng câu lệnh sau :

python pacman.py –-layout testMaze

Mê cung trên chỉ có 1 hành lang và thông thường để kiểm tra nhanh, người ta sẽ sử dụng mê cung nhỏ tinymaze:

python pacman.py –-layout tinyMaze

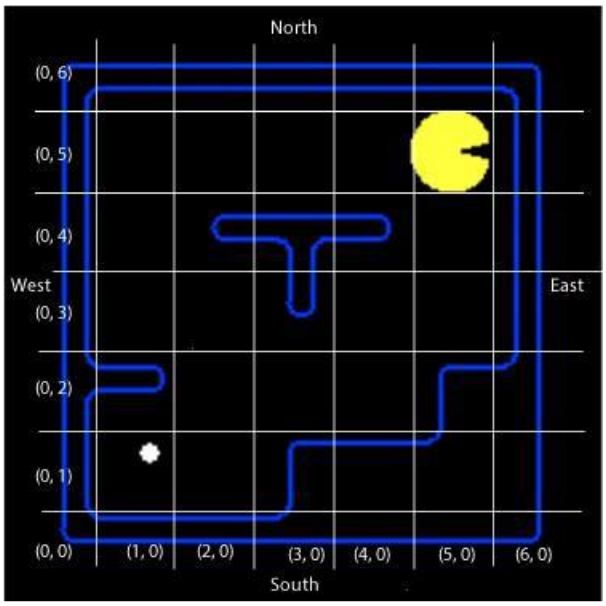
Hình 2: mê cung nhỏ

1 số tuỳ biến lựa chọn khác cho mê cung là : mediumMaze, bigMaze, openSearch …. Bạn có thể phóng to hay thu nhỏ khung nhìn mê cung bằng thông số --zoom vd:

```
python pacman.py --layout tinyMaze --zoom 2
```
python pacman.py –-layout bigMaze –-zoom 0.5

Bước 3: Toạ độ lưới và các hành động của Pac-Man

Mỗi thời điểm, Pac-Man luôn chiếm 1 ô vuông và Pac-Man luôn hướng về 1 trong 4 hướng: Đông, Tây, Nam, Bắc (East,West,South, North) vị trí Pac-Man được xác định bằng toạ độ x – y trong lưới như hình 3



Hình 3: Lưới và toạ độ Pac-Man. Pac-Man ở vị trí (5,5) và hạt đậu ở vị trí (1,1)

Hành động:

Pac-Man chỉ có các hành động sau:

'North': di chuyển 1 bước lên trên

'South': di chuyển 1 bước xuống dưới

'East': di chuyển 1 bước qua phải

'West': di chuyển 1 bước qua trái

'Stop': dừng lại, đứng yên tại chỗ

Bước 4: Làm quen với code
Bạn đã xem qua 1 số khái niệm và chức năng của chương trình, bây giờ chúng ta sẽ thao tác trên các lớp Agent.

Agent
Bạn có thể tạo các agent của riêng mình từ lớp Agent. Vd: tạo 1 agent dumb đơn giản chỉ luôn đi sang phải như code bên dưới

```
class DumbAgent(Agent):
        "An agent that goes East until it can't"
        def getAction(self, state):
                "The agent always goes East"
```

```
        return Directions.EAST
```

Các hành động có thể được cài đặt trong hàm getAction(), hàm sẽ thực thi 1 lần trong mỗi lần bộ đếm thời gian trôi đi.

Bước 5: Thử code mới

Thực thi đoạn code sau để kích hoạt DumAgent:

python pacman.py –-layout tinyMaze –-p DumbAgent

Bước 6: Tìm hiểu trạng thái GameState

Bạn có thể truy cập thông tin trong trò chơi thông qua lớp GameState, lớp này sẽ cung cấp các thông tin về trạng thái của trò chơi vd: vị trí hiện tại của agent Pac-Man, nước đi hợp lệ của Pac-Man (state.getLegalPacmanAction()), …

```
class DumbAgent(Agent):
    "An agent that goes East until it can't."
    def getAction(self, state):
    "The agent receives a GameState (defined in pacman.py)."
      print("Location: ", state.getPacmanPosition())
      print("Actions available: ", state.getLegalPacmanActions())
    if Directions.EAST in state.getLegalPacmanActions():
        print("Going East.")
        return Directions.EAST
    else:
        print("Stopping.")
        return Directions.STOP
```

Bạn có thể tìm hiểu thêm các phương thức trong lớp GameState được định nghĩa pacman.py

Bước 7: Tạo Agent di chuyển ngẫu nhiên

Hãy tạo 1 lớp mới với tên RandomAgent (trong file SearchAgents.py), agent này sẽ thực hiện hành động ngẫu nhiên. Thực thi code và quan sát xem agent này có ăn được hạt đậu không? Có thực hiện hành động phi pháp không ?(thực hiện bước đi không hợp lệ và gây lỗi chương trình?)

