PROJECT - 1
CS 246: Artificial Intelligence

Designing of search agents using Pacman

Submitted By - The Holy Trinity

Jimut Bahan Pal
Debadri Chatterjee
Sounak Modak

Submitted to the Department of Computer Science
in partial fulfilment of the requirements
for the degree of M.Sc.

RKMVERI, Belur Math
Howrah - 711202

August, 2019

ARTIFICIAL INTELLIGENCE : PROJECT 1

DESIGNING OF SEARCH AGENTS USING PACMAN

TEAM: THE HOLY TRINITY

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In this project we are experimenting with 4 main search algorithms.

- Depth First Search
- Breadth First Search
- Uniform Cost Search
- A* search

The Pacman AI projects were developed at UC Berkeley, primarily by John DeNero and Dan Klein [1] for educational purposes. 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

2 DEPTH FIRST SEARCH

PROBLEM STATEMENT : FIND A FIXED FOOD DOT USING DEPTH FIRST SEARCH ALGORITHM

Depth First Search [2] 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 are shown in Figure 1, Figure 2, and Figure 3.

ALGORITHM

```
procedure DFS(G,v):
    let S be a stack
    S.push(v)
    while S is not empty
        v = S.pop()
        if v is not labeled as discovered:
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

COMPLEXITY

The complexity of Depth First Search is : $O(b^m)$
where b is the branching factor and m is the maximum depth of the tree.
The space complexity of Depth First Search is : O(bm)
where b and m mean the same things as before.

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.

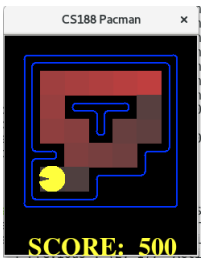

Figure 1: DFS on tiny maze.

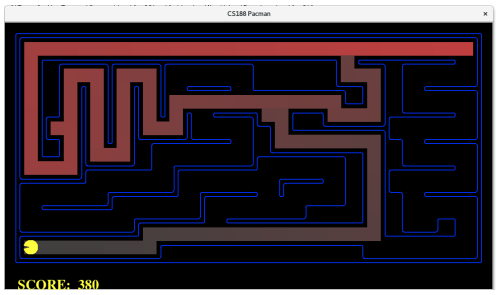

Figure 2: DFS on medium maze.

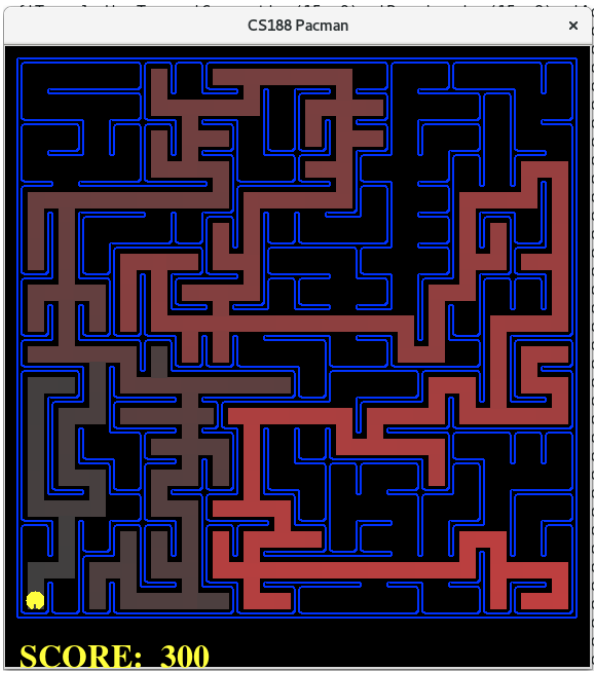

Figure 3: DFS on Big maze.

3 BREADTH FIRST SEARCH

PROBLEM STATEMENT : FIND A FIXED FOOD DOT USING BREADTH FIRST SEARCH ALGORITHM

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 are shown in Figure 4, Figure 5, and Figure 6.

ALGORITHM

function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure

