Báo cáo diễn giải về khung chương trình tìm kiếm

1)Tìm hiểu về 5 file trong chương chương trình

A> File Util.py

Thực hiện 1 file cấu trúc dữ liệu ngăn xếp với các chức năng thêm và lấy ra các phần tử và dùng hàng hợi ưu tiên cho các thuật toán cần thiết .

- Đầu tiên,tạo 1 lớp FixedRandom .Tạo 1 hàm fixedState xong cho random ngẫu nhiên.Đây là cấu trúc dữ liệu hữu ích để triển khai cho AgentSearch.py
- Tạo 1 lớp ngăn xếp để thực thi lệnh code cho file cấu trúc này với các chức năng thêm vào và lấy ra với cơ chế LIFO.Ngoài ra, còn tạo các hàm kiểm tra cần thiết cho những chức năng với file khác trong chương trình đường đi của Pacman.
- Tạo 1 lớp hàng đợi ưu tiên.Mỗi mục được chèn có độ ưu tiên liên quan đến nó và khách hàng thường quan tâm.
- Tạo 1 lớp đếm cho các đường đi , bước đi của PacMan trong chương trình từ lúc bắt đầu cho tới khi kết thúc
- Tạo 1 lớp đo thời gian để phục vụ cho chương trình khi mà pacman từ vị trí xuất phát khi mình bắt đầu chơi cho tới khi con pacman bị **ma pacman** ăn.

B > File Search.py

Trong file này , sẽ chạy những thuật toán tìm kiếm AI như: DFS ,BFS , UCS , A* ,Hericris.Ngoài ra , mình sẽ gọi lại những hàm khác của file Util.py khi import file này vào search.py

Trong search.py bạn sẽ triển khai các thuật toán tìm kiếm chung được gọi trong SearchAgent.py.

Đoạn chương trình code về các hàm như layout và các hướng đi sao cho tối ưu nhất.Trong chương trình Pacman này sẽ có 4 hướng đi : ***đông ,tây,nam bắc***.

Các thuật toán tìm kiếm trong chương trình thuộc file này thì khi kết thúc 1 hàm tìm kiếm nào đó đều gọi lại hàm raiseNotDefined() của file Util.py

C > File SearchAgent.py

Tập tin này chứa tất cả các tác nhân có thể được chọn để kiểm soát Pacman. Đến chọn một tác nhân, sử dụng tùy chọn '-p' khi chạy pacman.py. Đối số có thể là chuyển cho searchAgent.py của bạn bằng cách sử dụng '-a'

Nhận chức năng tìm kiếm các hàm cần thiết và sử dụng Heuristic khi cần.

