ARTIFICIAL INTELLIGENCE:
PROJEC T 1
DESIGNING OF SEARCH AGENTS USING PACMAN

TEAM MEMBERS:

1. AYA AHMED ALI
2. AYA MEDHAT MOSTAFA
3. AMIRA EID RAMADAN

PRODUCED TO:

Dr/ Mohamoud Atef

ENG/ MAI SHABAN

Contents:

1- Introduction

In this project we are experimenting with 4 main search algorithms.
_ Depth First Search
_ Breadth First Search
_ Gready Search
_ A* search

We will use python3 as out tool to code the search algorithms. After that we will compare these 4 different search algorithms and compare them in terms of:
_ Performance
_ Completeness
_ Optimality

1- Search with Single dot(goal):

1-2) Depth First Search

"We used a visited set for repeated state detection".
Depth First Search is an uninformed search strategy also called blind search strategy which means that the strategies have no additional information about the states other than the ones provided in the problem definition. It expands the deepest node in the current frontier of the search tree. It uses a **stack** to keep track of the generated nodes. As a stack follows LIFO principle, the last generated node is the first one chosen for expansion. Once a node is completely expanded, it is popped from the stack. The Depth First Search algorithm applied on pacman in small maze, medium maze and big maze.

★ Pseudo code of DFS Algorithm:

```
Order=W→N→S→E
Push the start cell in Frontier and Explored
Repeat until Goal is reached or Frontier is Empty:

    currCell=Frontier.pop=

    for each direction(ESNW):
        childCell=Next Possible Cell
        if childCell already in Explored list→Do nothing
        otherwise→ Append/Push childCell to both Explored & Frontier
```

★ **COMPLETE**

Depth First Search is complete if the tree is finite or else it is not complete (i.e, if the state space graph has cycles).

★ **OPTIMAL**

Depth First Search is not optimal. It only finds the leftmost solution in the search tree regardless of the depth or cost of the node.

★**Solutions**, path costs and n umbers of nodes expanded for each maze are as follows:

1- Medium maze
Path cost: 116
Node expanded: 138
Solution:

2- Big maze
Path cost: 238
Node expanded: 329
Solution:

3- Open Maze

Path cost: 238

Node expanded: 329

Solution:

1-3) Breadth First Search

"We also used a visited set for repeated state detection." Breadth first search is a level by level search. All the nodes in a particular level are expanded before moving on to the next level. It expands the shallowest node first. It uses a **queue** data structure to maintain a list of all the nodes which have been expanded. The Breadth First Search algorithm applied on pacman in small maze, medium maze and big maze

★ Pesudo Code

```
Add start cell in both Frontier and Explored
Repeat until Goal is reached or Frontier is Empty:
    currCell=Frontier.pop(0)=
    for each direction(ESNW):
        childCell=Next Possible Cell
        if childCell already in Explored list→Do nothing
        otherwise→ Append/Push childCell to both Explored & Frontier
```

★**COMPLETE**

Breadth First Search is complete because if a solution exits then the tree is finite.

★**OPT IMAL**

Breadth First Search is optimal only if the cost of all the arcs in the state space graph is the same.

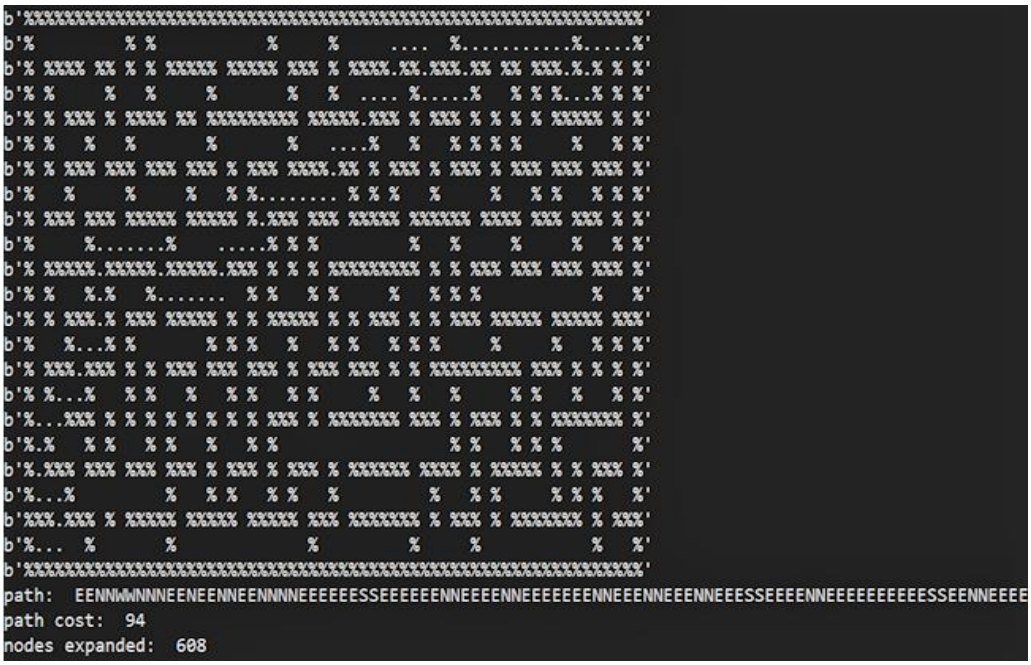
★**Solutions**, path costs and n umbers of nodes expanded for each maze are as follows:

1- Medium maze

Path cost: 94

Node expanded: 608

Solution:



2- Big maze

Path cost: 148

Node expanded: 1255

Solution:

3- Open Maze

Path cost: 45

Node expanded: 527

Solution:

```
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
b'%                     %.            %'
b'%                     %.            %'
b'%                     %.            %'
b'%                     %.            %'
b'%                     %......       %'
b'%                     %%%%%%.       %'
b'%                          %.       %'
b'%                          %.       %'
b'%                          %.       %'
b'%                          %.       %'
b'%       ..............     %.       %'
b'%       .%%%%%%%%%%%%........       %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .%                          %'
b'%       ..%                         %'
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
path:  SSSSEEEEESSSSSSSWWWWWWWNWWWWWWWWWWWWWSSSSSSSE
path cost:  45
nodes expanded:  527
```

1-4) Greedy Best First search

"we also used a visited set for repeated state detection". Gready search falls under the category of informed search strategy. The difference between the BFS is that we use the priority queue structure (manhattan distance as the key)to store the frontier nodes.

★**COMPLETE**
Gready Search is not complete is finite.

★**OPT IMAL**
Gready Search is not optimal

★**Solutions**, path costs and n umbers of nodes expanded for each maze are as follows:

1- Meduim Maze
Path cost: 94
Node expanded:103
Solution:

2- Big Maze

Path cost: 222

Node expanded: 290

Solution:

3- Open Maze

Path cost: 57

Node expanded: 155

Solution:

1-5) A* Search

A* search falls under the category of informed search strategy. This kind of search is also called best first search. A* search evaluates which node to combine using g(n) i.e, the cost to reach the node and h(n) i.e, the cost to get from the current node to the goal node, represented as : f(n) = g(n) + h(n) The A* Search algorithm applied on pacman in small maze, medium maze and big maze.

★**COMPLETE**

A* search is complete.

★**OPT IMAL**

A* search is optimal.

★ Pesudo Code

```
open=Priority Queue
g_score = {cell : infinity → for all cells and 0 for start cell}
f_score = {cell : infinity → for all cells and h(start) for start cell}

open.put → (f_score(start), h(start), start)
while open is Not empty or Goal reached:
    currCell → open.get cell value
    for each direction(ESNW):
        childCell = Next Possible Cell
        temp_g_score=g_score(currCell)+1
        temp_f_score= temp_g_score+h(childCell)

        if temp_f_score < f_score(chilCell):
            g_score(childCell) = temp_g_score
            f_score(childCell) = temp_f_score
            open.put → (f_score(childCell), h(childCell), childC
```