```
node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
frontier ← a FIFO queue with node as the only element
explored ← an empty set
loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /*chooses the shallowest node in the frontier*/
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem,node,action)
        if child.STATE is not in explored or frontier then
            if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
            frontier ← INSERT (child,frontier)
```

COMPLEXITY

The time complexity of Breadth First Search is : $O(b^d)$
where b is the branching factor of the search tree and d is the depth at which the shallowest goal node is situated.
The space complexity of Breadth First Search is : $O(b^d)$ i.e, it is dominated by the size of the frontier.

COMPLETE

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

OPTIMAL

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

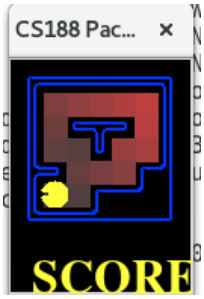

Figure 4: BFS on tiny maze.

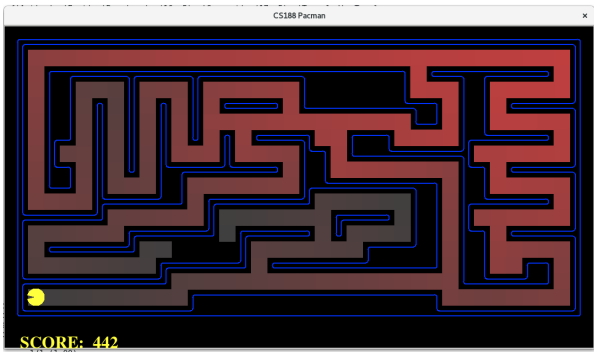

Figure 5: BFS on medium maze.

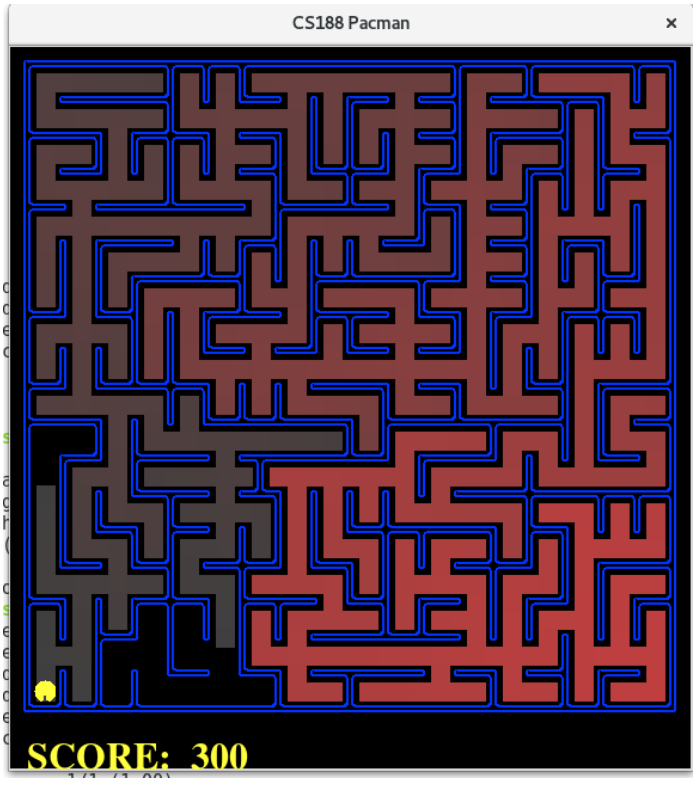

Figure 6: BFS on Big maze.

4 UNIFORM COST SEARCH

PROBLEM STATEMENT : FIND A FIXED FOOD DOT BY VARYING THE COST FUNCTION AND USING UNIFORM COST SEARCH ALGORITHM

Uniform Cost Search ,instead of expanding the shallowest node like Breadth First Search, expands the node n with the lowest cost path g(n). This is why the frontier is stored as a priority queue ordered by g. Uniform Cost Search applies the goal test to a node once it is selected for expansion rather than when it is first generated(as in the case of Breadth First Search and Depth First