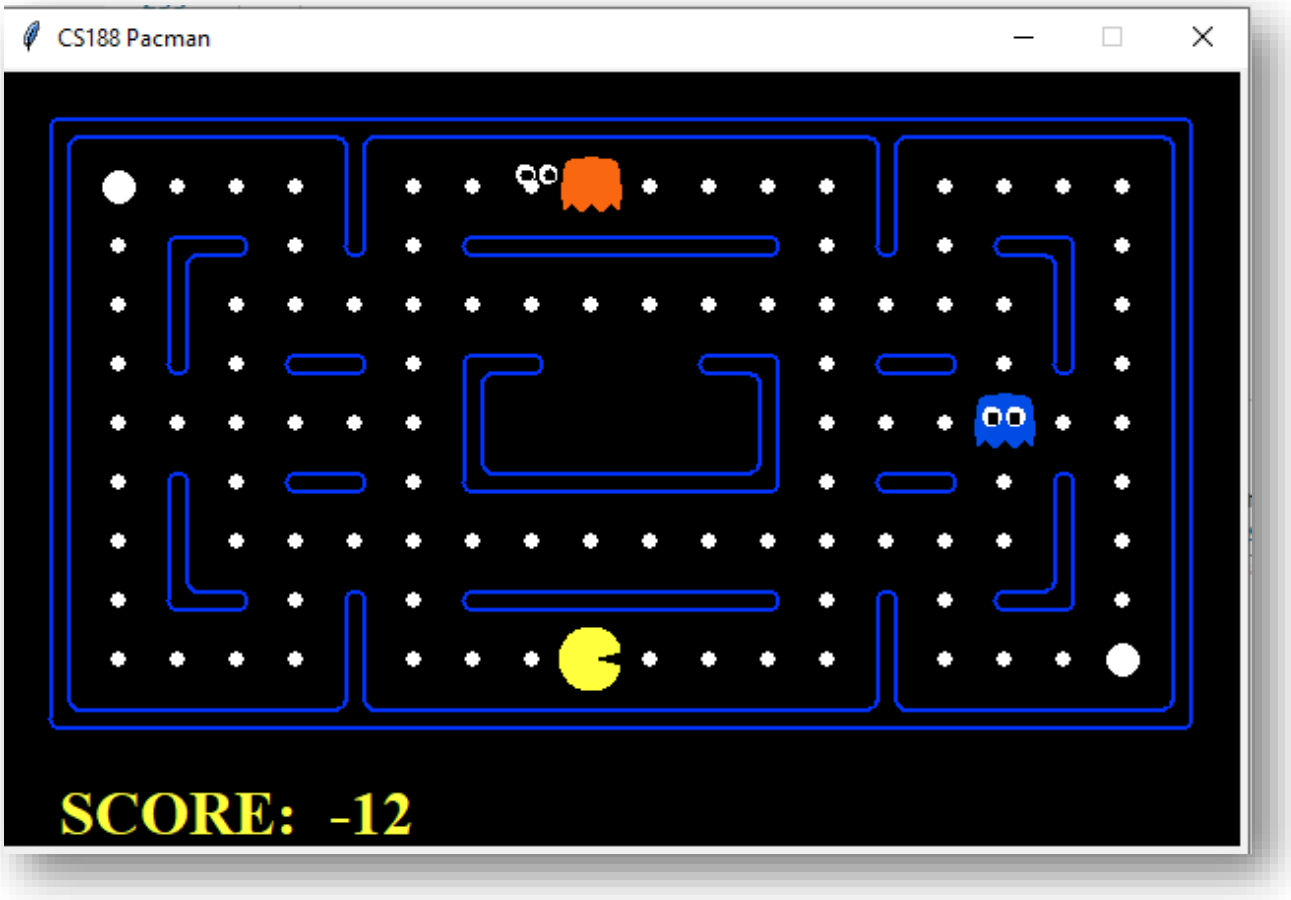
Lấy loại vấn đề để xữ lý từ các tên hàm cụ thể

Thực thi các hành động tìm kiếm đường đi của PacMan từ vị trí góc cho tới khi ăn được các phần thưởng.Bố cục theo trục tung,hoành và lấy gốc toạ độ (0;0) từ phía màn hình xuống.