Bước 8: Khám phá môi trường của trò chơi

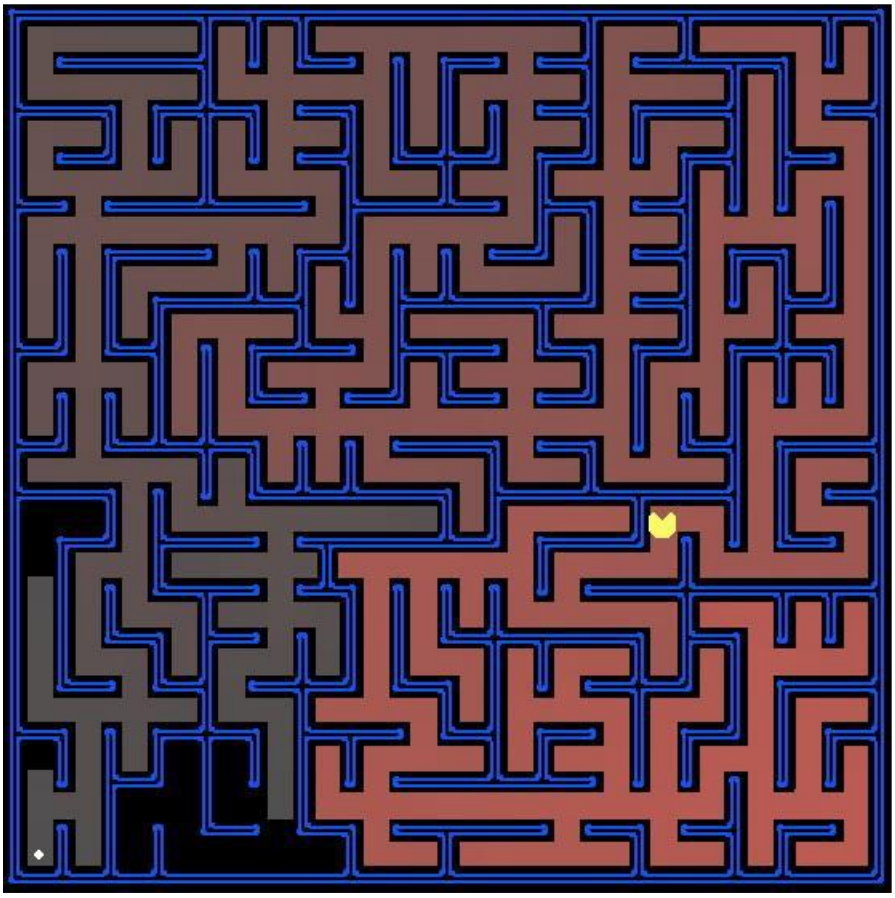
Các thông số mê cung nằm trong thư mục layouts với các tập tin dạng text có đuôi (*lay).

Trong trò chơi này có thể có nhiều agent (Pac-Man và các con ma khác), mỗi agent sẽ có 1 số index duy nhất (Pac-Man luôn có index là 0 và các con ma có index bắt đầu từ 1 về sau)

Pac-Man có thể cảm nhận được một số tiêu chí sau:
Vị trí của nó,
Vị trí của các con ma khác,
Vị trí của tường

Vị trí của hạt đậu lớn
Vị trí của các hạt đậu nhỏ
Số hạt đậu nhỏ còn lại (chưa ăn)
Xác định đã thắng hoặc thua trò chơi
Điểm số hiện thời

Phần 2: Tìm kiếm mù/ tìm kiếm không có thông tin hỗ trợ trong Pac-Man



Hình 4: tìm kiếm trong thế giới Pac-Man

1 Tìm kiếm sâu (DFS – Depth-First Search)
2 Tìm kiếm rộng (BFS – Breadth-First Search)
3 Tìm kiếm đồng nhất giá (UCS – Uniform-Cost Search)

Chúng ta hãy giúp agent Pac-Man tìm đường đi trong ma trận bằng cách cài đặt 1 số thuật toán tìm kiếm, trong phần này sẽ cập nhật các tập tin sau:

search.py # cài đặt các thuật toán tìm kiếm trong đây

searchAgents.py #Cài đặt Agent tìm kiếm trong đây – tức thực hiện các hành động theo chiến lược tìm kiếm.

Các tập tin không nên thay đổi:

Util.py #các cấu trúc dữ liệu hỗ trợ cài đặt giải thuật tìm kiếm

Pacman.py # tập tin main để chạy trò chơi.

