

Report Assignment 1

Search

Guillermo Martín-Coello Juarez & Daniel Varela Sánchez

Section 1 (1 point)

1.1. Personal comment on the approach and decisions of the proposed solution (0.5pt)

This first question consisted of working on a depth first algorithm. To save the path from the parents to the goal node we decided to base our code in the use of a dictionary with all de antecessors of each node.

1.1.1. List & explanation of the framework functions used

To complete the depth first algorithm we used the provided framework functions. This consisted on:

- `getStartState()`: function to obtain the state the game begins in.
- `isGoalState(state)`: function with a state as an argument. Returns boolean, true if the provided state is a goal state, else false.
- `getSuccessors(state)`: function with a state as an argument. Returns a list with all the successors of the provided state.

The class Stack from the utils package was also used with the corresponding methods (push and pop)

1.1.2. Includes code written by students

```

from util import Stack
from game import Directions
st = Stack()
parent = {}

st.push(problem.getStartState())
current = problem.getStartState()
parent[current] = []
visited=[]
while st.isEmpty()==False:

    current = st.pop()
    if problem.isGoalState(current)==True:
        return parent[current]
    visited.append(current)
    successors = problem.getSuccessors(current)
    #successors.reverse()

    #print("current:" + str(current))
    #print("Successors: ",successors)

    for s in successors:
        if s[0] not in visited:
            parent[s[0]] = list(parent[current])
            parent[s[0]].append(s[1])

            st.push(s[0])

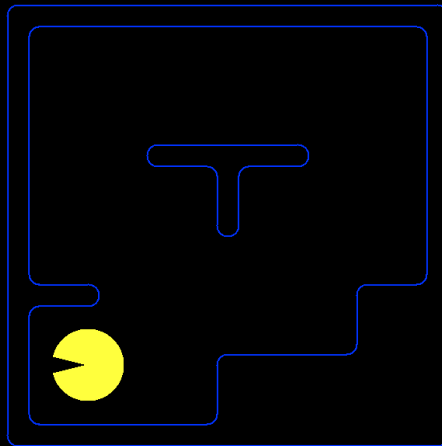
    #print("Return:" + str(parent[current]))
return []

```

Code for exercise 1

1.1.3. Screenshots of executions and test carried out analyzing the results

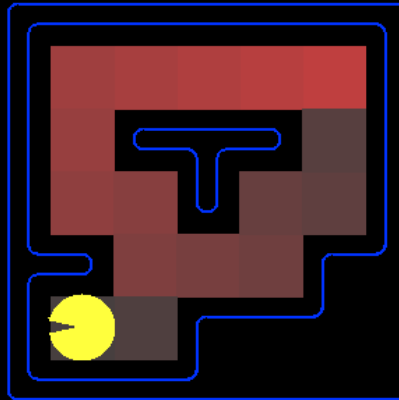
```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
[SearchAgent] using function tinyMazeSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:      502.0
Win Rate:    1/1 (1.00)
Record:      Win
```



SCORE: 502

Successful execution of tinyMazeSearch to prove that the search methods are working as expected

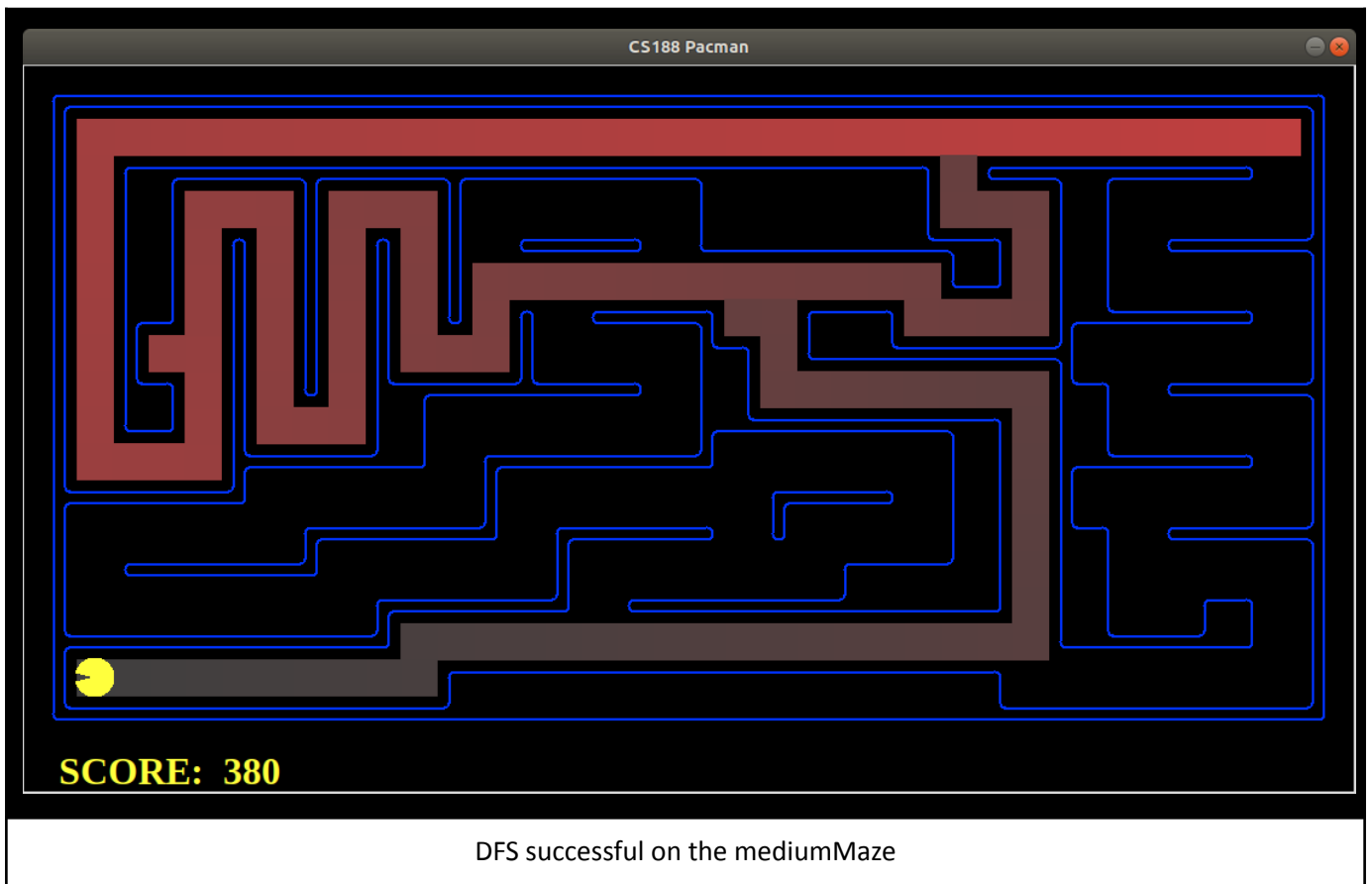
```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l tinyMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:      500.0
Win Rate:    1/1 (1.00)
Record:      Win
```