Search). This is because the first goal node that is generated may be on a sub-optimal path. It also includes a test to check whether a better goal state has been encountered. The Uniform Cost Search algorithm applied on pacman in small maze, medium maze and big maze are shown in Figure 7, Figure 8, and Figure 9.

ALGORITHM

function UNIFORM-COST-SEARCH(problem) returns a solution,or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier) /*chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem,node,action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child,frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child

COMPLEXITY

Uniform-cost search is guided by path costs rather than depths, so its complexity is not easily characterized in terms of b and d. Instead, let c^* be the cost of the optimal solution, and assume that every action costs at least ϵ. Then the algorithm's worst-case time and space complexity is $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, which can be much greater than b^d.
The space complexity of this algorithm is : $O(b^{\lfloor C^*/\epsilon \rfloor})$

COMPLETE

Uniform Cost Search is complete if the optimal solution has a finite cost and there are no arcs with negetive cost.

OPTIMAL

Yes Uniform Cost Search is optimal.

Figure 7: UCS on tiny maze.

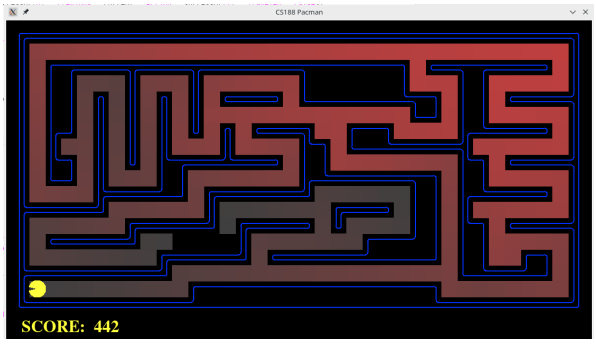

Figure 8: UCS on medium maze.

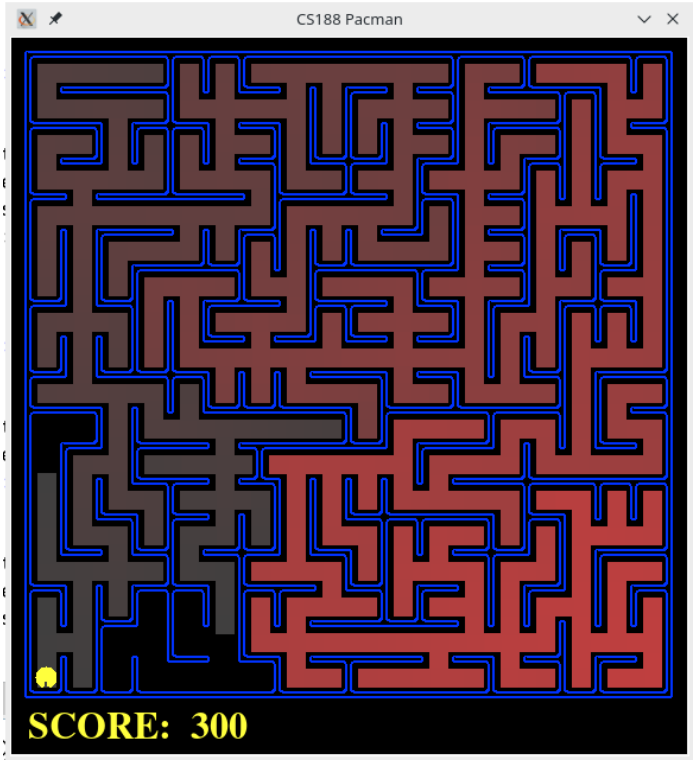

Figure 9: UCS on Big maze.

5 A* SEARCH

PROBLEM STATEMENT : FIND A FIXED FOOD DOT USING A* SEARCH ALGORITHM

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 are shown in Figure 10, Figure 11, and Figure 12.

ALGORITHM

