**IMPLEMENTATION OF SEARCH ALGORITHMS IN PACMAN GAME IN CONDITIONED ENVIRONMENT**

Gokul Balagopal

2MSEE, Department of ECE, University of Houston

PSID: 1590661, gbalagopal@uh.edu

**INTRODUCTION**

The following report explains implementations of various search algorithms like Depth First Search, Breadth First Search, A\* and others in a PacMan game environment under various constraints. In each problem, Pacman agent is required to find paths through the maze efficiently either to reach a location or collect the food quickly not violating the legality of maze like invalid directions, moving through wall etc. Its movement is governed by search algorithms and other conditions as stipulated by the problem in various Pacman scenarios. The code for its working is consisted in various Python code files but for implementation primarily Search.py and SearchAgents.py files were edited.

**1. PROBLEM 1: FINDING A FIXED FOOD SPOT USING DEPTH FIRST SEARCH**

*1.1 DEPTH FIRST SEARCH:*

Depth first search(DFS) is a type of Uninformed search meaning there is no additional information about the states other than that provided by the problem definition. In DFS the strategy is to expand the deepest node of current frontier in the search tree until it has no successors. As the nodes are expanded they are removed from the frontier (which is a queue containing unexpanded nodes) so that the search now shifts to the next deepest unexpanded node. DFS uses Last-in-First-out type of queue as its frontier.

*1.2 GRAPH SEARCH ALGORITHM FOR DFS:*

**function** GRAPH-SEARCH(problem) **returns** a solution, or failure

frontier=initial\_state

*explored=[]*

**loop do**

**if** the frontier is empty **then return** failure

choose a leaf node and remove it from the frontier

**if** the node=goal\_state **then return** the corresponding solution

add the node to the explored set

expand the chosen node, adding the resulting nodes to the frontier

***only if not in the frontier or explored set***

*1.3 TINY MAZE:*

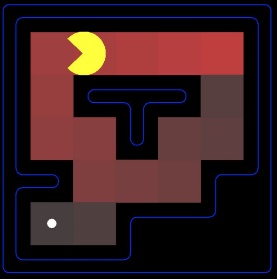


Figure 1.3: python pacman.py -l tinyMaze -p SearchAgent

*1.4 MEDIUM MAZE:*

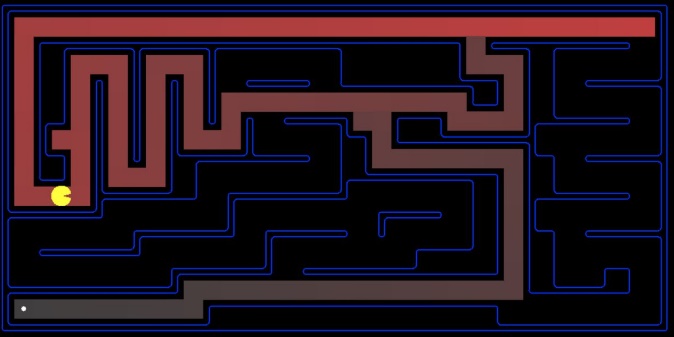


Figure 1.4: python pacman.py -l mediumMaze -p SearchAgent

*1.5 BIG MAZE:*

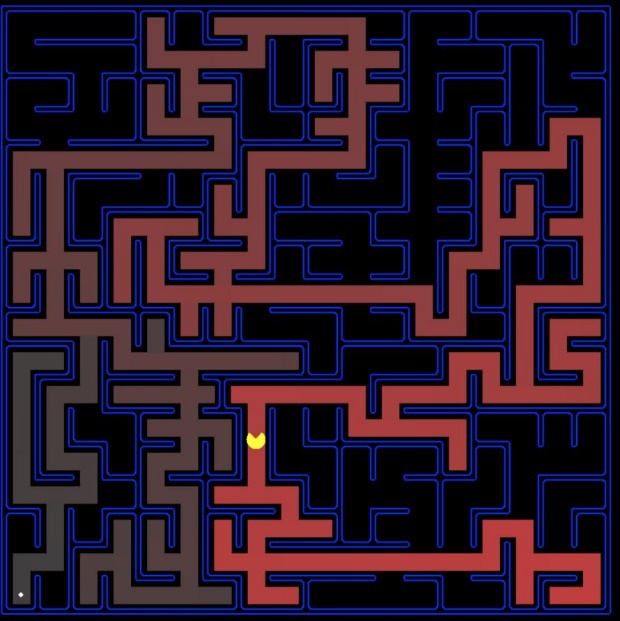


Figure 1.5: python pacman.py -l bigMaze -z .5 -p SearchAgent

**2. PROBLEM 2: FINDING A FIXED FOOD SPOT USING BREADTH FIRST SEARCH**

*2.1 BREADTH FIRST SEARCH:*

Breadth first search(BFS) is also a type of Uninformed search where all the nodes in the search tree at a given depth are expanded first before expanding the nodes of next level. Unlike DFS, BFS uses FIFO type queue for its frontier. BFS differs slightly from graph search algorithm by performing goal test when the node is generated and not when it is selected for expansion.