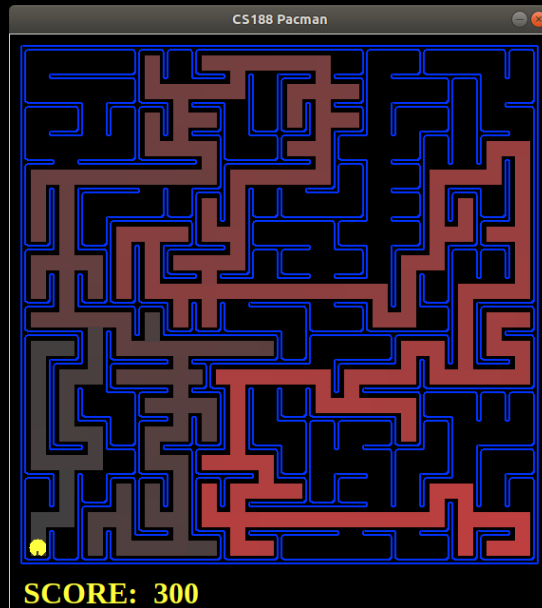
SCORE: 500

DFS successful on the tinyMaze

```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l mediumMaze -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```



```
argo@osboxes:~/Desktop/search$ python3 pacman.py -l bigMaze -z .5 -p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 390
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```



DFS successful on the bigMaze

```
arqo@osboxes:~/Desktop/search$ python3 autograder.py -q q1
Starting on 3-3 at 14:52:40

Question q1
=====
*** PASS: test_cases/q1/graph_backtrack.test
***   solution:      ['1:A->C', '0:C->G']
***   expanded_states: ['A', 'D', 'C']
*** PASS: test_cases/q1/graph_bfs_vs_dfs.test
***   solution:      ['2:A->D', '0:D->G']
***   expanded_states: ['A', 'D']
*** PASS: test_cases/q1/graph_imp.test
***   solution:      []
***   expanded_states: ['A', 'C', 'B']
*** PASS: test_cases/q1/graph_infinite.test
***   solution:      ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q1/graph_manypaths.test
***   solution:      ['2:A->B2', '0:B2->C', '0:C->D', '2:D->E2', '0:E2->F', '0:F->G']
***   expanded_states: ['A', 'B2', 'C', 'D', 'E2', 'F']
*** PASS: test_cases/q1/pacman_1.test
***   pacman layout: mediumMaze
***   solution length: 130
***   nodes expanded: 146
*** PASS: test_cases/q1/pacman_imp.test
***   pacman layout: trapped
***   solution length: 0
***   nodes expanded: 6

### Question q1: 7/7 ###

Finished at 14:52:40

Provisional grades
=====
Question q1: 7/7
-----
Total: 7/7

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

7/7 grade in the q1 tester provided

1.2. Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc (0.5pt)

The result is not optimal (there are better possible paths than the ones achieved).

The result reaches the solution.

The result expands all the nodes needed on a depth first algorithm, meaning it expands first the children.

1.2.1. Answer to question 1.1

The exploration order is not what was expected. It is not a pacman exploration function, it is an algorithm exploration function. Pacman does not actually go to all the explored squares, but these were explored by the algorithm to find the path to the goal.

1.2.2. Answer to question 1.2

This is not a least cost solution. Depth first algorithm will not always find the least cost solution. This is because the algorithm always looks for the first path, not the best.

1.2.3. Answer to question 2

Section 2 (1 point)

2.1. Personal comment on the approach and decisions of the proposed solution (0.5pt)

To accomplish this question we used the skeleton of the previous question and change the mechanism of the algorithm by using a queue instead of a stack.

2.1.1. List & explanation of the framework functions used

To complete the breath first algorithm we used the provided framework functions. This consisted on:

- `getStartState()`: function to obtain the state the game begins in.
- `isGoalState(state)`: function with a state as an argument. Returns boolean, true if the provided state is a goal state, else false.
- `getSuccessors(state)`: function with a state as an argument. Returns a list with all the successors of the provided state.

The class Queue from the utils package was also used with the corresponding methods (push and pop)

2.1.2. Includes code written by students

```

from util import Queue
from game import Directions
st = Queue()
parent = {}

st.push(problem.getStartState())
current = problem.getStartState()
parent[current] = []
visited=[current]
while st.isEmpty()==False:

    current = st.pop()
    if problem.isGoalState(current)==True:
        return parent[current]

    successors = problem.getSuccessors(current)
    #successors.reverse()

    #print("current:" + str(current))
    #print("Successors: ",successors)

    for s in successors:
        if s[0] not in visited:
            visited.append(s[0])
            parent[s[0]] = list(parent[current])
            parent[s[0]].append(s[1])

            st.push(s[0])

```

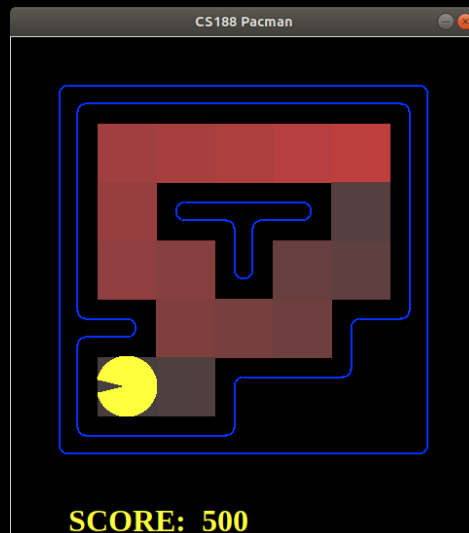
Code for exercise 2

2.1.3. Screenshots of executions and test carried out analyzing the results

```

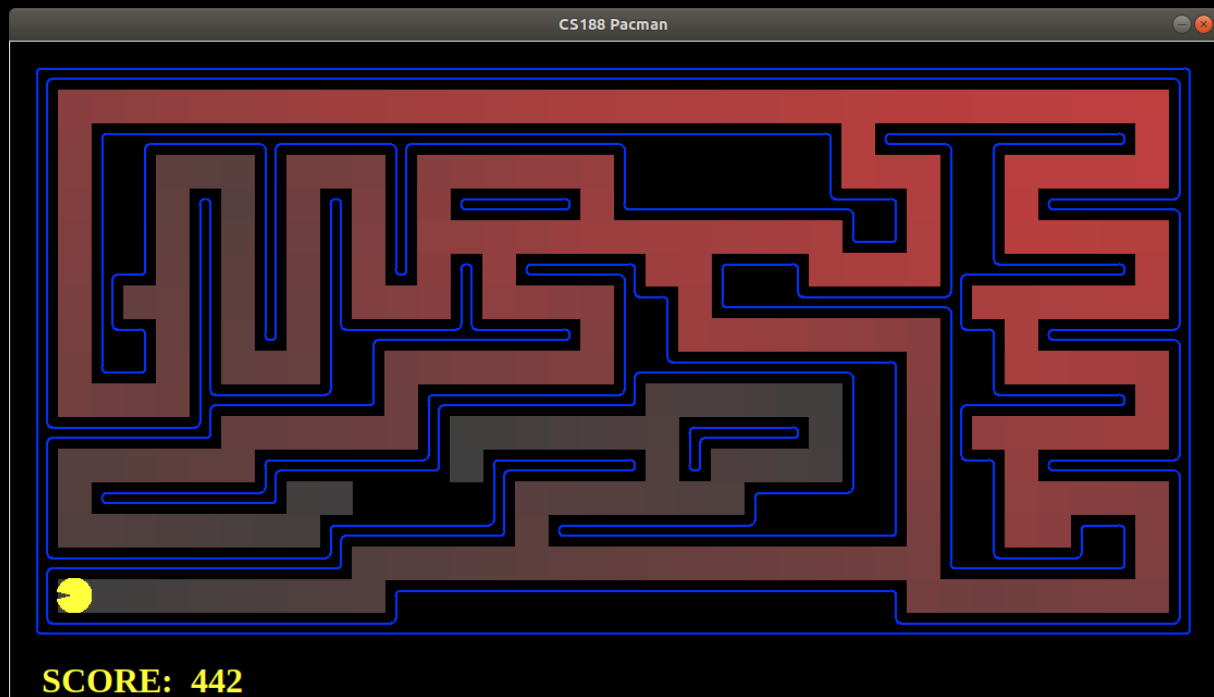
argo@osboxes:~/Desktop/search$ python3 pacman.py -l tinyMaze -p SearchAgent -z 2 --frameTime 0.2
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:      500.0
Win Rate:    1/1 (1.00)
Record:      Win

```

BFS successful in the tinyMaze