Game.py # các logic/ luật lệ quy định trong thế giới Pac-Man. Tập tin này mô tả về AgentState, Agent, Direction, và Grid

1 Khung chương trình :

Trong tập tin searchAgent.py, hãy quan sát cách thiết lập một tác nhân tìm kiếm – SearchAgent, là kế hoạch vạch ra đường đi trong thế giới Pac-Man và thực hiện con đường ấy từng bước một. Kiểm tra code bằng dòng lệnh sau :

python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

Với các lựa chọn tác nhân tìm kiếm là SearchAgent và sử dụng chiến lược tinyMazeSearch được cài đặt trong search.py.

function UninformedSearch(problem) **returns** a list of actions

khởi tạo hàng đợi đang duyệt **frontier** sử dụng trạng thái bắt đầu của vấn đề (problem)

khởi tạo **exploredSet** là set rỗng

#Dùng vị trí của Pac-Man

#Khởi tạo các trạng thái đã duyệt qua

while frontier không rỗng **do**

chọn 1 nút lá và xoá khỏi hàng đợi đang duyệt **(frontier)**

if nút đang duyệt chứa trạng thái kết thức (goal state)

return trả về các hành động từ trạng thái bắt đầu đến trạng thái kết thúc

thêm trạng thái vào danh sách đã duyệt **exploredSet**

for mỗi nút con của trạng thái đang duyệt

if chưa duyệt qua tức không thuộc danh sách đã duyệt **exploredSet**

thêm nó vào hàng đợi đang duyệt

return list rỗng (tức không có lời giải !)

Các thuật toán DFS, BFS và UCS chỉ khác nhau các quản lý hàng đợi đang duyệt (frontier), vì thế chúng ta có thể dùng khung chương trình trên để cài đặt các thuật toán tìm kiếm để nhất quán, dễ sửa lỗi và bảo trì code. Lưu ý: Các cấu trúc dữ liệu ngăn xếp (Stack), hàng đợi (Queue), hàng đợi ưu tiên PriorityQueue đã có lưu trong util.py

2 Tìm kiếm sâu:

Cài đặt thuật toán tìm kiếm sâu depthFirstSearch trong search.py.

```
def depthFirstSearch(problem):
    """
```

```
    Search the deepest nodes in the search tree first.

    Your search algorithm needs to return a list of actions that reaches the
    goal. Make sure to implement a graph search algorithm.

    To get started, you might want to try some of these simple commands to
    understand the search problem that is being passed in:

    print "Start:", problem.getStartState()
    print "Is the start a goal?", problem.isGoalState(problem.getStartState())
    print "Start's successors:", problem.getSuccessors(problem.getStartState())
    """
    "*** YOUR CODE HERE ***"
    util.raiseNotDefined()
```

Gợi ý: Bạn có thể cần cài đặt và sử dụng ngăn xếp - util.Stack(), để sử dụng các phương thức của ngăn xếp vui lòng đọc thêm trong tập tin util.py (như .push() , .pop() …).
Sử dụng các hàm kiểm tra của vấn đề như: problem.getStartState(), problem.isGoalState(_x_), problem.getSuccessors(_x_) , với _x_ là đối số của hàm.

Khi hoàn thành thuật toán, bạn có thể kiểm thử với các loại mê cung khác nhau:

```
python pacman.py -l tinyMaze -p SearchAgent –a fn=dfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
python pacman.py -l bigMaze -p SearchAgent -z .5 -a fn=dfs
```

Mê cung trong Pac-Man sẽ hiển thị màu các ô đã duyệt.
Nếu Pac-Man di chuyển quá chậm ta có thể dùng thêm thông số --frameTime 0 ở mỗi lần thực thi chương trình.

3 Tìm kiếm rộng:

Tương tự bài tập trên cài đặt thuật toán tìm kiếm rộng breadthFirstSearch – bfs trong tập tin search.py.

```
python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -z .5 -a fn=bfs
```

Các bạn hãy cài đặt, kiểm thử và thực thi chương trình của mình. Sau đó điền vào bảng sau kết quả tốt nhất chương trình của mình có thể đạt được:

4 Bài tập

		Depth-Fist Search			Breadth-First Search	
Mê cung	Số nút đã duyệt	Đường đi (giải pháp)	Có phải là giải pháp tối ưu không?	Số nút đã duyệt	Đường đi (giải pháp)	Có phải là giải pháp tối ưu không?
Nhỏ (tiny)						
Trung bình (medium)						
Lớn (big)						

Xuất file báo cáo và code gửi lên hệ thống elearning của khoá học theo thời hạn nộp (6 ngày – kể từ ngày ra đề).

Bạn hãy đặt tên theo định dạng sau: PacMan1-MSSV-HoVaTen