Report Thực Hành Tuần 06

Cài Đặt 2.7 Heuristic Cho Bài Toán Ăn Hết Thức Ăn và 2.8 Tìm Kiếm Nhanh

Họ tên: Nguyễn Sanh Tuấn.

MSSV: 1760227.

Lớp: 17CK2-ca 3 thực hành chiều t7.

1. Heuristic Cho Bài Toán Ăn Hết Thức Ăn

Tìm hiểu về yêu cầu và hướng giải:

- Heuristic phải đảm bào tính nhất quán để đảm bào chính xác. Đầu tiên hãy cố gắng đến một heuristic mà ta thấy; hầu hết tất cả các heurictis được chấp nhận sẽ nhất quán là tốt.
- Nếu sử dụng A* bao giờ chi phí tìm thấy heuristic cũng tệ hơn tìm kiếm UCS(tìm kiếm có chi phí), heuristic khi tìm được là không nhất quán, có lẽ không được chấp nhận ! Mặt khác, heuristic không phù hợp hoặc không nhất quán có thể tìm thấy tối ưu của giải pháp.
- Trạng thái là một tuple (pacmanPocation, foodGrid) trong đó foodGrid là một lưới (xem game.py) là Đúng hoặc Sai. Bạn có thể gọi foodGrid.asList () để nhận một danh sách các tọa độ thực phẩm thay thế.
- Nếu bạn muốn truy cập vào thông tin như tường, viên nang, v.v., bạn có thể truy vấn vấn đề. Ví dụ: Problem.walls cung cấp cho bạn một Lưới là nơi các bức tường.
- Nếu bạn muốn * lưu trữ * thông tin sẽ được sử dụng lại trong các bước đi khác đến heuristic, có một từ điển gọi là problem.heuristicInfo mà bạn có thể sử dụng. Ví dụ: nếu bạn chỉ muốn đếm các bức tường một lần và lưu trữ nó giá trị, thử: problems.heuristicInfo ['wallCount'] = problems.walls.count ()
- Các bước di tiếp theo đến heuristic này có thể truy cập problems.heuristicInfo ['wallCount]

Ý tưởng giải quyết bài toán

- Tìm đường đi ngắn nhất cho Pacman trong mê cung sao cho Pacman ăn hết tất cả các điểm thức ăn. Sử dụng thuật toán UCS, A* đã được cài đặt ở mục 2.4.

- Hàm này tính toán:

a. khoảng cách Manhattan giữa vị trí hiện tại và thực phẩm gần nhất

b. khoảng cách Manhattan giữa thực phẩm gần nhất và thực phẩm xa nhất với nó

2 giá trị này được thêm vào và trả về là heuristic.

Cài đặt thuật toán

```
def foodHeuristic(state, problem):
position, foodGrid = state
    "*** YOUR CODE HERE ***"
   # "nonvisited" chứa tọa độ của các địa điểm thực phẩm chưa được truy cập
    nonvisited = foodGrid.asList()

    # "currentpos" chứa vị trí hiện tại của Pacman
```

```
    currentpos = state[0]

    # "distances" là để lưu trữ khoảng cách Manhattan giữa vị trí hiện tại và
từng vị trí thực phẩm
    distances = []

    heuristic = 0

    # nếu không còn thức ăn, hàm trả về 0
    if len(nonvisited) == 0:
        return heuristic

    # vòng lặp để tính khoảng cách Manhattan giữa vị trí hiện tại và từng vị t
rí thực phẩm
    for elem in nonvisited:
        dist = abs(elem[0] - currentpos[0]) + abs(elem[1] - currentpos[1])
        distances.append((dist,elem))
    # khoảng cách được sắp xếp theo thứ tự tăng dần
    distances = sorted(distances)

    # khoảng cách Manhattan giữa vị trí hiện tại và vị trí thực phẩm gần nhất
được thêm vào heuristic
    heuristic = heuristic + distances[0][0]

    # vị trí hiện tại được thay đổi thành vị trí của thực phẩm gần nhất
    closest_food = distances[0][1]

    # loại bỏ thực phẩm gần nhất khỏi danh sách không mong muốn
    nonvisited = list(nonvisited)
    for elem in nonvisited:
        if elem == closest_food:
            nonvisited.pop(nonvisited.index(elem))
    nonvisited = tuple(nonvisited)

    # nếu không còn thức ăn, hãy quay lại heuristic
    if len(nonvisited) == 0:
        return heuristic

    # vòng lặp để tính khoảng cách Manhattan giữa thực phẩm gần nhất và từng đ
ịa điểm thực phẩm còn lại
    distances = []
    for elem in nonvisited:
        dist = abs(elem[0] - closest_food[0]) + abs(elem[1] - closest_food[1])
        distances.append((dist,elem))
    # khoảng cách được sắp xếp theo thứ tự giảm dần
    distances = sorted(distances, reverse=True)
```