```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores: 442.0
Win Rate: 1/1 (1.00)
Record: Win
```

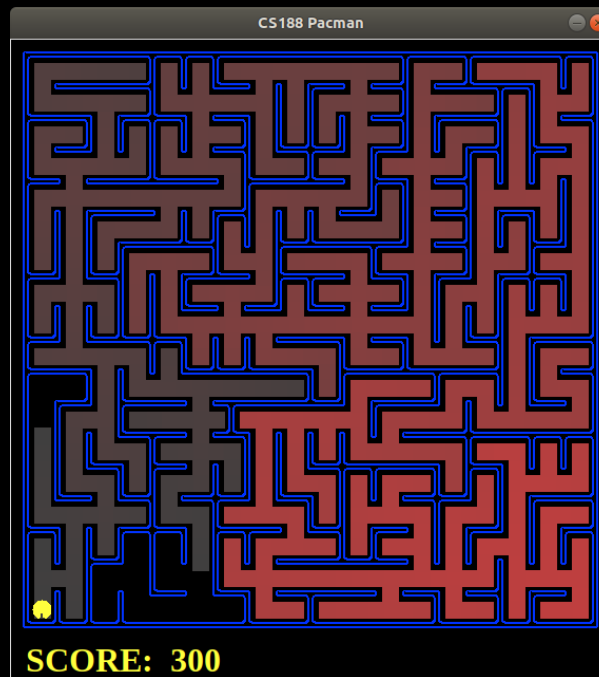


BFS successful in the mediumMaze

```

arqo@osboxes:~/Desktop/search$ python3 pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win

```



BFS successful in the bigMaze

```

arqo@osboxes:~/Desktop/search$ python3 eightpuzzle.py
A random puzzle:
-----
| 4 | 3 | 1 |
-----
|   | 5 | 2 |
-----
| 6 | 7 | 8 |
-----
BFS found a path of 7 moves: ['up', 'right', 'right', 'down', 'left', 'left', 'up']

```

Since our search code is generic it works well on the 8-puzzle problem without changing it

```
arqo@osboxes:~/Desktop/search$ python3 autograder.py -q q2
Starting on 3-3 at 15:03:11
```

Question q2

=====

```
*** PASS: test_cases/q2/graph_backtrack.test
***   solution:      ['1:A->C', '0:C->G']
***   expanded_states: ['A', 'B', 'C', 'D']
*** PASS: test_cases/q2/graph_bfs_vs_dfs.test
***   solution:      ['1:A->G']
***   expanded_states: ['A', 'B']
*** PASS: test_cases/q2/graph_imp.test
***   solution:      []
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_infinite.test
***   solution:      ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_manypaths.test
***   solution:      ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q2/pacman_1.test
***   pacman layout:      mediumMaze
***   solution length: 68
***   nodes expanded:      269
*** PASS: test_cases/q2/pacman_imp.test
***   pacman layout:      trapped
***   solution length: 0
***   nodes expanded:      6
```

Question q2: 7/7

Finished at 15:03:11

Provisional grades

=====

Question q2: 7/7

Total: 7/7

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

7/7 in the q2 test provided

2.2. Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc (0.5pt)

The result is not optimal (there are better possible paths than the ones achieved).

The result reaches the solution.

The result expands all the nodes needed on a breath first algorithm, meaning it expands first the brothers.

2.2.1. Answer to question 3

BFS does find the least cost solution. This does not mean it is the least computational cost.

Section 3 (1 point)

3.1. Personal comment on the approach and decisions of the proposed solution (0.5pt)

To approach this question we worked around using a priority queue to implement the uniform cost algorithm in the easiest and strongest possible way.

3.1.1. List & explanation of the framework functions used

To complete the uniform cost algorithm we used the provided framework functions. This consisted on:

- `getStartState()`: function to obtain the state the game begins in.
- `isGoalState(state)`: function with a state as an argument. Returns boolean, true if the provided state is a goal state, else false.
- `getSuccessors(state)`: function with a state as an argument. Returns a list with all the successors of the provided state.

The class `PriorityQueue` from the `utils` package was also used with the corresponding methods (`push` and `pop`)

3.1.2. Includes code written by students

```
from util import PriorityQueue
from game import Directions
st = PriorityQueue()
parent = {}
costs = {}

st.push(problem.getStartState(), 0)
current = problem.getStartState()
parent[current] = []
costs[current] = 0
visited=[]
while st.isEmpty()==False:

    current = st.pop()
    visited.append(current)
    if problem.isGoalState(current)==True:
        return parent[current]

    successors = problem.getSuccessors(current)
    #successors.reverse()

    #print("current:" + str(current))
    #print("Successors: ",successors)

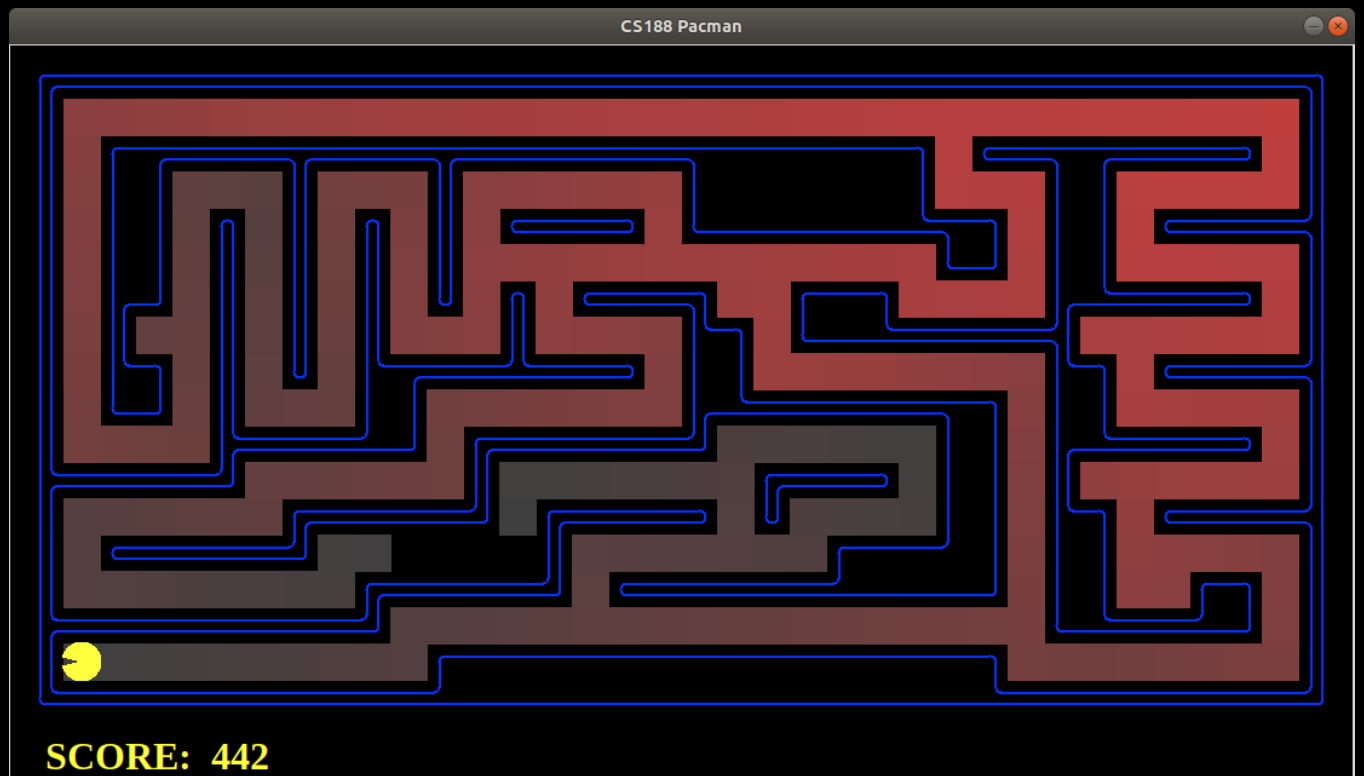
    for s in successors:
        if s[0] not in visited:
            if (s[0] in costs and costs[s[0]]>s[2]+costs[current]) or s[0] not in costs:
                parent[s[0]] = list(parent[current])
                parent[s[0]].append(s[1])
                costs[s[0]] = costs[current] + s[2]
                st.push(s[0], costs[s[0]])