*2.2 GRAPH SEARCH ALGORITHM FOR BFS:*

**function** BFS(problem) **returns** a solution, or failure

node.STATE = problem.INITIAL-STATE, PATH-COST = 0

**if** problem.GOAL-TEST(node.STATE) **then return** SOLUTION(node)

frontier = stack()

explored=[]

**loop do**

**if** EMPTY?( frontier) **then return** failure

node=POP( frontier ) ##choosing 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 )

*2.3 BIG MAZE:*

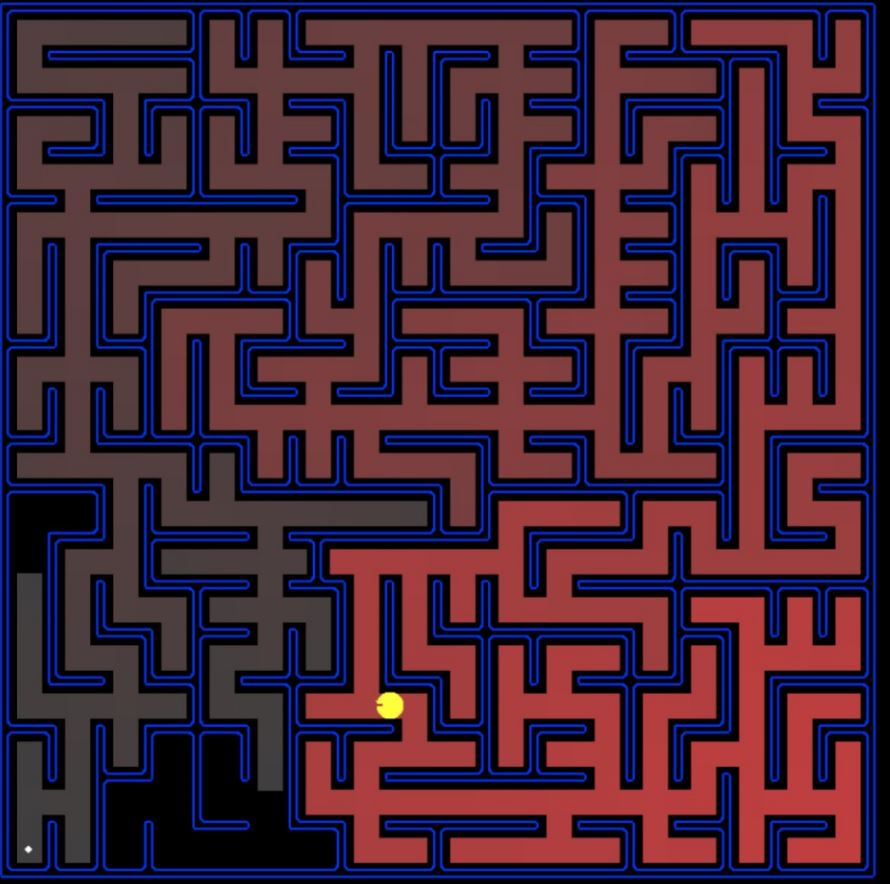


Figure 2.3: python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

*2.4 MEDIUM MAZE:*

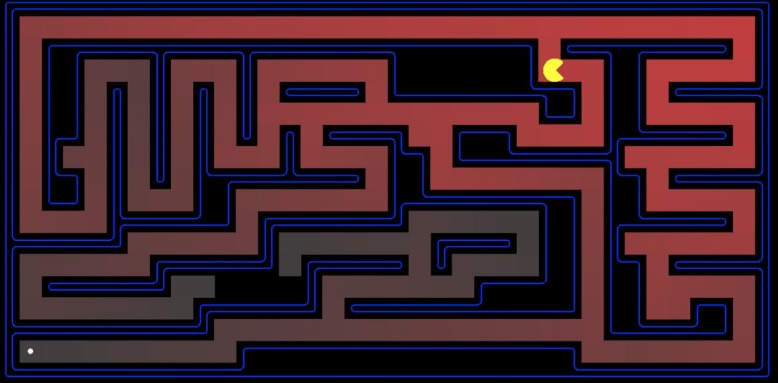


Figure 2.4: python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

*2.5 DOES BFS FIND LEAST COST SOLUTION?*

BFS is guaranteed to find the path of least cost because BFS always expands the shallowest expanded node and any new path to the state already in the frontier is discarded. In that case this new path must be as deep as the one already found the frontier; hence its cost must be the least.

**3. PROBLEM 3: FINDING A FIXED FOOD SPOT USING UNIFORM COST SEARCH**

*3.1 UNIFORM COST SEARCH:*

Uniform cost search(UCS) is an Uninformed search where the node n corresponding to lowest cost (given by cost g(n)) is expanded. Here the frontier is a priority queue where the nodes are ordered based on g (in other words each node has depth information). Goal test is performed when the node is selected for expansion, also another test is performed to check if a better path was found to the node currently in the frontier.