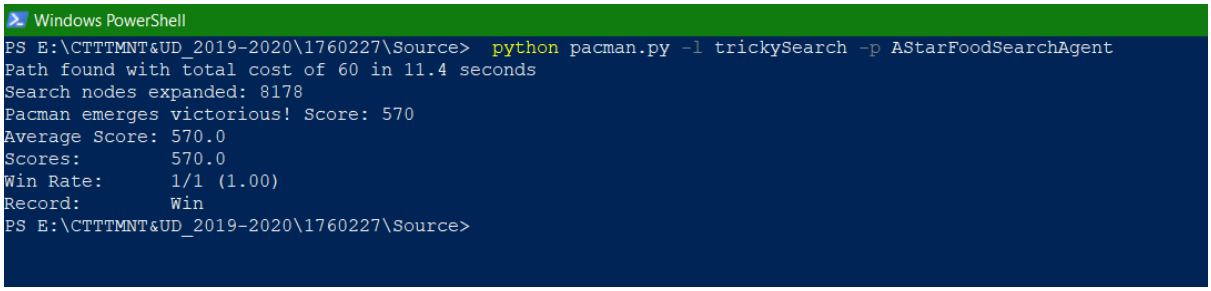
```
    # khoảng cách Manhattan giữa thực phẩm gần nhất và thực phẩm xa nhất với n
ó được thêm vào heuristic
    heuristic = heuristic + distances[0][0]

    return heuristic
```