    #print("Return:" + str(parent[current]))
    return []
```

Code for exercise 3

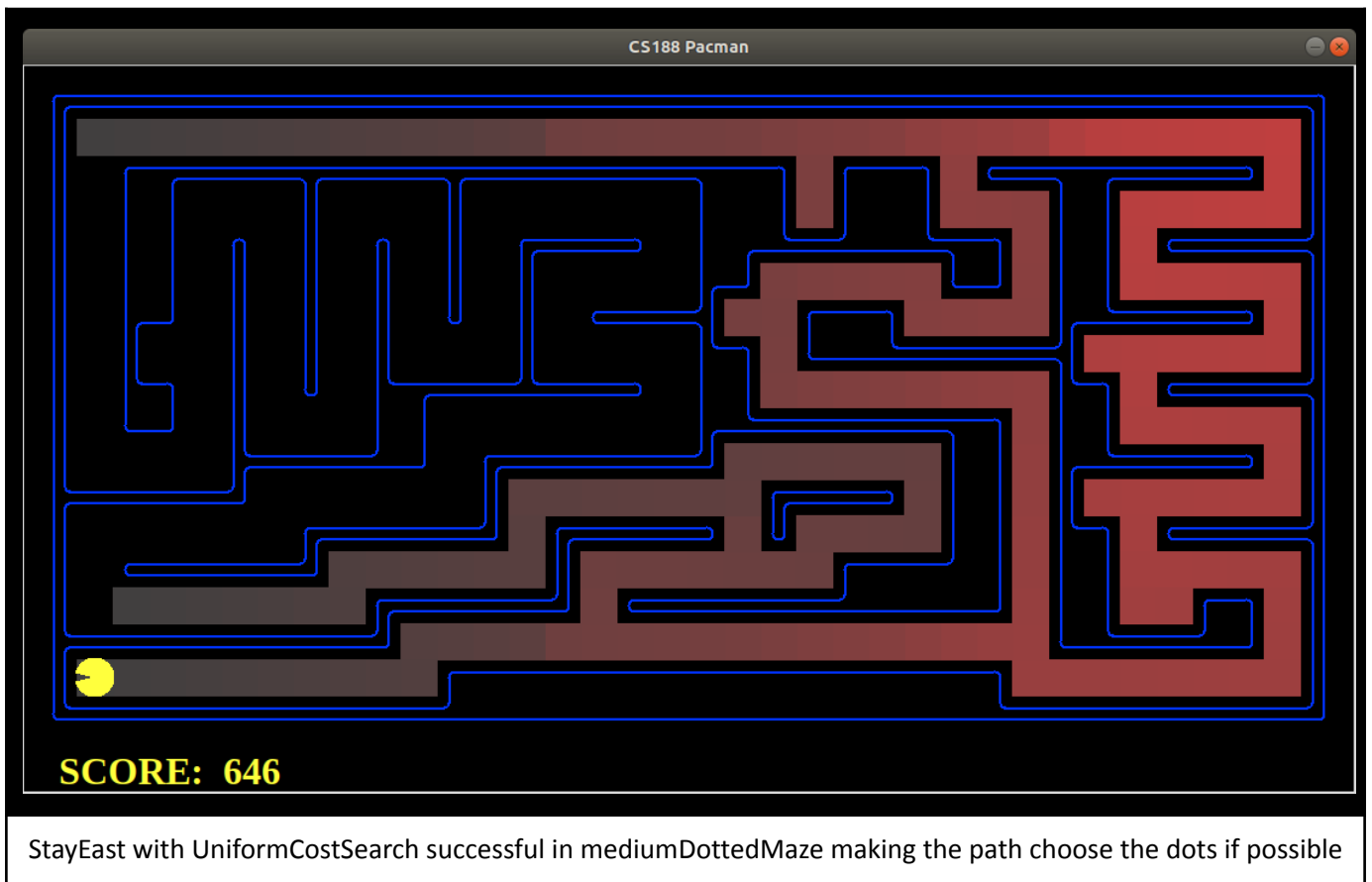
3.1.3. Screenshots of executions and test carried out analyzing the results