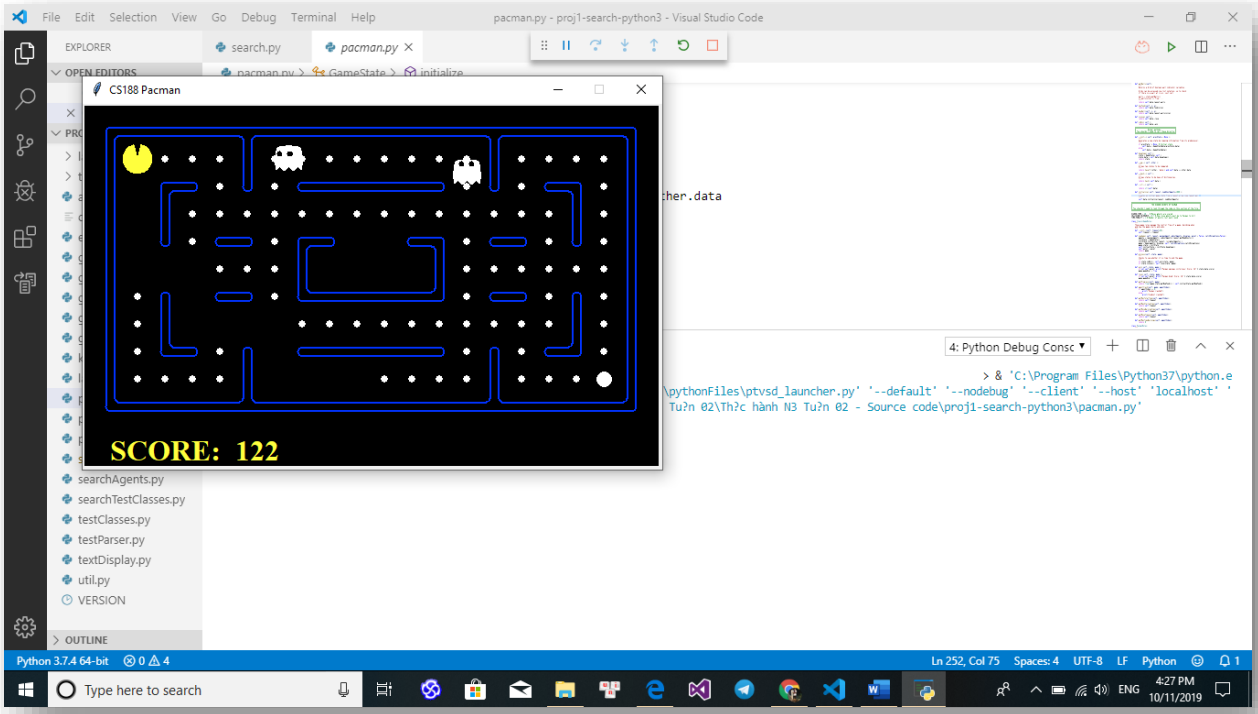
Viết các hàm tìm kiếm đường đi theo các hướng mà thuật toán phải cài đặt.

D > File Pacman.py

Trong file này dùng để chạy chương trình hiển thị ra màn hình Console với giao diện và lối, hướng xữ lý của các thuật toán



Sau khi Pacman ăn phần thưởng thì các con ***ma pacman*** sẽ đông lại thành màu trắng.



Pacman.py sẽ gọi lại file Util.py,layout.py (P là pacman, % là tường , . là bước đi và trong những bước đi có phần thưởng cho Pacman ăn để đạt điểm.Mỗi bước đi, pacman sẽ bị trừ 1 điểm)

Tạo 1 lớp `GameState` để chứa các hàm liên quan.

GameState chỉ định trạng thái trò chơi đầy đủ, bao gồm thức ăn, viên nang, cấu hình tác nhân và thay đổi điểm số.GameStates được sử dụng bởi đối tượng Game để nắm bắt trạng thái thực tế của trò chơi và có thể được sử dụng bởi các đại lý để lý do về trò chơi.

Phần lớn thông tin trong GameState được lưu trữ trong đối tượng GameStateData. Chúng tôi đặc biệt khuyên bạn nên truy cập dữ liệu đó thông qua các phương thức truy cập bên dưới hơn là đề cập trực tiếp đến đối tượng GameStateData. Lưu ý rằng trong Pacman cổ điển, Pacman luôn là tác nhân 0.