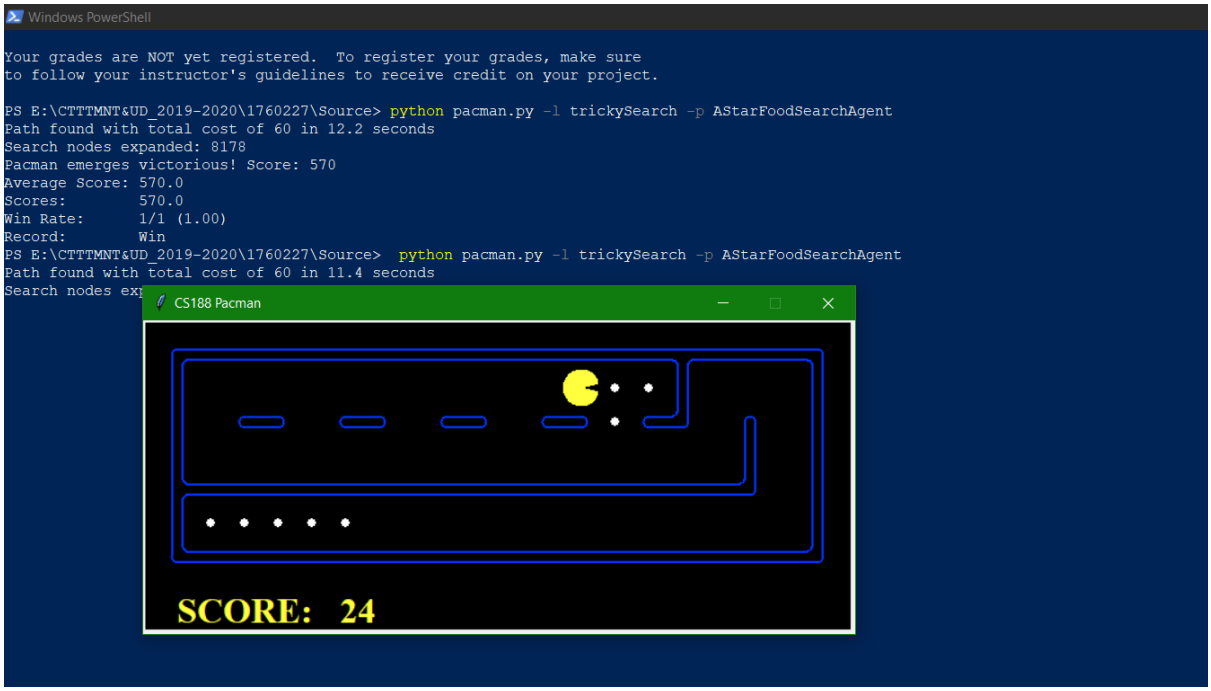
Kiểm tra kết quả cài đặt của thuật toán

- Kiểm tra trực quan cho game pacman bằng câu lệnh:

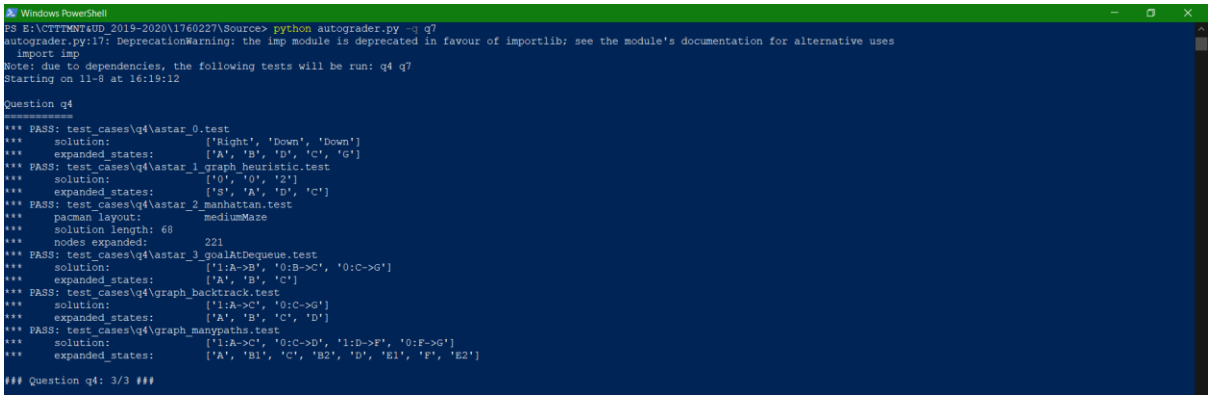
`python pacman.py -l trickySearch -p AStarFoodSearchAgent` với AStarFoodSearchAgent là "shortcut" của:
-p SearchAgent -a "fn=astar,prob=FoodSearchProblem,heuristic=foodHeuristic

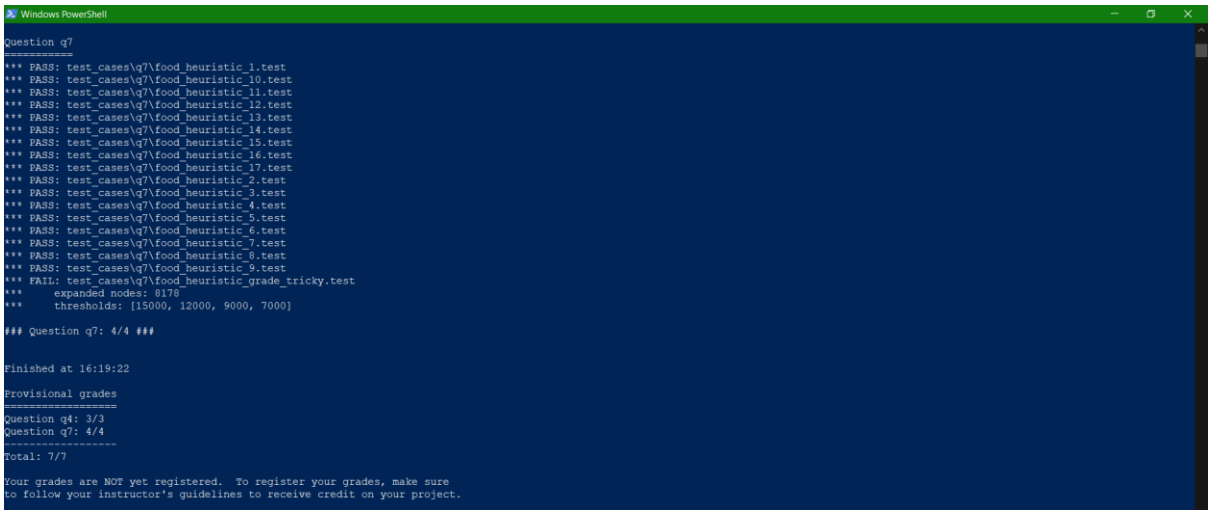
- Màn hình kết quả:





- Kiểm tra cài đặt với các bộ test khác nhau bằng cách gĩ câu lệnh:
  python autograder.py -q q7
  • Nếu số node mở > 15000 thì bạn sẽ được 1/4 điểm.
  • Nếu 12000 < số node mở ≤ 15000 thì bạn sẽ được 2/4 điểm.
  • Nếu 9000 < số node mở ≤ 12000 thì bạn sẽ được 3/4 điểm.
  • Nếu 7000 < số node mở ≤ 9000 thì bạn sẽ được 4/4 điểm.
  • Nếu số node mở ≤ 7000 thì bạn sẽ được 5/4 điểm.
- Màn hình kết quả kiểm tra:





2. Tìm kiếm nhanh.

Yêu cầu:

- Tìm một đường đi hợp lý và tìm nhanh.
- viết một agent để giải quyết nhanh bài toán ăn thức ăn theo chiến lược: ăn điểm thức ăn gần nhất (bạn hình dung như sau: từ ô hiện tại, agent sẽ thực hiện tìm kiếm điểm thức ăn gần nhất, rồi thực đi các hành động để đi đến và ăn điểm thức ăn này; tại ô mới này, agent lại thực hiện tìm kiếm điểm thức ăn gần nhất và thực thi các hành động ...; cứ thế cho đến khi agent ăn hết các điểm thức ăn).

=> Viết hoàn chỉnh hàm findPathToClosestDot trong class ClosestDotSearchAgent

*Giải pháp thực hiện:

Cách nhanh nhất để viết hoàn chỉnh hàm findPathToClosestDot là viết hoàn chỉnh hàm isGoalState trong class AnyFoodSearchProblem, rồi sau đó trong hàm findPathToClosestDot, gọi một hàm tìm kiếm thích hợp với bài toán truyền vào thuộc class AnyFoodSearchProblem.

Cài đặt thuật toán thực hiện theo giải pháp

- Hàm isGoalState(self, gameState):

```
def isGoalState(self, state):
        """
        The state is Pacman's position. Fill this in with a goal test that wil
l
        complete the problem definition.
        """
```

```
        x,y = state

        "*** YOUR CODE HERE ***"
        return state in self.food.asList()
        util.raiseNotDefined()
```

- Hàm chính findPathToClosestDot(self, gameState)