```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:      442.0
Win Rate:    1/1 (1.00)
Record:      Win
```

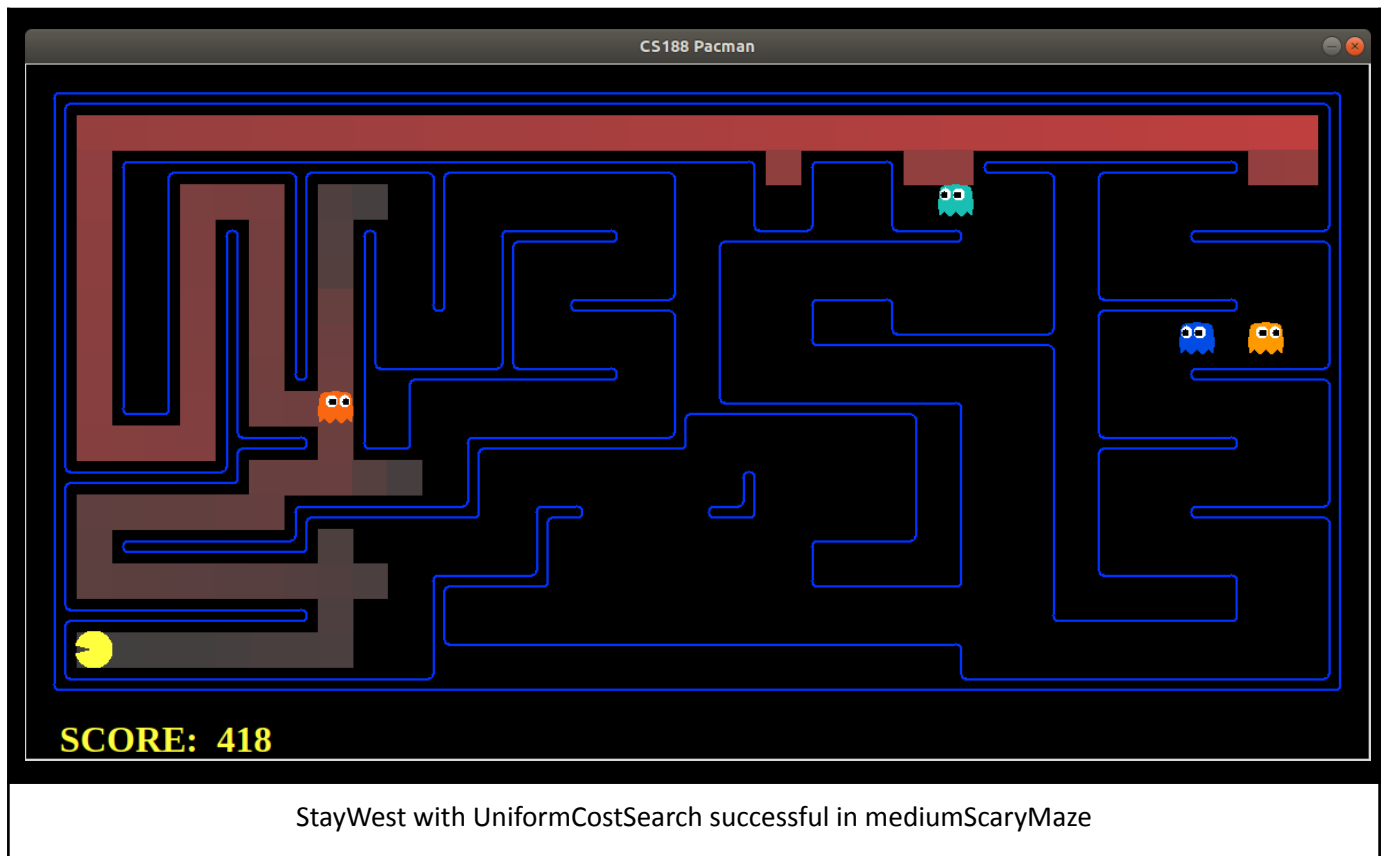


UniformCostSearch successful in mediumMaze

```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:      646.0
Win Rate:    1/1 (1.00)
Record:      Win
```



```
arqo@osboxes:~/Desktop/search$ python3 pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores: 418.0
Win Rate: 1/1 (1.00)
Record: Win
```



```
arqo@osboxes:~/Desktop/search$ python3 autograder.py -q q3
Starting on 3-3 at 15:09:32
```

Question q3

```
=====
*** PASS: test_cases/q3/graph_backtrack.test
***   solution:      ['1:A->C', '0:C->G']
***   expanded_states: ['A', 'B', 'C', 'D']
*** PASS: test_cases/q3/graph_bfs_vs_dfs.test
***   solution:      ['1:A->G']
***   expanded_states: ['A', 'B']
*** PASS: test_cases/q3/graph_imp.test
***   solution:      []
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q3/graph_infinite.test
***   solution:      ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q3/graph_manypaths.test
***   solution:      ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q3/pacman_imp.test
***   pacman layout:  trapped
***   solution length: 0
***   nodes expanded: 6
*** PASS: test_cases/q3/ucs_0_graph.test
***   solution:      ['Right', 'Down', 'Down']
***   expanded_states: ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q3/ucs_1_problemC.test
***   pacman layout:  mediumMaze
***   solution length: 68
***   nodes expanded: 269
*** PASS: test_cases/q3/ucs_2_problemE.test
***   pacman layout:  mediumMaze
***   solution length: 74
***   nodes expanded: 260
*** PASS: test_cases/q3/ucs_3_problemW.test
***   pacman layout:  mediumMaze
***   solution length: 152
***   nodes expanded: 173
*** PASS: test_cases/q3/ucs_4_testSearch.test
***   pacman layout:  testSearch
***   solution length: 7
***   nodes expanded: 14
*** PASS: test_cases/q3/ucs_5_goalAtDequeue.test
***   solution:      ['1:A->B', '0:B->C', '0:C->G']
***   expanded_states: ['A', 'B', 'C']
```

Question q3: 7/7

Finished at 15:09:32

Provisional grades

=====

Question q3: 7/7

Total: 7/7

7/7 in the q3 test provided

3.2. Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc (0.5pt)

The result is optimal in solution cost. It is not necessarily optimal in computational cost.

The result reaches the solution.

The result expands the nodes with the least cost function. If a node has already been visited, it will not be visited again.

Section 4 (2 points)

4.1. Personal comment on the approach and decisions of the proposed solution (1pt)

To implement the A* algorithm we used the uniform cost algorithm as a base and modified it to implement the heuristic use with values and comparisons.

4.1.1. List & explanation of the framework functions used

To complete the A* algorithm we used the provided framework functions. This consisted on:

getStartState(): function to obtain the state the game begins in.

isGoalState(state): function with a state as an argument. Returns boolean, true if the provided state is a goal state, else false.

getSuccessors(state): function with a state as an argument. Returns a list with all the successors of the provided state.

The class PriorityQueue from the utils package was also used with the corresponding methods (push and pop)

4.1.2. Includes code written by students

```
from util import PriorityQueue
from game import Directions
st = PriorityQueue()
parent = {}
costs = {}

st.push(problem.getStartState(), 0)
current = problem.getStartState()
parent[current] = []
costs[current] = 0
visited=[]
while st.isEmpty()==False:

    current = st.pop()
    visited.append(current)
    if problem.isGoalState(current)==True:
        return parent[current]

    successors = problem.getSuccessors(current)
    #successors.reverse()

    #print("current:" + str(current))
    #print("Successors: ",successors)

