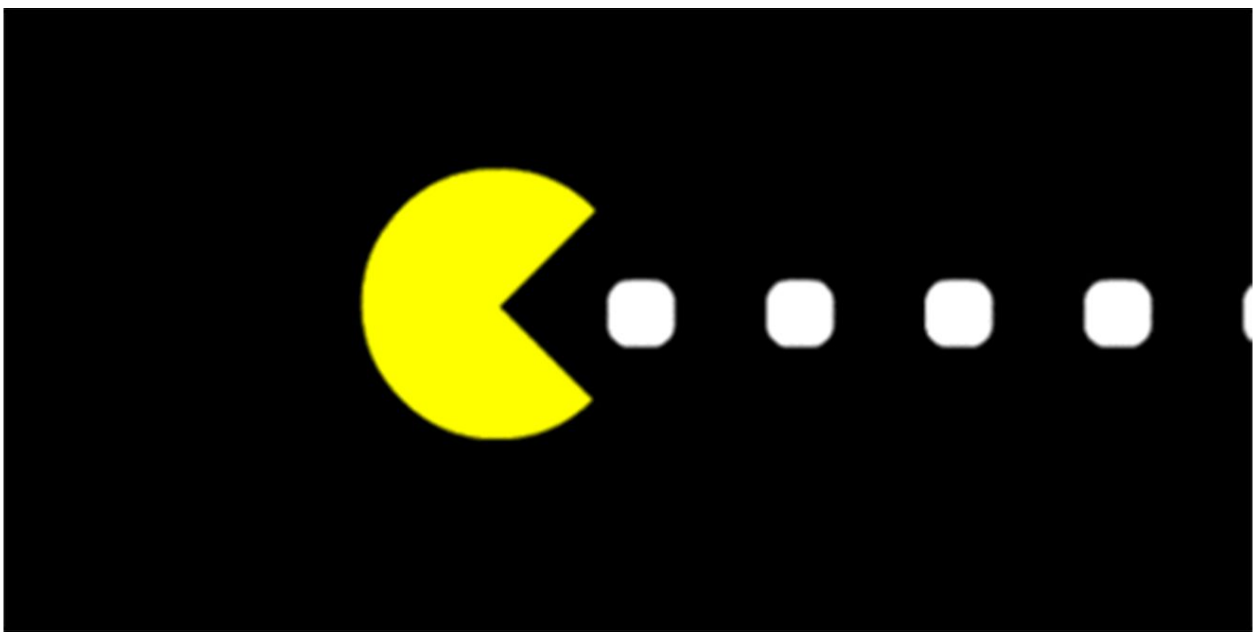
CSE 537
ARTIFICIAL INTELLIGENCE

Project 1 - PACMAN

REPORT

Submitted on 21st September 2018



Presented By:

Debjyoti Roy
112070373

Siddarth Harinarayanan
112026390

Question 1. Depth First Search

Solution Tiny Maze:

debjyotis-mbp:search debroy$ **python -m memory_profiler 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
Filename: search.py

```
Line #    Mem usage    Increment   Line Contents
================================================
    76   56.285 MiB   56.285 MiB   @profile
    77                             def depthFirstSearch(problem):
    78                                 """
    79                                 Search the deepest nodes in the search tree first.
    80
    81                                 Your search algorithm needs to return a list of actions that reaches the
    82                                 goal. Make sure to implement a graph search algorithm.
    83
    84                                 To get started, you might want to try some of these simple commands to
    85                                 understand the search problem that is being passed in:
    86
    87                                 print "Start:", problem.getStartState()
    88                                 print "Is the start a goal?", problem.isGoalState(problem.getStartState())
    89                                 print "Start's successors:", problem.getSuccessors(problem.getStartState())
    90                                 """
    91
    92                                 #### Referenced the logic and pseudocode of DFS from
https://en.wikipedia.org/wiki/Depth-first_search
    93   56.285 MiB    0.000 MiB       stack = []
    94   56.285 MiB    0.000 MiB       stack.append((problem.getStartState(), [])) #using a stack to store the nodes
being visited
    95   56.285 MiB    0.000 MiB       visited = []
    96   56.289 MiB    0.000 MiB       while len(stack) > 0:
    97                                     # print "stack ", stack
    98   56.289 MiB    0.000 MiB           node, directions = stack.pop(-1)      # popping the latest inserted node as that
would be the next level in the tree, as we are doing dfs
    99   56.324 MiB    0.035 MiB           if problem.isGoalState(node):        # check if we have reached goal node then
we can return the directions from the start to this goal state
   100   56.324 MiB    0.000 MiB               return directions
   101   56.289 MiB    0.000 MiB           if node not in visited:
   102   56.289 MiB    0.000 MiB               visited.append(node)
   103   56.289 MiB    0.000 MiB               successors = problem.getSuccessors(node)
   104   56.289 MiB    0.000 MiB               for successor in successors:     # for all the successors of the current node
push them in the stack
   105   56.289 MiB    0.004 MiB                   stack.append((successor[0], directions + [successor[1]]))
```