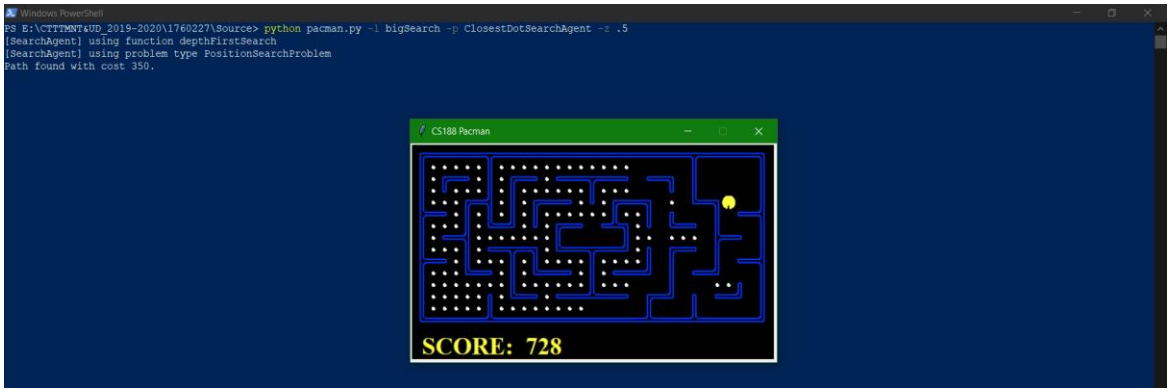
```
def findPathToClosestDot(self, gameState):
        """
        Returns a path (a list of actions) to the closest dot, starting from
        gameState.
        """
        # Here are some useful elements of the startState
        startPosition = gameState.getPacmanPosition()
        food = gameState.getFood()
        walls = gameState.getWalls()
        problem = AnyFoodSearchProblem(gameState)

        "*** YOUR CODE HERE ***"
        return search.uniformCostSearch(problem)
        util.raiseNotDefined()
```

*Kiểm tra kết quả cài đặt thuật toán:

- kiểm tra một cách trực quan trên game Pacman bằng cách gõ câu lệnh:

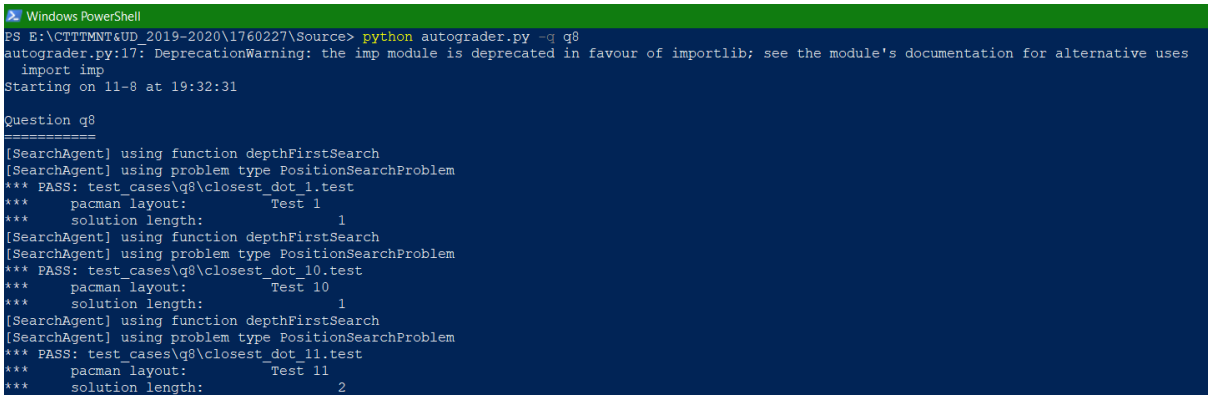
  python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5