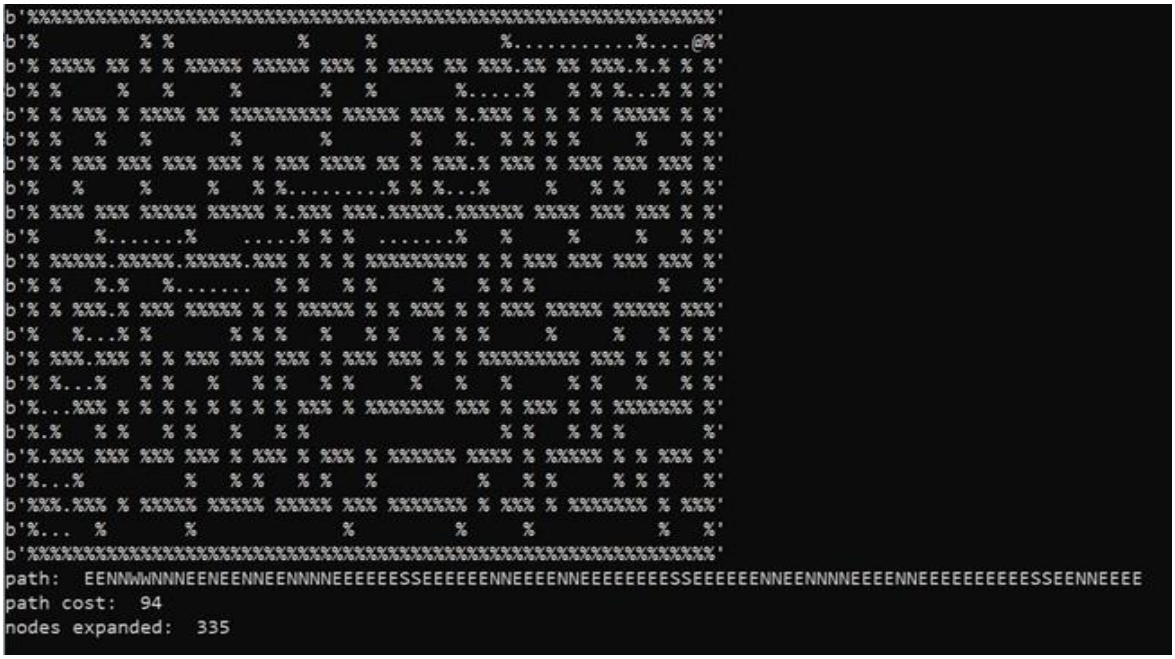
★**Solutions**, path costs and n umbers of nodes expanded for each maze are as follows:

1- Meduim Maze

Path cost: 94

Node expanded:335

Solution:



2- Big Maze

Path cost: 148

Node expanded: 1113

Solution:

3- Open Maze

Path cost: 148

Node expanded: 1113

Solution:

```
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
b'%                     %......       %'
b'%                     %     .       %'
b'%                     %     .       %'
b'%                     %     .       %'
b'%                     %     .       %'
b'%                     %%%%%%.       %'
b'%                          %.       %'
b'%                          %.       %'
b'%                          %.       %'
b'%                          %.       %'
b'%       ...................%.       %'
b'%       .%%%%%%%%%%%%     ...       %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .%                          %'
b'%       .@%                         %'
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
path:  EEEEESSSSSSSSSSSWWNWWWWWWWWWWWWWWWWWWSSSSSSSE
path cost:  45
nodes expanded:  238
```

1-6) Conclusion of Search with Single dot (goal):

The path costs and n umbers of nodes expanded for each algorithm on each maze is summarized

	Path cost			Expanded nodes		
	medium	big	open	mediu m	big	open
DFS	116	238	238	138	329	329
BFS	94	148	45	608	1255	527
Greedy	94	222	57	103	290	155
A*	94	148	148	335	1113	1113

From this table we can see that:

1. **In medium Maze**: Best Search Algorithm is (Gready), however both BFS, GREEDY and A* have the same Path Cost but Gready Search used the least Expanded nodes**.**
2. **In Big Maze**: Best Search Algorithm is (A*), however both BFS and A* have the same Path Cost but A* Search used the least Expanded nodes**.**
3. **In Open Maze**: Best Search Algorithm is (BFS), because BFS search take the least Path Cost**.**

2- Search with multiple dots(goals):

-- Using A* Search Algorithm:

1- Tiny search
Path cost: 7
Node expanded: 10
Solution:

```
b'%%%%%%%%%%'
b'%    %    %'
b'% % % %% %'
b'% %     % %'
b'%  %.%    %'
b'%    .....%'
b'% %%%% %.%'
b'%        %@%'
b'%%%%%%%%%%'
path:  SEEEESS
path cost:  7
nodes expanded:  10
```

2- Small search
Path cost: 165
Node expanded: 332
Solution:

```
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
b'%      .........%...   .........%'
b'%    %%%%%.%%%%%%.%. %.%.%%   .%'
b'%    %. % ....%   .%. %.%..%   .%'
b'%..... %      ......%....% .%%%%%'
b'%%%%%.%%%% %%% .%%%%%%% .....%'
b'%..........     ...        % %%% .%'
b'%%.%%%%%%%%.%%%%%%%%%%%% %....%'
b'% .       %   ...............%.%%%%'
b'%..         %%%%%.....%........    %'
b'%.% %%%      %  .%    %% %%.%%%%%%'
b'%               %  .%           .....@%'
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
path:  EEEEEEEWWWWWSSEEEESEEEESSEEWWN
NENNNEEESSSSEEENNNEEEEEEEEESSNNWWWWWSSE
SSEEEEESSWWWSSWWWWWWNWWSWWWWSSNNNWWN
NWWWWWWNNNSWWWWEEEEESSWWWWESSSWSNENN
NEEEEEEEEEESSEEEEEEEEEEEEEESSSEEEEE
path cost:  165
nodes expanded:  332
```