Khi nhấn Ctrl+F5 thì lúc kết thúc trò chơi ,hệ thống sẽ thông báo kết quả đã đạt được qua hình ảnh sau:

Hình ảnh kết quả này thì trong đoạn code gần cuối ở file Pacman.py:

```
            gameDisplay = display
            rules.quiet = False
        game = rules.newGame( layout, pacman, ghosts, gameDisplay, beQuiet, catchExceptions)
        game.run()
        if not beQuiet: games.append(game)

        if record:
            import time, pickle
            fname = ('recorded-game-%d' % (i + 1)) +  '-'.join([str(t) for t in time.localtime()[1:6]])
            f = open(fname, 'wb')
            components = {'layout': layout, 'actions': game.moveHistory}
            pickle.dump(components, f)
            f.close()

    if (numGames-numTraining) > 0:
        scores = [game.state.getScore() for game in games]
        wins = [game.state.isWin() for game in games]
        winRate = wins.count(True)/ float(len(wins))
        print('Average Score:', sum(scores) / float(len(scores)))
        print('Scores:       ', ', '.join([str(score) for score in scores]))
        print('Win Rate:      %d/%d (%.2f)' % (wins.count(True), len(wins), winRate))
        print('Record:       ', ', '.join([ ['Loss', 'Win'][int(w)] for w in wins]))

    return games

if __name__ == '__main__':
    """
    The main function called when pacman.py is run
    from the command line:

    > python pacman.py
```

E > File Game.py

Trong file này , mình sẽ gọi lại file cấu trúc dữ liệu cho cho các thuật toán tìm kiếm (bài tập tuần này chỉ làm DFS).

Tạo 1 lớp Agent. The Agent sẽ nhận được GameState (từ {pacman, capt, sonar} .py) và phải trả lại một hành động từ Chỉ đường. (Bắc, Nam, Đông, Tây, Dừng nếu tìm được trạng thái đích)

Tạo 1 lớp hướng đi cho Pacman thực hiện các hành động.

```
class Directions:
    NORTH = 'North'
    SOUTH = 'South'
    EAST = 'East'
    WEST = 'West'
    STOP = 'Stop'

    LEFT =       {NORTH: WEST,
                  SOUTH: EAST,
                  EAST:  NORTH,
                  WEST:  SOUTH,
                  STOP:  STOP}

    RIGHT =      dict([(y,x) for x, y in LEFT.items()])

    REVERSE = {NORTH: SOUTH,
               SOUTH: NORTH,
               EAST: WEST,
               WEST: EAST,
               STOP: STOP}
```

Kế tiếp chúng ta sẽ tạo 1 lớp Configuration:

Cấu hình giữ tọa độ (x, y) của một ký tự, cùng với nó hướng đi.Quy ước cho các vị trí, giống như biểu đồ, là (0,0) là góc dưới bên trái, x tăng theo chiều ngang và y tăng theo chiều dọc. Do đó, hướng bắc là hướng tăng y, hoặc (0,1).

```
# Directions
 _directions = {Directions.NORTH: (0, 1),
                Directions.SOUTH: (0, -1),
                Directions.EAST:  (1, 0),
                Directions.WEST:  (-1, 0),
                Directions.STOP:  (0, 0)}
```

2)Cài đặt thuật toán tìm kiếm DFS

Trong thuật toán tìm kiếm theo chiều sâu này mình sẽ minh hoạ code và mô tả kỹ mà mình đã viết trong chương trình tại file search.py:

```
"*** YOUR CODE HERE ***"
#Trong thuật toán này, sd Stack đã khai báo ở Util.py
# gọi problem.getStartState() để lấy trạng thái bắt đầu
start = problem.getStartState()
c = problem.getStartState()
exploredState = []
exploredState.append(start)
states = util.Stack()
stateTuple = (start, [])
states.push(stateTuple)
#problem.isGoalState(state) để kiểm tra trạng thái state có phải là trạng thái đích không
while not states.isEmpty() and not problem.isGoalState(c):
    state, actions = states.pop()
    exploredState.append(state)
    #problem.getSuccessors(state) để mở trạng thái state
    successor = problem.getSuccessors(state)
    for i in successor:
        coordinates = i[0]
        if not coordinates in exploredState:
            c = i[0]
            direction = i[1]
            states.push((coordinates, actions + [direction]))
return actions + [direction]
util.raiseNotDefined()
```

Lấy trạng thái ban đầu: Trạng thái bắt đầu thì sẽ rỗng. Khi mà bắt đầu trò chơi,thì thuật toán sẽ thêm điểm bắt đầu ngẫu nhiên cho Pacman và sẽ gọi lại hàm Util.Stack() trong file Util.py.Sau đó dùm hàm Tuple để tạo 1 bộ dữ liệu chứ các phần tử các trạng thái bắt đầu mà khi mình chạy nó ra ngẫu nhiên theo từng layout theo các lệnh.