Solution Medium Maze:

```
debjyotis-mbp:search debroy$ python -m memory_profiler pacman.py -l mediumMaze
-p SearchAgent
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.1 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
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
    76   97.152 MiB   97.152 MiB   @profile
    77                             def depthFirstSearch(problem):
    78                                 """
    79                                 Search the deepest nodes in the search tree first.
    80
    81                                 Your search algorithm needs to return a list of actions that reaches the
    82                                 goal. Make sure to implement a graph search algorithm.
    83
    84                                 To get started, you might want to try some of these simple commands to
    85                                 understand the search problem that is being passed in:
    86
    87                                 print "Start:", problem.getStartState()
    88                                 print "Is the start a goal?", problem.isGoalState(problem.getStartState())
    89                                 print "Start's successors:", problem.getSuccessors(problem.getStartState())
    90                                 """
    91
    92                                 #### Referenced the logic and pseudocode of DFS from
https://en.wikipedia.org/wiki/Depth-first_search
    93   97.152 MiB    0.000 MiB       stack = []
    94   97.152 MiB    0.000 MiB       stack.append((problem.getStartState(), [])) #using a stack to store the nodes
being visited
    95   97.152 MiB    0.000 MiB       visited = []
    96   97.195 MiB    0.000 MiB       while len(stack) > 0:
    97                                     # print "stack ", stack
    98   97.195 MiB    0.004 MiB           node, directions = stack.pop(-1)      # popping the latest inserted node as that
would be the next level in the tree, as we are doing dfs
    99   97.457 MiB    0.219 MiB           if problem.isGoalState(node):        # check if we have reached goal node then we
can return the directions from the start to this goal state
   100   97.457 MiB    0.000 MiB               return directions
   101   97.195 MiB   -0.066 MiB           if node not in visited:
   102   97.195 MiB    0.000 MiB               visited.append(node)
   103   97.195 MiB    0.000 MiB               successors = problem.getSuccessors(node)
   104   97.195 MiB   -0.086 MiB               for successor in successors:     # for all the successors of the current node
push them in the stack
   105   97.195 MiB    0.051 MiB                   stack.append((successor[0], directions + [successor[1]]))
```

Solution medium Maze if not in reversed order of successors:

It takes 269 expanded nodes

E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l mediumMaze -z .5 -p SearchAgent -a fn=dfs --frameTime 0
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 246 in 0.0 seconds
Search nodes expanded: **269**
Pacman emerges victorious! Score: 264
Average Score: 264.0
Scores: 264.0
Win Rate: 1/1 (1.00)
Record: Win

Solution Large Maze:

debjyotis-mbp:search debroy$ **python -m memory_profiler pacman.py -l bigMaze -p SearchAgent --frameTime 0**
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.3 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
Filename: search.py

```
Line #    Mem usage    Increment   Line Contents
================================================
    76  142.652 MiB  142.652 MiB   @profile
    77                             def depthFirstSearch(problem):
    78                                 """
    79                                 Search the deepest nodes in the search tree first.
    80
    81                                 Your search algorithm needs to return a list of actions that reaches the
    82                                 goal. Make sure to implement a graph search algorithm.
    83
    84                                 To get started, you might want to try some of these simple commands to
    85                                 understand the search problem that is being passed in:
    86
    87                                 print "Start:", problem.getStartState()
    88                                 print "Is the start a goal?", problem.isGoalState(problem.getStartState())
    89                                 print "Start's successors:", problem.getSuccessors(problem.getStartState())
    90                                 """
    91
    92                                 #### Referenced the logic and pseudocode of DFS from
https://en.wikipedia.org/wiki/Depth-first_search
    93  142.652 MiB    0.000 MiB       stack = []
    94  142.652 MiB    0.000 MiB       stack.append((problem.getStartState(), [])) #using a stack to store the nodes
being visited
    95  142.652 MiB    0.000 MiB       visited = []
    96  142.809 MiB    0.000 MiB       while len(stack) > 0:
```

```
   97                              # print "stack ", stack
   98  142.809 MiB    0.000 MiB         node, directions = stack.pop(-1)      # popping the latest inserted node as that
would be the next level in the tree, as we are doing dfs
   99  143.262 MiB    0.457 MiB         if problem.isGoalState(node):        # check if we have reached goal node then
we can return the directions from the start to this goal state
  100  143.262 MiB    0.000 MiB             return directions
  101  142.809 MiB    0.000 MiB         if node not in visited:
  102  142.809 MiB    0.004 MiB             visited.append(node)
  103  142.809 MiB    0.055 MiB             successors = problem.getSuccessors(node)
  104  142.809 MiB    0.000 MiB             for successor in successors:     # for all the successors of the current node
push them in the stack
  105  142.809 MiB    0.094 MiB                 stack.append((successor[0], directions + [successor[1]]))
```

Question 2: Breadth First Search

Solution MediumMaze:

E:\StonyBrook CS\AI\ai\search>**C:\Python27\python -m memory_profiler 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 0.3 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
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   104   24.574 MiB   24.574 MiB   @profile
   105                             def breadthFirstSearch(problem):
   106                                 """Search the shallowest nodes in the search tree first."""
   107   24.582 MiB    0.008 MiB       queue = []
   108   24.582 MiB    0.000 MiB       queue.append((problem.getStartState(), []))
   109   24.582 MiB    0.000 MiB       visited = [problem.getStartState()]
   110   24.594 MiB    0.000 MiB       while len(queue) > 0:
   111                                     # print "queue ", queue
   112   24.594 MiB    0.000 MiB           node, directions = queue.pop(0)
   113   24.836 MiB    0.242 MiB           if problem.isGoalState(node):
   114   24.836 MiB    0.000 MiB               return directions
   115   24.594 MiB    0.000 MiB           successors = problem.getSuccessors(node)
   116   24.594 MiB    0.000 MiB           for successor in successors:
```

```
  117   24.594 MiB    0.000 MiB             if successor[0] not in visited:
  118   24.594 MiB    0.012 MiB                 queue.append((successor[0], directions +
[successor[1]]))
  119   24.594 MiB    0.000 MiB                 visited.append(successor[0])
```

Solution LargeMaze:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l
bigMaze -p SearchAgent -a fn=bfs --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.5 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
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   104   27.219 MiB   27.219 MiB   @profile
   105                             def breadthFirstSearch(problem):
   106                                 """Search the shallowest nodes in the search tree first."""
   107   27.227 MiB    0.008 MiB       queue = []
   108   27.227 MiB    0.000 MiB       queue.append((problem.getStartState(), []))
   109   27.227 MiB    0.000 MiB       visited = [problem.getStartState()]
   110   27.348 MiB    0.000 MiB       while len(queue) > 0:
   111                                     # print "queue ", queue
   112   27.348 MiB    0.000 MiB           node, directions = queue.pop(0)
   113   27.906 MiB    0.559 MiB           if problem.isGoalState(node):
   114   27.906 MiB    0.000 MiB               return directions
   115   27.348 MiB    0.070 MiB           successors = problem.getSuccessors(node)
   116   27.348 MiB    0.004 MiB           for successor in successors:
   117   27.348 MiB    0.000 MiB               if successor[0] not in visited:
   118   27.348 MiB    0.043 MiB                   queue.append((successor[0], directions + [successor[1]]))
   119   27.348 MiB    0.004 MiB                   visited.append(successor[0])
```

Solution Eight Puzzle Problem:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python eightpuzzle.py
A random puzzle:
-------------
| 3 | 1 | 2 |
-------------
| 4 | 7 | 5 |
-------------
| 6 |   | 8 |
```