3- Medium search

Path cost: 256

Node expanded: 591

Solution:

```
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
b'% ...%   ......%............% .%     % %   %@.%'
b'% . .%%%%.%%%%%%.%  % % %%   .......%%%%  %  %%%.%'
b'% . .%.% ...% .%.%  % %  %   %  .%.........%  .%'
b'%.. .............%......%%%% %  .%% .%%%.% .....%'
b'%.%%%%%.%%%%%%  .%%%%.% ..........% .  %..%%%%%%%'
b'%...    .%....% .......% %%%  %%%%%% .% %..%...% %'
b'%%%.%% .%.%%%%%%% %%%%% %    %  ..%%.% %....%.% %'
b'%  . % .%.     %  %      .% %%  .....%  .%%%%.% %'
b'%.......%.  %  %  %     %........  %     ....%...%'
b'% % %%%... % %    %% %% %%% %% %  % %%%%%%%%%%%.%'
b'%   %....%    %              %    %        .....%'
b'%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%'
path:  SEEEEEEENNESEEENNNNNEEEESSSSSSEEEWWWNNEEENEESSSE
ESSWWWWEEEENNWWNNNWWSWWNNWNNWWWWWWNWWNSSSSEWWWWWWWWWWNW
WWWWEESSEWWWWWWNNWNSWWWWWWWWNSWWNNNWWSSSWSSEESSSWWEEEEE
ESSWWEEENENNNNEEEWWWSSSSWWNNNNNNEENNNEEEEEWWWWWSSEESEEE
EENNNEEEEEEEEEEEEESEEEEEEESEEEEEEEEEESEEEEENNNW
path cost:  256
nodes expanded:  591
```

-- Using BFS Algorithm:

Tiny search

Path cost: 7

Node expanded: 38

Solution:

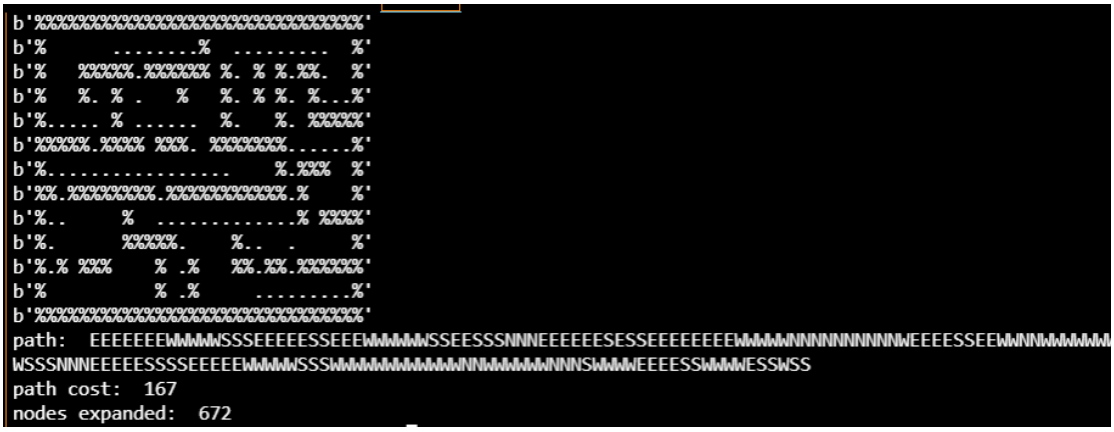
```
b'%%%%%%%%%%'
b'%   %    %'
b'% % % %% %'
b'% %    % %'
b'%  %.%   %'
b'%   .....%'
b'% %%%% %.%'
b'%      %.%'
b'%%%%%%%%%%'
path:  SEEEESS
path cost:  7
nodes expanded:  38
```

Small search

Path cost: 167

Node expanded: 672

Solution:



Medium search

Path cost: 246

Node expanded: 1195

Solution:

-- Conclusion of Search with Single dot (goal):

	Path cost			Expanded nodes		
	tiny	small	medium	tiny	small	**medium**
BFS	7	167	246	38	672	**1195**
A*	**7**	**165**	**256**	**10**	**332**	**591**

From this table we can see that:

1. **In small search** : Best Search Algorithm is (A*), because A* search take the least Path Cost**.**
2. **In tiny search** : Best Search Algorithm is (A*), however both BFS and A* have the same Path Cost but A* Search used the least Expanded nodes**.**
3. **In Medium Maze**: Best Search Algorithm is (BFS), because BFS search take the least Path Cost**.**