Kiểm tra trạng thái State có phải là trạng thái đích không: Khi mà trạng thái không rỗng và không phải là trạng thái đích thì sẽ thực thi: lấy ra hành động và vị trí và thêm các vị trí khác để tìm đường đi thích hợp ,mau tới thức ăn cho Pacman hơn.Khi mà tới trạng thái đích thì sẽ trả về hướng của pacman qua hành động nó thực thi.Còn khi mà trạng thái cuối chưa có trong kế hoạch sẽ đưa vào trạng thái đóng và mở rộng quy mô ,hướng tìm kiếm thức ăn(dựa vào hàm Successor)

Sau khi đã cài đặt xong, bạn có thể kiểm tra một cách trực quan phần cài đặt của bạn trên game Pacman bằng cách gõ các câu lệnh sau:

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

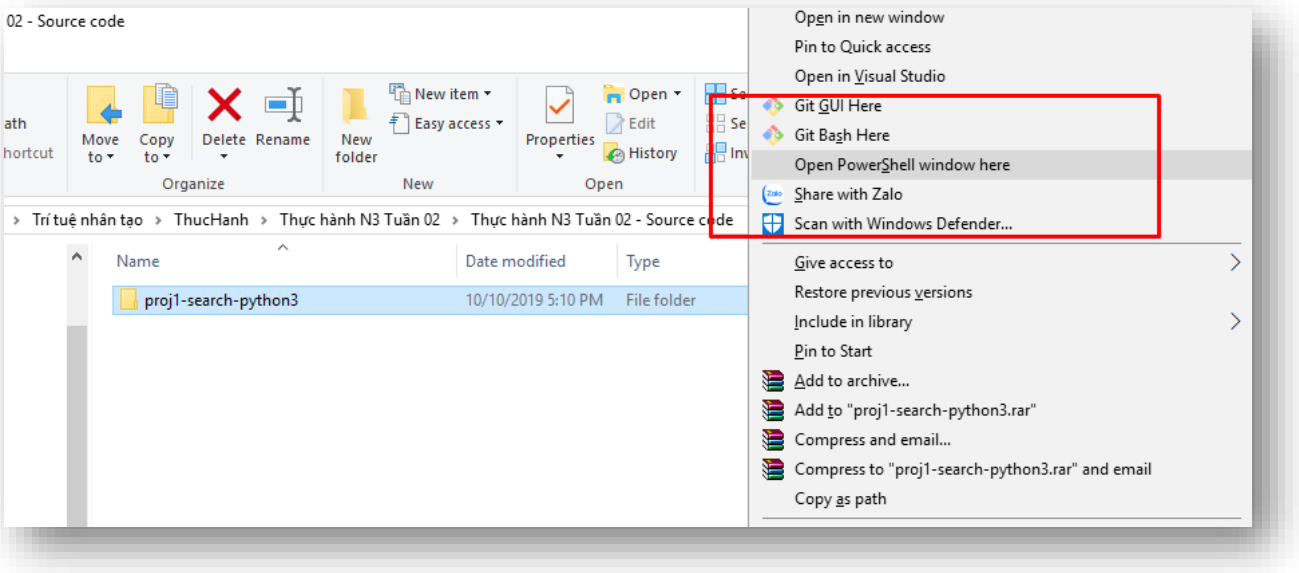
python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs

python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=dfs

Câu lệnh thứ nhất ứng với mê cung tinyMaze: đầu tiên, SearchAgent sẽ sử dụng thuật toán DFS mà bạn vừa cài đặt để lên kế hoạch trong đầu; sau khi đã tìm ra được kế hoạch (chuỗi các hành động để đi từ trạng thái bắt đầu đến trạng thái đích), SearchAgent sẽ thực thi kế hoạch này. Khi chạy, bạn sẽ thấy các ô ở mê cung có màu đỏ với mức độ đậm khác nhau; mức độ đậm này cho biết thứ tự mở của các ô khi chạy thuật toán tìm kiếm: các ô có màu đỏ càng đậm thì được mở càng sớm. Các câu lệnh còn lại cũng tương tự nhưng mà làm với mê cung mediumMaze và bigMaze.

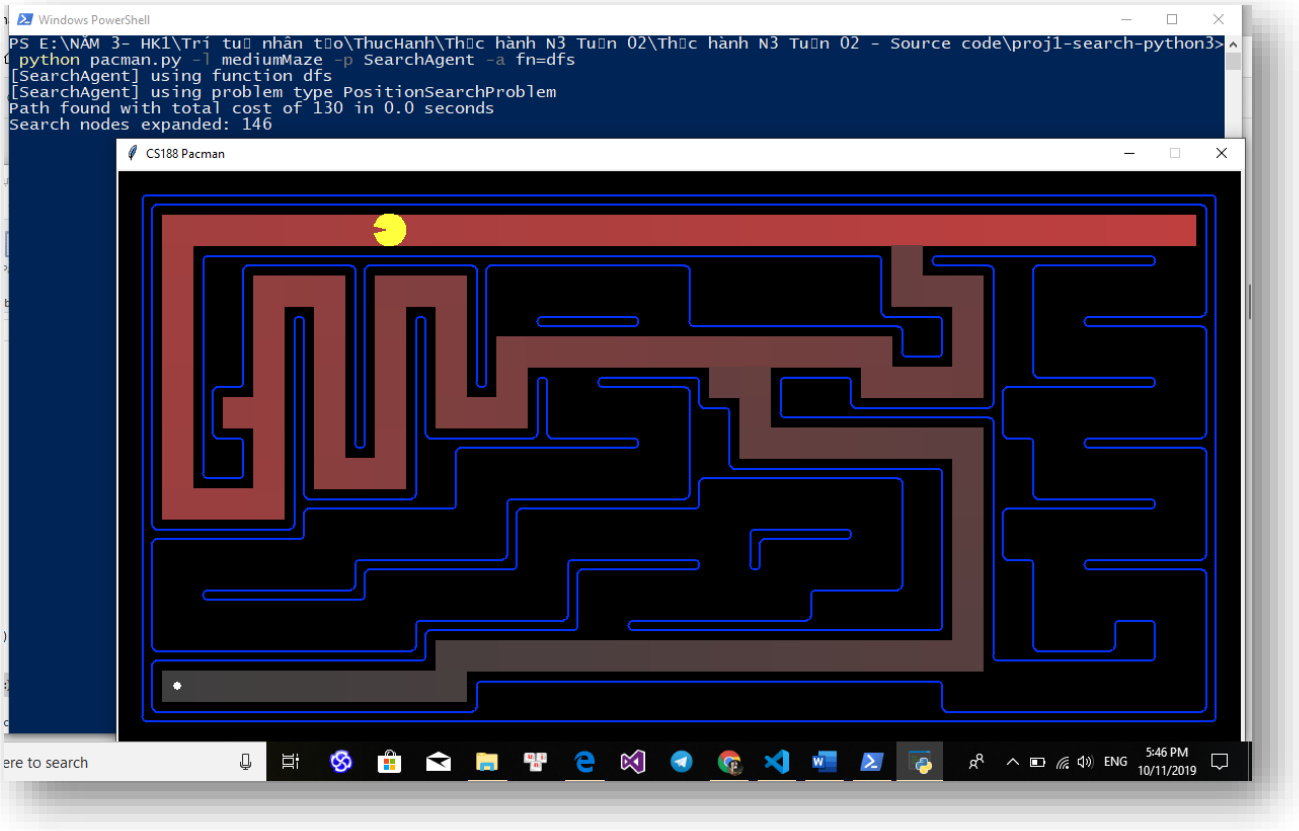
Cuối cùng, bạn gõ ***python autograder.py -q q1*** để kiểm tra phần cài đặt của bạn với các bộ test khác nhau.