```
-------------
BFS found a path of 3 moves: ['up', 'left', 'up']
After 1 move: up
-------------
| 3 | 1 | 2 |
-------------
| 4 |   | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
After 2 moves: left
-------------
| 3 | 1 | 2 |
-------------
|   | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
After 3 moves: up
-------------
|   | 1 | 2 |
-------------
| 3 | 4 | 5 |
-------------
| 6 | 7 | 8 |
-------------
Press return for the next state...
```

Question 3: Uniform Cost Search

Solution mediumMaze: This is the same as the BFS since the weights on the edges of the graph are all the same

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
svmem(total=4164263936L, available=679788544L, percent=83.7, used=3484475392L, free=679788544L)
Path found with total cost of 68 in 0.7 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
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   120   26.121 MiB   26.121 MiB   @profile
   121                             def uniformCostSearch(problem):
   122                                 """Search the node of least total cost first."""
   123   26.129 MiB    0.008 MiB       pQueue = util.PriorityQueue()
   124   26.129 MiB    0.000 MiB       pQueue.push((problem.getStartState(), []), 0)
   125   26.129 MiB    0.000 MiB       visited = []
   126
   127                                 # For each node in the priority queue, check if its goal state or append its successors
   128   26.141 MiB    0.000 MiB       while not pQueue.isEmpty():
   129   26.141 MiB    0.000 MiB           item = pQueue.pop()
   130   26.141 MiB    0.000 MiB           state = item[0]
   131   26.141 MiB    0.000 MiB           currentPath = item[1]
   132   26.141 MiB    0.000 MiB           if state in visited:  # If node already visited, continue to next node
   133   26.141 MiB    0.000 MiB               continue
   134
   135   26.141 MiB    0.000 MiB           visited.append(item[0])
   136                                     # If goal state, return with current Path as the solution
   137   26.406 MiB    0.266 MiB           if problem.isGoalState(item[0]):
   138   26.410 MiB    0.004 MiB               print(psutil.virtual_memory())
   139   26.410 MiB    0.000 MiB               return currentPath
   140
   141                                     # Get list of successors of current node and append it to priority queue if not visited
   142   26.141 MiB    0.000 MiB           successorsList = problem.getSuccessors(item[0])
   143   26.141 MiB    0.000 MiB           for x in successorsList:
   144   26.141 MiB    0.012 MiB               tempPath = list(currentPath)
   145   26.141 MiB    0.000 MiB               tempPath.append(x[1])
   146                                         # cost of 'tempPath' with heuristic value gives the approximate estimate of cost to goal
for priority queue
   147   26.141 MiB    0.000 MiB               pQueue.push((x[0], tempPath), (problem.getCostOfActions(tempPath)))
```

Solution MediumDottedMaze:

```
debjyotis-MacBook-Pro:search debroy$ python -m memory_profiler pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.5 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
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   124   97.102 MiB   97.102 MiB   @profile
   125                             def uniformCostSearch(problem):
   126                                 """Search the node of least total cost first."""
   127   97.102 MiB    0.000 MiB       pQueue = util.PriorityQueue()
   128   97.102 MiB    0.000 MiB       pQueue.push((problem.getStartState(), []), 0)
```

```
  129   97.102 MiB    0.000 MiB       visited = []
  130
  131                                 # For each node in the priority queue,
  132                                 # check if its goal state or append its successors
  133   97.117 MiB    0.000 MiB       while not pQueue.isEmpty():
  134   97.117 MiB    0.000 MiB           item = pQueue.pop()
  135   97.117 MiB    0.000 MiB           state = item[0]
  136   97.117 MiB    0.004 MiB           currentPath = item[1]
  137   97.117 MiB    0.000 MiB           if state in visited:  # If node already visited, skip processing it and continue to the
next item
  138   97.117 MiB    0.000 MiB               continue
  139
  140   97.117 MiB    0.000 MiB           visited.append(item[0])
  141   97.406 MiB    0.289 MiB           if problem.isGoalState(item[0]):   # return with current Path as the solution if
the goal state is reached
  142   97.406 MiB    0.000 MiB               return currentPath
  143
  144   97.117 MiB    0.004 MiB           successorsList = problem.getSuccessors(item[0])
  145   97.117 MiB    0.004 MiB           for x in successorsList:     # for all the successors of the current node push them
in the priority queue
  146   97.117 MiB    0.004 MiB               tempPath = list(currentPath)
  147   97.117 MiB    0.000 MiB               tempPath.append(x[1])
  148   97.117 MiB    0.000 MiB               pQueue.push((x[0], tempPath), (problem.getCostOfActions(tempPath)))
```