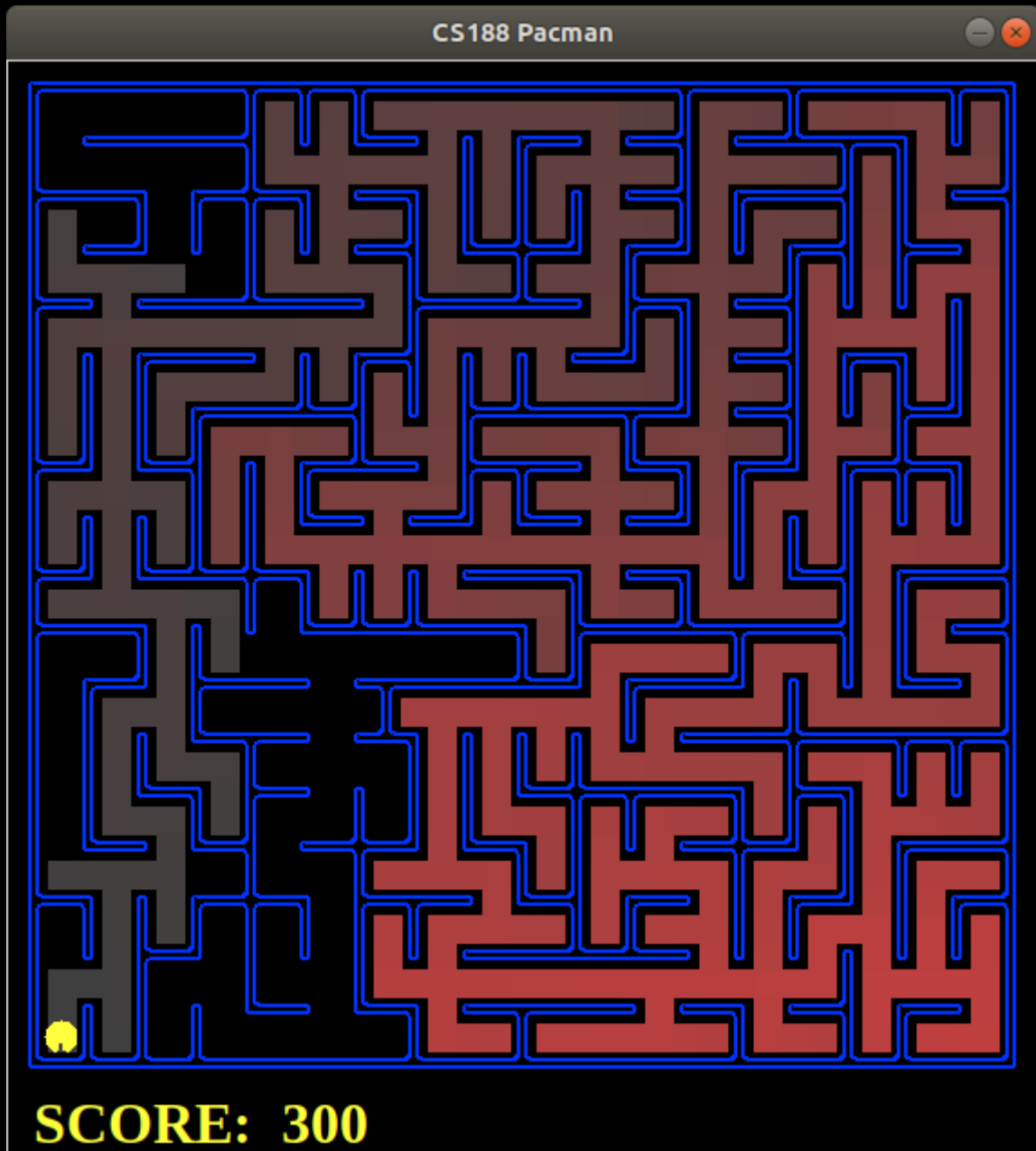
    for s in successors:
        if s[0] not in visited:
            if (s[0] in costs and costs[s[0]]>s[2]+costs[current]+heuristic(s[0], problem)) or s[0] not in costs:
                parent[s[0]] = list(parent[current])
                parent[s[0]].append(s[1])
                costs[s[0]] = costs[current] + s[2]
                st.push(s[0], costs[s[0]]+ heuristic(s[0], problem))

    #print("Return:" + str(parent[current]))
    return []
```

Code for exercise 4

4.1.3. Screenshots of executions and test carried out analyzing the results

```
arg0@osboxes:~/Desktop/search$ python3 pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 549
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores: 300.0
Win Rate: 1/1 (1.00)
Record: Win
```



bigMaze solved with A*

```
arqo@osboxes:~/Desktop/search$ python3 autograder.py -q q4
Starting on 3-3 at 15:13:35
```

Question q4

=====

```
*** PASS: test_cases/q4/astar_0.test
***   solution:      ['Right', 'Down', 'Down']
***   expanded_states: ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q4/astar_1_graph_heuristic.test
***   solution:      ['0', '0', '2']
***   expanded_states: ['S', 'A', 'D', 'C']
*** PASS: test_cases/q4/astar_2_manhattan.test
***   pacman layout: mediumMaze
***   solution length: 68
***   nodes expanded: 221
*** PASS: test_cases/q4/astar_3_goalAtDequeue.test
***   solution:      ['1:A->B', '0:B->C', '0:C->G']
***   expanded_states: ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
***   solution:      ['1:A->C', '0:C->G']
***   expanded_states: ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
***   solution:      ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states: ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
```

Question q4: 3/3

Finished at 15:13:35

Provisional grades

=====

Question q4: 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.

7/7 in the q4 test provided

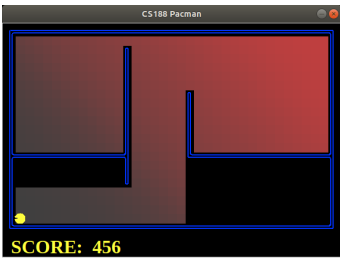
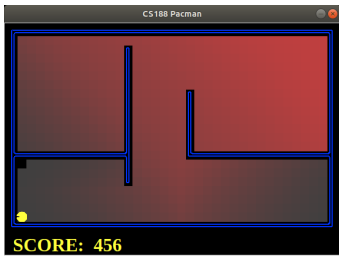
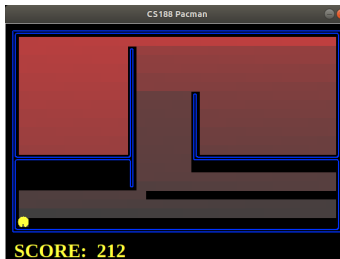
4.2. Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc (1pt)

The behaviour of pacman is not optimal. This is because elimination of repeated nodes can cause a better path to be evaded.

Pacman reaches the solution.

This algorithm expands the nodes with least cost and heuristic. This balance between the cost to the node and the path to the goal.

4.2.1. Answer to question 4

		
A*	BFS	DFS

When we try to execute the openMaze with the three different algorithms developed, we obtain very different results: For the A* algorithm with Manhattan heuristic gets a clean path shown by the colours (which represent the decision tree) where it is visible that it is an informed search because it follows a logical path to our eyes (for us who know the objective). On the other hand, for the BFS algorithm, almost the whole map is searched but the end path is the optimal one. Lastly, for the DFS, we see that neither the decision tree nor the path taken make sense to us.

The previous explanation can be interpreted also by the extended nodes count and the total cost of the solution shown as output which goes as following:

A* : 535 nodes extended and a cost of 54

BFS : 682 nodes extended and a cost of 54

DFS : 806 nodes extended and a cost of 298

```
argo@osboxes:~/Desktop/search$ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 535
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
argo@osboxes:~/Desktop/search$ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=bfs
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.0 seconds
Search nodes expanded: 682
Pacman emerges victorious! Score: 456
Average Score: 456.0
Scores: 456.0
Win Rate: 1/1 (1.00)
Record: Win
argo@osboxes:~/Desktop/search$ python3 pacman.py -l openMaze -z .5 -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.0 seconds
Search nodes expanded: 806
Pacman emerges victorious! Score: 212
Average Score: 212.0
Scores: 212.0
Win Rate: 1/1 (1.00)
Record: Win
argo@osboxes:~/Desktop/search$
```

Section 5 (2 points)

5.1. Personal comment on the approach and decisions of the proposed solution (1pt)

Question 5 made us formulate a new problem for the algorithms to solve. To approach this goal we created a list with the already accessed corners. This made it easy to add the corners when reached so we could know when the goal was achieved.

5.1.1. List & explanation of the framework functions used

To complete this question we had to use various functions of the framework, including:

`directionToVector(action)`: function to find the position reached after the corresponding action was performed.

5.1.2. Includes code written by students

```
def __init__(self, startingGameState):
    """
    Stores the walls, pacman's starting position and corners.
    """
    self.walls = startingGameState.getWalls()
    self.startingPosition = startingGameState.getPacmanPosition()
    top, right = self.walls.height-2, self.walls.width-2
    self.corners = ((1,1), (1,top), (right, 1), (right, top))
    for corner in self.corners:
        if not startingGameState.hasFood(*corner):
            print('Warning: no food in corner ' + str(corner))
    self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
    # Please add any code here which you would like to use
    # in initializing the problem
    """ YOUR CODE HERE """
    #self.corners_left = list(self.corners)
    # self.corners_left = ()

def getStartState(self):
    """
    Returns the start state (in your state space, not the full Pacman state
    space)
    """
    """ YOUR CODE HERE """
    return (self.startingPosition, ())

def isGoalState(self, state):
    """
    Returns whether this search state is a goal state of the problem.
    """
    """ YOUR CODE HERE """
    #if state in self.corners_left:
    #    self.corners_left.remove(state)
    #    print("YEEEEEEEEEEEEEEEEEEEEEEEEEEHAW")
    ##if len(self.corners_left)==len(self.corners):
    #if not self.corners_left:
    #    return True
    #else:
    #    return False
    #if len(self.corners_left)==len(self.corners):
    #    return True
    #else:
    #    return False
    if len(state[1])==len(self.corners):
        return True
    else:
        return False
```

```

def getSuccessors(self, state):
    """
    Returns successor states, the actions they require, and a cost of 1.