Bây giờ, mình sẽ chạy thử chương trình bằng lệnh : Bạn chọn **folder** xong nhấn nút **Shift + chuột phải chọn Open PowerShell window here**

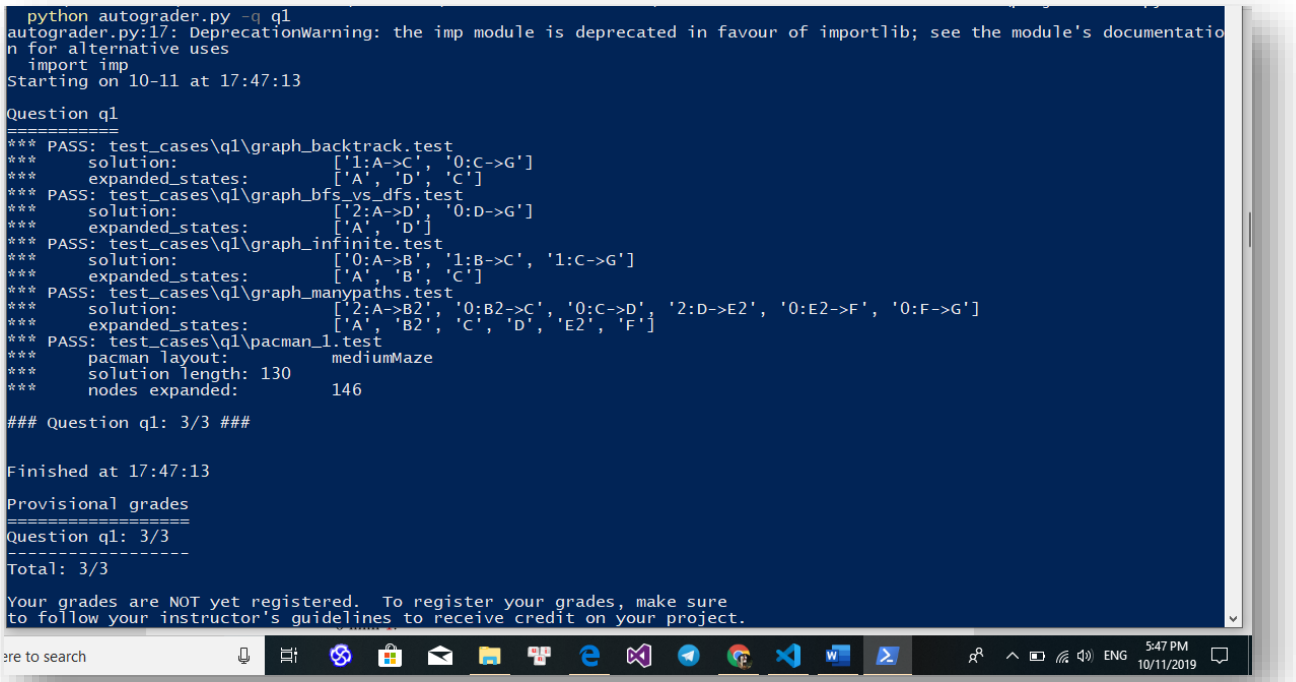


Demo chương trình bằng lệnh:

python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs



Chạy lệnh: ***python autograder.py -q q1***



Lệnh này sẽ cho thấy đường đi ngắn nhất tới đích theo thuật toán DFS và test_case q1