Solution MediumScaryMaze:

debjyotis-MacBook-Pro:search debroy$ **python -m memory_profiler pacman.py -l mediumScaryMaze -p StayWestSearchAgent**
Path found with total cost of **68719479864** in 0.3 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
Filename: search.py

```
Line #    Mem usage    Increment   Line Contents
================================================
  124   99.664 MiB   99.664 MiB   @profile
  125                             def uniformCostSearch(problem):
  126                                 """Search the node of least total cost first."""
  127   99.664 MiB    0.000 MiB       pQueue = util.PriorityQueue()
  128   99.664 MiB    0.000 MiB       pQueue.push((problem.getStartState(), []), 0)
  129   99.664 MiB    0.000 MiB       visited = []
  130
  131                                 # For each node in the priority queue,
  132                                 # check if its goal state or append its successors
  133   99.688 MiB    0.000 MiB       while not pQueue.isEmpty():
  134   99.688 MiB    0.000 MiB           item = pQueue.pop()
  135   99.688 MiB    0.000 MiB           state = item[0]
  136   99.688 MiB    0.000 MiB           currentPath = item[1]
  137   99.688 MiB    0.000 MiB           if state in visited:  # If node already visited, skip processing it and continue to
the next item
  138   99.684 MiB    0.000 MiB               continue
  139
  140   99.688 MiB    0.000 MiB           visited.append(item[0])
  141   99.883 MiB    0.199 MiB           if problem.isGoalState(item[0]):   # return with current Path as the solution if the
goal state is reached
```

```
  142   99.883 MiB    0.000 MiB            return currentPath
  143
  144   99.688 MiB   -0.086 MiB         successorsList = problem.getSuccessors(item[0])
  145   99.688 MiB    0.000 MiB         for x in successorsList:     # for all the successors of the current node push them
in the priority queue
  146   99.688 MiB    0.020 MiB            tempPath = list(currentPath)
  147   99.688 MiB    0.004 MiB            tempPath.append(x[1])
  148   99.688 MiB    0.000 MiB            pQueue.push((x[0], tempPath), (problem.getCostOfActions(tempPath)))
```

Question 4. A* search

Solution A*:

debjyotis-MacBook-Pro:search debroy$ **python -m memory_profiler pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic**
[SearchAgent] using function astar
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of **210** in 2.6 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
Filename: search.py

```
Line #    Mem usage    Increment   Line Contents
================================================
  159   76.648 MiB   76.648 MiB   @profile
  160                             def aStarSearch(problem, heuristic=nullHeuristic):
  161                                 """Search the node that has the lowest combined cost and heuristic first."""
  162
  163   76.648 MiB    0.000 MiB       pQueue = util.PriorityQueue()
  164   76.648 MiB    0.000 MiB       pQueue.push((problem.getStartState(), []), 0)
  165   76.648 MiB    0.000 MiB       visited = []
  166
  167                                 # For each node in the priority queue,
  168                                 # check if its goal state or append its successors
  169   76.734 MiB   -0.988 MiB       while not pQueue.isEmpty():
  170   76.734 MiB   -0.352 MiB           item = pQueue.pop()
  171   76.734 MiB   -1.645 MiB           state = item[0]
  172   76.734 MiB    0.004 MiB           currentPath = item[1]
  173   76.734 MiB   -0.309 MiB           if state in visited:  # If node already visited, skip processing it and continue to
the next item
  174   76.734 MiB   -0.145 MiB               continue
  175
  176   76.734 MiB   -0.164 MiB           visited.append(item[0])
  177   77.316 MiB    0.582 MiB           if problem.isGoalState(item[0]):    # return with current Path as the solution if
the goal state is reached
  178   77.316 MiB    0.000 MiB               return currentPath
  179
  180   76.734 MiB    0.066 MiB           successorsList = problem.getSuccessors(item[0])
  181   76.734 MiB   -0.109 MiB           for x in successorsList:   # for all the successors of the current node push them
in the priority queue
  182   76.734 MiB   -0.559 MiB               tempPath = list(currentPath)
  183   76.734 MiB   -0.023 MiB               tempPath.append(x[1])
```

```
  184                              # cost of 'tempPath' with heuristic value gives the approximate estimate of cost to goal
for priority queue
  185   76.734 MiB   -1.320 MiB             pQueue.push((x[0], tempPath),
(problem.getCostOfActions(tempPath)+heuristic(x[0], problem)))
```