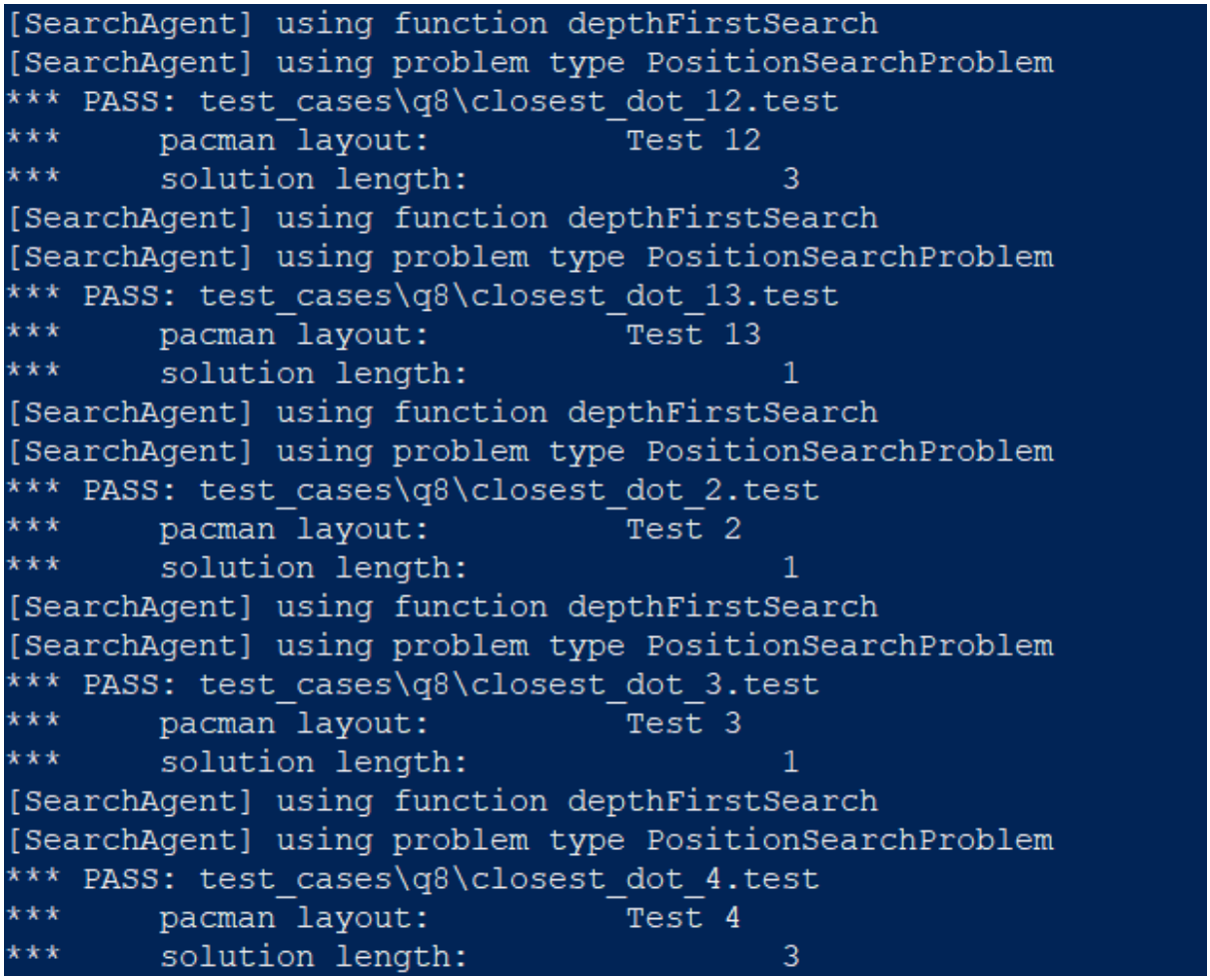
  ⇨ Màn hình kết quả kiểm tra:




```
Windows PowerShell
PS E:\CTTTMNT&UD_2019-2020\1760227\Source> python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with cost 350.
Pacman emerges victorious! Score: 2360
Average Score: 2360.0
Scores:        2360.0
Win Rate:      1/1 (1.00)
Record:        Win
PS E:\CTTTMNT&UD_2019-2020\1760227\Source>
```

- kiểm tra phần cài đặt của bạn với các bộ test khác nhau bằng câu lệnh: python autograder.py -q q8
  ⇨ Màn hình kiểm tra cài đặt thuật toán: pass 3/3





```
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_5.test
***     pacman layout:          Test 5
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_6.test
***     pacman layout:          Test 6
***     solution length:                2
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_7.test
***     pacman layout:          Test 7
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_8.test
***     pacman layout:          Test 8
***     solution length:                1
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
*** PASS: test_cases\q8\closest_dot_9.test
***     pacman layout:          Test 9
***     solution length:                1

### Question q8: 3/3 ###


Finished at 19:32:31

Provisional grades
==================
Question q8: 3/3
------------------
Total: 3/3

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

----------THE END----------