    As noted in search.py:
    For a given state, this should return a list of triples, (successor,
    action, stepCost), where 'successor' is a successor to the current
    state, 'action' is the action required to get there, and 'stepCost'
    is the incremental cost of expanding to that successor
    """

    successors = []
    for action in [Directions.NORTH, Directions.SOUTH, Directions.EAST, Directions.WEST]:
        # Add a successor state to the successor list if the action is legal
        # Here's a code snippet for figuring out whether a new position hits a wall:
        #   x,y = currentPosition
        #   dx, dy = Actions.directionToVector(action)
        #   nextx, nexty = int(x + dx), int(y + dy)
        #   hitsWall = self.walls[nextx][nexty]

        """ YOUR CODE HERE """

        x,y = state[0]

        dx, dy = Actions.directionToVector(action)
        nextx, nexty = int(x + dx), int(y + dy)
        hitsWall = self.walls[nextx][nexty]
        if hitsWall==False:
            next=(nextx, nexty)
            left = state[1]
            if next in self.corners and next not in state[1]:
                left=left+(next,)

            successors.append(((next, left), action, 1))
            #print(state)
            #print(self.corners_left)

    self._expanded += 1 # DO NOT CHANGE
    return successors

```

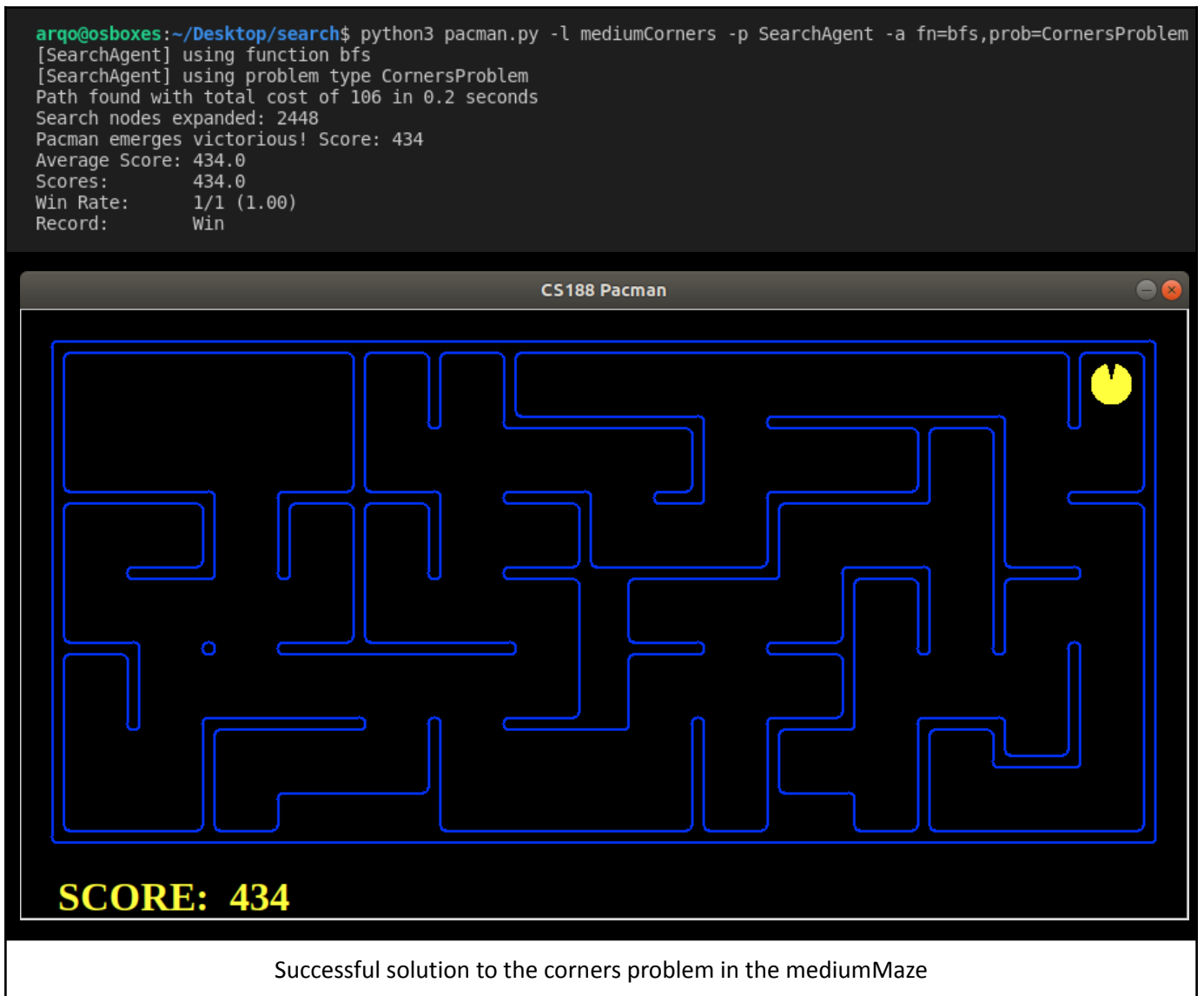
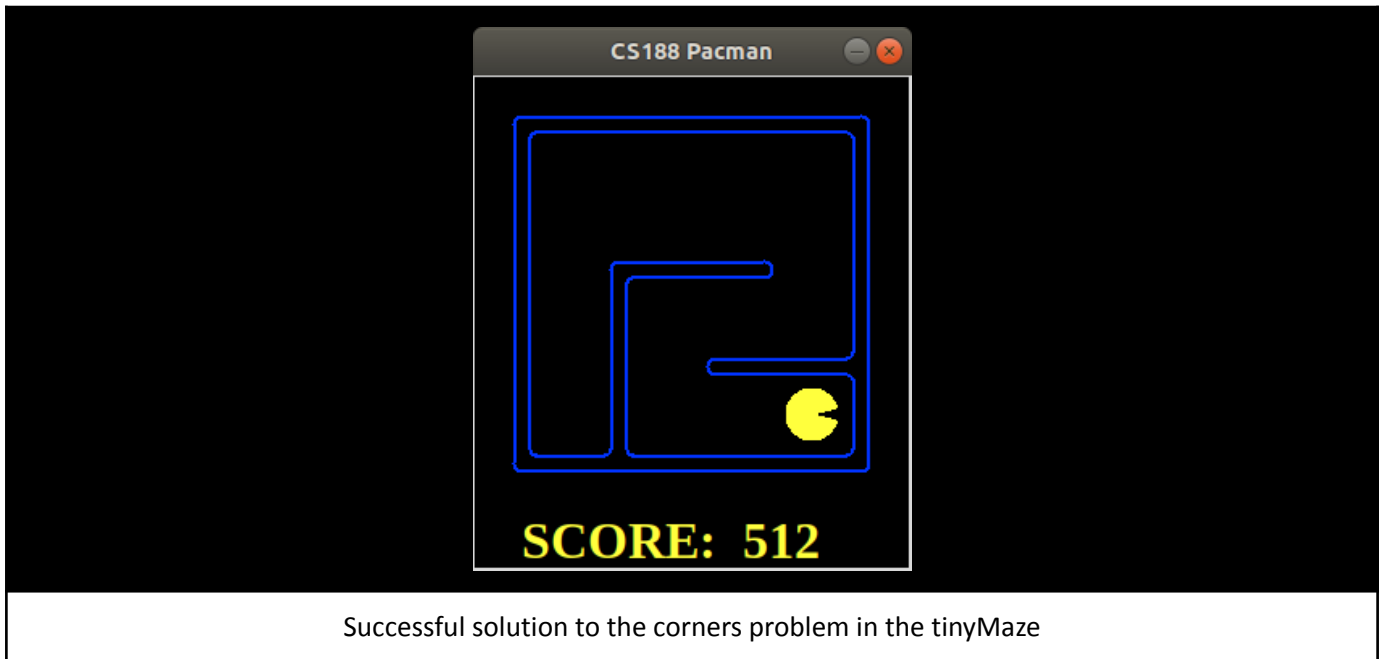
Code for exercise 5

5.1.3. Screenshots of executions and test carried out analyzing the results