What happens on open maze for various search strategies?

Below are the results for each of the search algorithms for an OpenMaze search problem. We observe that Astar reaches the goal state faster than all other strategies, expands only 535 nodes.

OpenMaze for Astar Search Algorithm:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l
OpenMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic --frameTime
0
[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.2 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
```

OpenMaze for BFS:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l
OpenMaze -z .5 -p SearchAgent -a fn=bfs --frameTime 0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.1 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
```

OpenMaze for DFS:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l
OpenMaze -z .5 -p SearchAgent -a fn=dfs --frameTime 0
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 298 in 0.1 seconds
Search nodes expanded: 576
Pacman emerges victorious! Score: 212
Average Score: 212.0
Scores:        212.0
```

```
Win Rate:     1/1 (1.00)
Record:       Win
```

OpenMaze for Uniform Cost Search:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python pacman.py -l OpenMaze -z .5 -p
SearchAgent -a fn=ucs --frameTime 0
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 54 in 0.2 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
```

Question 5. Finding all corners

Solution: BFS on tiny corners maze

We defined the **state space** of the problem to be a tuple containing two elements: 1. the current node of the pacman and 2. The list of already visited corners.

```
debjyotis-MacBook-Pro:search debroy$ python -m memory_profiler 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 0.3 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
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   107   56.246 MiB   56.246 MiB   @profile
   108                             def breadthFirstSearch(problem):
   109                                 """Search the shallowest nodes in the search tree first."""
   110   56.246 MiB    0.000 MiB       queue = []
   111   56.246 MiB    0.000 MiB       queue.append((problem.getStartState(), []))  # using a queue to store the nodes
being visited
   112   56.246 MiB    0.000 MiB       visited = [problem.getStartState()]
   113   56.277 MiB    0.000 MiB       while len(queue) > 0:
   114                                     # print "queue ", queue
   115   56.277 MiB    0.000 MiB           node, directions = queue.pop(0)   # popping the earliest inserted node as that
would be the same level in the tree, as we are doing bfs
   116   56.277 MiB    0.000 MiB           if problem.isGoalState(node):     # check if we have reached goal node then we
can return the directions from the start to this goal state
```

```
  117   56.277 MiB    0.000 MiB            return directions
  118   56.277 MiB    0.004 MiB         successors = problem.getSuccessors(node)
  119   56.277 MiB   -0.242 MiB         for successor in successors:
  120   56.277 MiB    0.000 MiB            if successor[0] not in visited:  # for all the successors of the current node push
them in the queue if that node is not visited
  121   56.277 MiB    0.031 MiB                queue.append((successor[0], directions + [successor[1]]))
  122   56.277 MiB    0.000 MiB                visited.append(successor[0])
```