```
function reconstruct_path(cameFrom, current)
    total_path := current
    while current in cameFrom.Keys:
        current := cameFrom[current]
        totalPath.prepend(current)
    return totalPath
function A_Star(start, goal, h)
openSet := start
    cameFrom := an empty map
    gScore := map with default value of Infinity
    gScore[start] := 0
    fScore := map with default value of Infinity
    fScore[start] := h(start)
    while openSet is not empty
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)
        openSet.Remove(current)
        closedSet.Add(current)
        for each neighbor of current
            if neighbor in closedSet
                continue
            tentative_gScore := gScore[current] + d(current, neighbor)
            if neighbor not in openSet
                openSet.add(neighbor)
            if tentative_gScore < gScore[neighbor]
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := gScore[neighbor] + h(neighbor)
    return failure
```

COMPLEXITY

The time complexity of A* depends on the heuristic. In the worst case of an unbounded search space it will be : $O(b^d)$
where b is the branching factor and d is the depth at which the solution resides in the tree. The space complexity of A* is : $O(b^d)$

COMPLETE

A* search is complete

OPTIMAL

A* search is optimal

Figure 10: A-star on tiny maze.

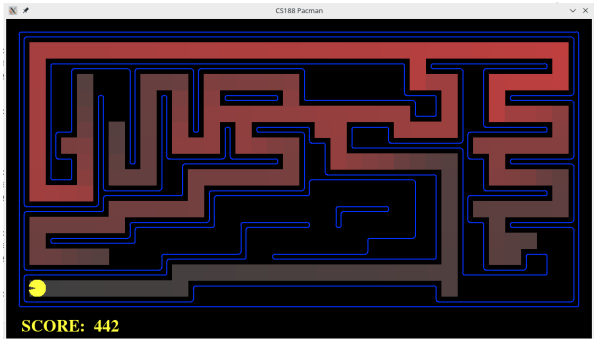

Figure 11: A-star on medium maze.

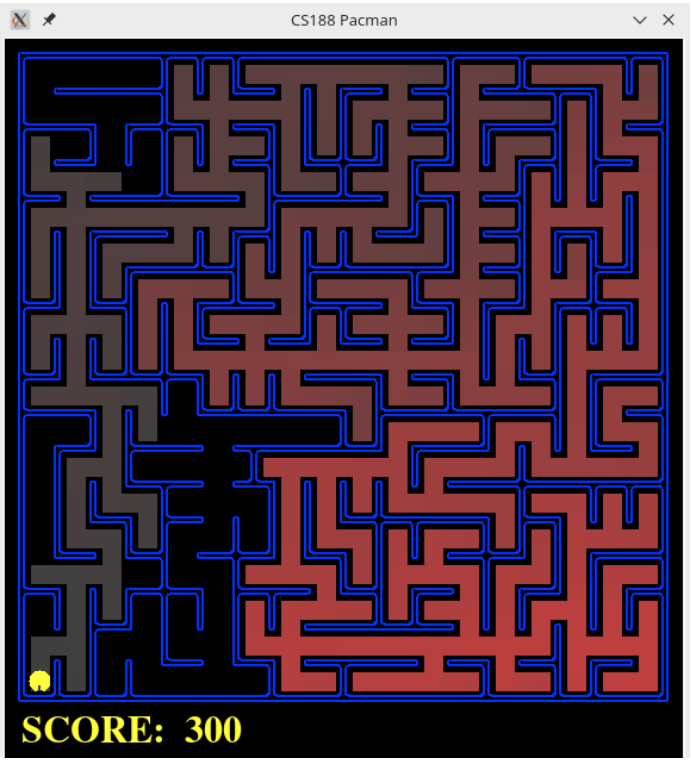

Figure 12: A-star on Big maze.

6 RESULTS AND DISCUSSION

We have worked with 3 uninformed search strategies and 1 informed search strategy. Obviously the informed search strategy worked better than the uninformed search strategies. Among the three uninformed search strategies Uniform Cost Search gives the best time complexity. But uniform cost search is useful only when the path costs are different otherwise it reduces to breadth first search. If all the

path costs are the same then breadth first search gives us the best time complexity of $O(b^d)$. But it has an exponential space complexity. Depth First Search gives us a linear space complexity but its time complexity is same as Breadth First Search. Furthermore DFS is neither complete nor optimal whereas the other two are complete and optimal(given certain conditions). The informed search strategy: A* search is optimal if we use admissible heuristics and consistent path cost(in case of graph search). But like BFS it has an exponential space complexity. All the results obtained from the experiment are shown in Table 1.

REFERENCES

[1] The Pac-Man Projects, UC Berkeley CS188 Intro to AI – Course Materials, available online at http : //ai.berkeley.edu/project$_o$verview.html, last accessed on 27th August, 2019.

[2] Russell, S., Norvig, P. (2010). Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ: Prentice Hall.

Table 1: Table of Results

Search algorithm used	Small Maze	Medium Maze	Big Maze	Description
Depth First Search	10	130	210	Cost
	15	146	390	Search Node Expanded
	500	380	300	Average Score
	500	380	300	Scores
	1/1	1/1	1/1	Win Rate
	Win	Win	Win	Record
Breadth First Search	8	68	210	Cost
	15	269	620	Search Node Expanded
	502	442	300	Average Score
	502	442	300	Scores
	1/1	1/1	1/1	Win Rate
	Win	Win	Win	Record
Uniform Cost Search	8	68	210	Cost
	15	269	620	Search Node Expanded
	502	442	300	Average Score
	502	442	300	Scores
	1/1	1/1	1/1	Win Rate
	Win	Win	Win	Record
A* using Manhattan Distance	8	68	210	Cost
	14	221	549	Search Node Expanded
	502	442	300	Average Score
	502	442	300	Scores
	1/1	1/1	1/1	Win Rate
	Win	Win	Win	Record