```

arqo@osboxes:~/Desktop/search$ python3 pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 28 in 0.0 seconds
Search nodes expanded: 435
Pacman emerges victorious! Score: 512
Average Score: 512.0
Scores:      512.0
Win Rate:    1/1 (1.00)
Record:      Win

```



```

arqo@osboxes:~/Desktop/search$ python3 autograder.py -q q5
Note: due to dependencies, the following tests will be run: q2 q5
Starting on 3-3 at 15:16:32

Question q2
=====
*** PASS: test_cases/q2/graph_backtrack.test
***   solution:          ['1:A->C', '0:C->G']
***   expanded_states:   ['A', 'B', 'C', 'D']
*** PASS: test_cases/q2/graph_bfs_vs_dfs.test
***   solution:          ['1:A->G']
***   expanded_states:   ['A', 'B']
*** PASS: test_cases/q2/graph_imp.test
***   solution:          []
***   expanded_states:   ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_infinite.test
***   solution:          ['0:A->B', '1:B->C', '1:C->G']
***   expanded_states:   ['A', 'B', 'C']
*** PASS: test_cases/q2/graph_manypaths.test
***   solution:          ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states:   ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases/q2/pacman_1.test
***   pacman layout:     mediumMaze
***   solution length:   68
***   nodes expanded:    269
*** PASS: test_cases/q2/pacman_imp.test
***   pacman layout:     trapped
***   solution length:   0
***   nodes expanded:    6

### Question q2: 7/7 ###

Question q5
=====
*** PASS: test_cases/q5/corner_tiny_corner.test
***   pacman layout:     tinyCorner
***   solution length:    28

### Question q5: 3/3 ###

Finished at 15:16:32

Provisional grades
=====
Question q2: 7/7
Question q5: 3/3
-----
Total: 10/10

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

```

3/3 in the q5 test provided

5.2. Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc (1pt)

The behavior of pacman is the expected one. It is optimal and it reaches the solution. The nodes it expands depend on the algorithm used.

Section 6 (3 points)

6.1. Personal comment on the approach and decisions of the proposed solution (1.5pt)

To complete this question we made a heuristic with less cost the closest it was to the corners.

6.1.1. List & explanation of the framework functions used

No interesting framework functions were used on this exercise.

6.1.2. Includes code written by students

```
x , y = state[0]
max = 0
for corner in problem.corners:
    if corner not in state[1]:
        x2, y2 = corner
        distance = abs(x2 - x) + abs(y2 - y)
        #distance = manhattanHeuristic(state[0],corner)
        if max == 0 or max < distance:
            max = distance
            c.. = corner

return max
```

Code for exercise 6

6.1.3. Screenshots of executions and test carried out analyzing the results

```

arqo@osboxes:~/Desktop/search$ python3 autograder.py -q q6
Note: due to dependencies, the following tests will be run: q4 q6
Starting on 3-3 at 17:22:38

Question q4
=====
*** PASS: test_cases/q4/astar_0.test
***   solution:          ['Right', 'Down', 'Down']
***   expanded_states:    ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases/q4/astar_1_graph_heuristic.test
***   solution:          ['0', '0', '2']
***   expanded_states:    ['S', 'A', 'D', 'C']
*** PASS: test_cases/q4/astar_2_manhattan.test
***   pacman layout:      mediumMaze
***   solution length:    68
***   nodes expanded:     221
*** PASS: test_cases/q4/astar_3_goalAtDequeue.test
***   solution:          ['1:A->B', '0:B->C', '0:C->G']
***   expanded_states:    ['A', 'B', 'C']
*** PASS: test_cases/q4/graph_backtrack.test
***   solution:          ['1:A->C', '0:C->G']
***   expanded_states:    ['A', 'B', 'C', 'D']
*** PASS: test_cases/q4/graph_manypaths.test
***   solution:          ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***   expanded_states:    ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q4: 3/3 ###

```

Successful feedback of test q4 provided

```

Question q6
=====
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
*** PASS: heuristic value less than true cost at start state
path: ['North', 'East', 'East', 'East', 'East', 'North', 'North', 'West', 'West', 'West',
th', 'North', 'North', 'East', 'East', 'North', 'North', 'North', 'North', 'North', 'Nor
outh', 'South', 'South', 'South', 'East', 'East', 'East', 'East', 'East', 'East', 'South
st', 'East', 'North', 'North', 'East', 'East', 'East', 'East', 'South', 'South', 'South'
North', 'North', 'North', 'North', 'North', 'North', 'North', 'West', 'West', 'North', 'I
path length: 106
*** FAIL: Heuristic resulted in expansion of 1355 nodes
path: ['East', 'South', 'South', 'West', 'West', 'South', 'West', 'South', 'South', 'Eas
path length: 22
*** FAIL: Heuristic resulted in expansion of 37 nodes

### Question q6: 3/6 ###

Finished at 17:22:38

Provisional grades
=====
Question q4: 3/3
Question q6: 3/6
-----
Total: 6/9

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

```

Half the points for test q6 provided because we were unable to extend

6.2. Conclusions on the behavior of pacman, it is optimal (y / n), reaches the solution (y / n), nodes that it expands, etc (1.5pt)

6.2.1. Answer to question 5: heuristics

This heuristic tries to make pacman go to the corners. To achieve this we compare the distance from each tile to all the corners and use the biggest value as a reference to use. That's how the heuristic will help the user to find the best possible path. The heuristic calculation only takes into account the corners left to discover.

Section 7

This has been a very productive practice where we have been able to see in action all the knowledge about informed and uninformed search methods that we are acquiring in the theoretical classes. It has been very rewarding to see how the algorithms we provided were the basis for a game solving problem, on top of that, it was very easy to follow and provided extra information and hints that were very useful in order to obtain the functionality required. We have encountered some difficulties when it came to exercise 6 because we struggled to find an efficient heuristic that expanded less than 1200 and we found one (commented in the code) that should work but the autograder says it is inconsistent for some reason.

In a nutshell we are very fond of this practice and we reckon we have acquired all the necessary knowledge about the bases of the search algorithms.