Solution: BFS on MediumCorners

```
debjyotis-MacBook-Pro:search debroy$ python -m memory_profiler pacman.py -l
mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem
[SearchAgent] using function bfs
[SearchAgent] using problem type CornersProblem
Path found with total cost of 106 in 1.7 seconds
Search nodes expanded: 2448
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
  107   81.934 MiB   81.934 MiB   @profile
  108                             def breadthFirstSearch(problem):
  109                                 """Search the shallowest nodes in the search tree first."""
  110   81.934 MiB    0.000 MiB       queue = []
  111   81.934 MiB    0.000 MiB       queue.append((problem.getStartState(), []))  # using a queue to store the nodes
being visited
  112   81.934 MiB    0.000 MiB       visited = [problem.getStartState()]
  113   82.242 MiB   -0.129 MiB       while len(queue) > 0:
  114                                     # print "queue ", queue
  115   82.242 MiB    0.004 MiB         node, directions = queue.pop(0)   # popping the earliest inserted node as that
would be the same level in the tree, as we are doing bfs
  116   82.242 MiB   -0.012 MiB         if problem.isGoalState(node):     # check if we have reached goal node then we
can return the directions from the start to this goal state
  117   82.242 MiB    0.000 MiB            return directions
  118   82.242 MiB   -0.043 MiB         successors = problem.getSuccessors(node)
  119   82.242 MiB   -0.684 MiB         for successor in successors:
  120   82.242 MiB   -0.266 MiB            if successor[0] not in visited:  # for all the successors of the current node push
them in the queue if that node is not visited
  121   82.242 MiB   -0.094 MiB                queue.append((successor[0], directions + [successor[1]]))
  122   82.242 MiB    0.086 MiB                visited.append(successor[0])
```

Question 6. Corners Heuristic Implementation

Solution: We calculate the total manhattan distance from the current position to the nearest corner and to all the remaining corners from that corner based on the least manhattan distance from each corner to the next nearest corner.

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python -m memory_profiler pacman.py -l
mediumCorners -p AStarCornersAgent -z 0.5
Path found with total cost of 106 in 2.6 seconds
Search nodes expanded: 901
Pacman emerges victorious! Score: 434
Average Score: 434.0
Scores:        434.0
Win Rate:      1/1 (1.00)
Record:        Win
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   159   25.695 MiB   25.695 MiB   @profile
   160                             def aStarSearch(problem, heuristic=nullHeuristic):
   161                                 """Search the node that has the lowest combined cost and heuristic first."""
   162
   163   25.703 MiB    0.008 MiB       pQueue = util.PriorityQueue()
   164   25.707 MiB    0.004 MiB       pQueue.push((problem.getStartState(), []), 0)
   165   25.707 MiB    0.000 MiB       visited = []
   166
   167                                 # For each node in the priority queue,
   168                                 # check if its goal state or append its successors
   169   26.098 MiB    0.000 MiB       while not pQueue.isEmpty():
   170   26.098 MiB    0.000 MiB           item = pQueue.pop()
   171   26.098 MiB    0.000 MiB           state = item[0]
   172   26.098 MiB    0.000 MiB           currentPath = item[1]
   173   26.098 MiB    0.000 MiB           if state in visited:  # If node already visited, skip processing it and continue to
the next item
   174   26.098 MiB    0.000 MiB               continue
   175
   176   26.098 MiB    0.000 MiB           visited.append(item[0])
   177   26.098 MiB    0.000 MiB           if problem.isGoalState(item[0]):    # return with current Path as the solution if
the goal state is reached
   178   26.098 MiB    0.000 MiB               return currentPath
   179
   180   26.098 MiB    0.094 MiB           successorsList = problem.getSuccessors(item[0])
   181   26.098 MiB    0.000 MiB           for x in successorsList:   # for all the successors of the current node push them
in the priority queue
   182   26.098 MiB    0.219 MiB               tempPath = list(currentPath)
   183   26.098 MiB    0.004 MiB               tempPath.append(x[1])
   184                                         # cost of 'tempPath' with heuristic value gives the approximate estimate of cost to goal
for priority queue
   185   26.098 MiB    0.074 MiB               pQueue.push((x[0], tempPath),
(problem.getCostOfActions(tempPath)+heuristic(x[0], problem)))
```

Question 7. Food heuristic

Solution: We tried various heuristics for the solution to this problem.

Try 1: We tried to make the heuristic to be the distance from the current location of pacman to the nearest food. This was underestimating the actual cost and was expanding more search nodes than expected.

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python pacman.py -l trickySearch -p
AStarFoodSearchAgent
Path found with total cost of 60 in 24.3 seconds
Search nodes expanded: 13898
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:       570.0
Win Rate:     1/1 (1.00)
Record:       Win
```

Try 2: Next, we tried to use our corners heuristic idea here as well. Although it is admissible, the heuristic was again expanding more search nodes than expected. So we tried to improve the admissibility.

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python pacman.py -l trickySearch -p
AStarFoodSearchAgent
Path found with total cost of 60 in 37.8 seconds
Search nodes expanded: 14605
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:       570.0
Win Rate:     1/1 (1.00)
Record:       Win
```

Try 3: Next, we tried to calculate the average distance from the current node to the distance to all the remaining foods on the grid.

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python pacman.py -l trickySearch -p
AStarFoodSearchAgent
Path found with total cost of 60 in 22.1 seconds
Search nodes expanded: 11632
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:       570.0
Win Rate:     1/1 (1.00)
Record:       Win
```

Try 4: Finally, we tried the heuristic idea of adding the distance from the current node to the nearest food and the distance from the nearest found food location to the farthest food from it. This heuristic performed better than our previous heuristics and it is admissible as well.

```
debjyotis-MacBook-Pro:search debroy$ python pacman.py -l trickySearch -p
AStarFoodSearchAgent
Path found with total cost of 60 in 9.6 seconds
Search nodes expanded: 8178
Pacman emerges victorious! Score: 570
Average Score: 570.0
```

```
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
Filename: search.py

Line #    Mem usage    Increment   Line Contents
================================================
   159   63.320 MiB   63.320 MiB   @profile
   160                             def aStarSearch(problem, heuristic=nullHeuristic):
   161                                 """Search the node that has the lowest combined cost and heuristic first."""
   162
   163   63.320 MiB    0.000 MiB       pQueue = util.PriorityQueue()
   164   63.320 MiB    0.000 MiB       pQueue.push((problem.getStartState(), []), 0)
   165   63.320 MiB    0.000 MiB       visited = []
   166
   167                                 # For each node in the priority queue,
   168                                 # check if its goal state or append its successors
   169   93.242 MiB   -2.508 MiB       while not pQueue.isEmpty():
   170   93.242 MiB   -2.441 MiB           item = pQueue.pop()
   171   93.242 MiB   -2.438 MiB           state = item[0]
   172   93.242 MiB   -2.520 MiB           currentPath = item[1]
   173   93.242 MiB   -2.445 MiB           if state in visited:  # If node already visited, skip processing it and continue to
the next item
   174   93.141 MiB   -0.945 MiB               continue
   175
   176   93.242 MiB   -1.352 MiB           visited.append(item[0])
   177   93.242 MiB   -1.402 MiB           if problem.isGoalState(item[0]):    # return with current Path as the solution if
the goal state is reached
   178   93.242 MiB    0.000 MiB               return currentPath
   179
   180   93.242 MiB   22.570 MiB           successorsList = problem.getSuccessors(item[0])
   181   93.242 MiB   -3.195 MiB           for x in successorsList:   # for all the successors of the current node push them in
the priority queue
   182   93.242 MiB   -0.480 MiB               tempPath = list(currentPath)
   183   93.242 MiB   -2.809 MiB               tempPath.append(x[1])
   184                                         # cost of 'tempPath' with heuristic value gives the approximate estimate of cost to goal
for priority queue
   185   93.242 MiB   -2.672 MiB               pQueue.push((x[0], tempPath),
(problem.getCostOfActions(tempPath)+heuristic(x[0], problem)))
```

Food Search Problem with UCS:

```
E:\StonyBrook CS\AI\ai\search>C:\Python27\python pacman.py -l trickySearch -p SearchAgent -a
fn=ucs,prob=FoodSearchProblem
[SearchAgent] using function ucs
[SearchAgent] using problem type FoodSearchProblem
Path found with total cost of 60 in 48.6 seconds
Search nodes expanded: 16688
Pacman emerges victorious! Score: 570
Average Score: 570.0
Scores:        570.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Critical Analysis:

We infer the following points after working with solving the pacman puzzles for various mazes using different search algorithms:

1. For unweighted graphs like the tiny, medium and big mazes, DFS does not tend to always find the optimal solution unlike BFS and Uniform Cost Search. The search nodes expanded while using DFS may be lesser, as we encountered while checking the open maze problem, but the path taken is not optimal hence cost wise it is not optimal.
2. For weighted graphs like the mazes containing food, UCS and A* algorithms perform better to find the optimal solutions. A* tends to perform very well if the heuristic is consistent and is admissible and it performs better than UCS.