*3.2 MEDIUM MAZE:*

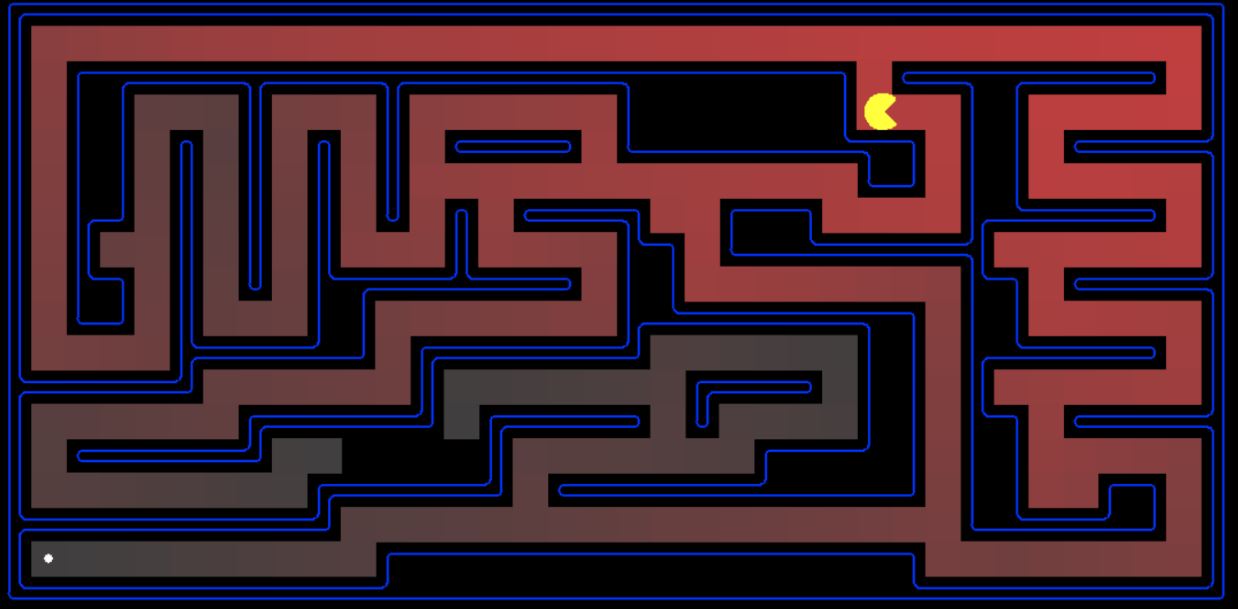


Figure 3.2: python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

*3.3 MEDIUM-DOTTED MAZE:*

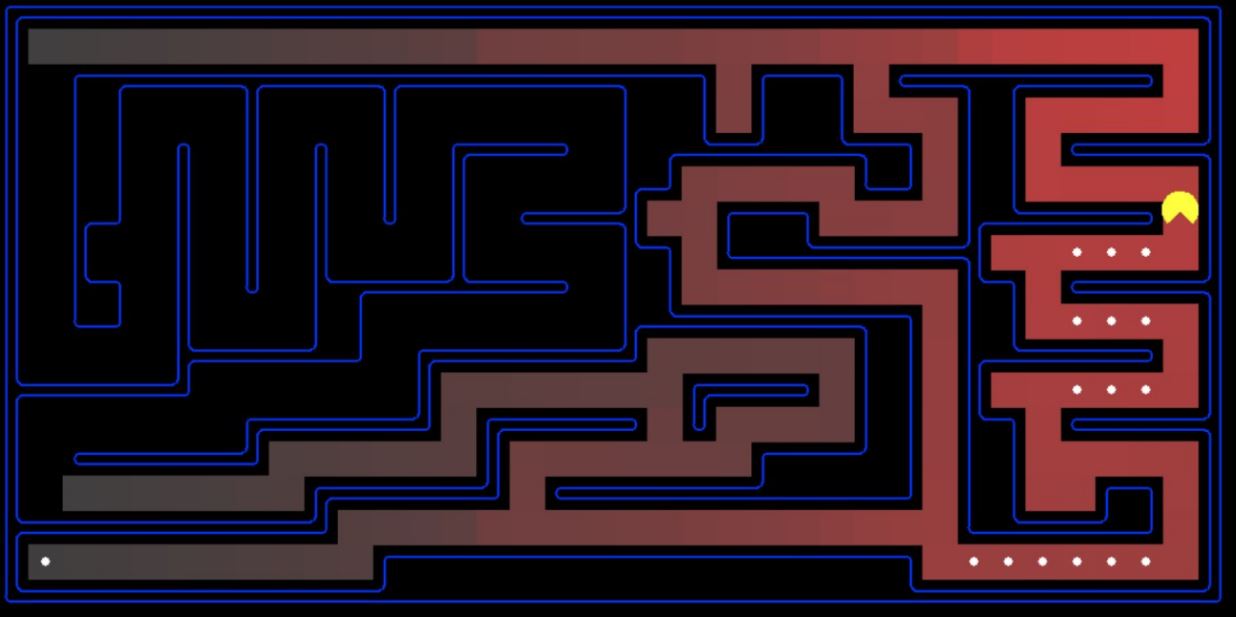


Figure 3.3: python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

*3.4 MEDIUM-SCARY MAZE:*

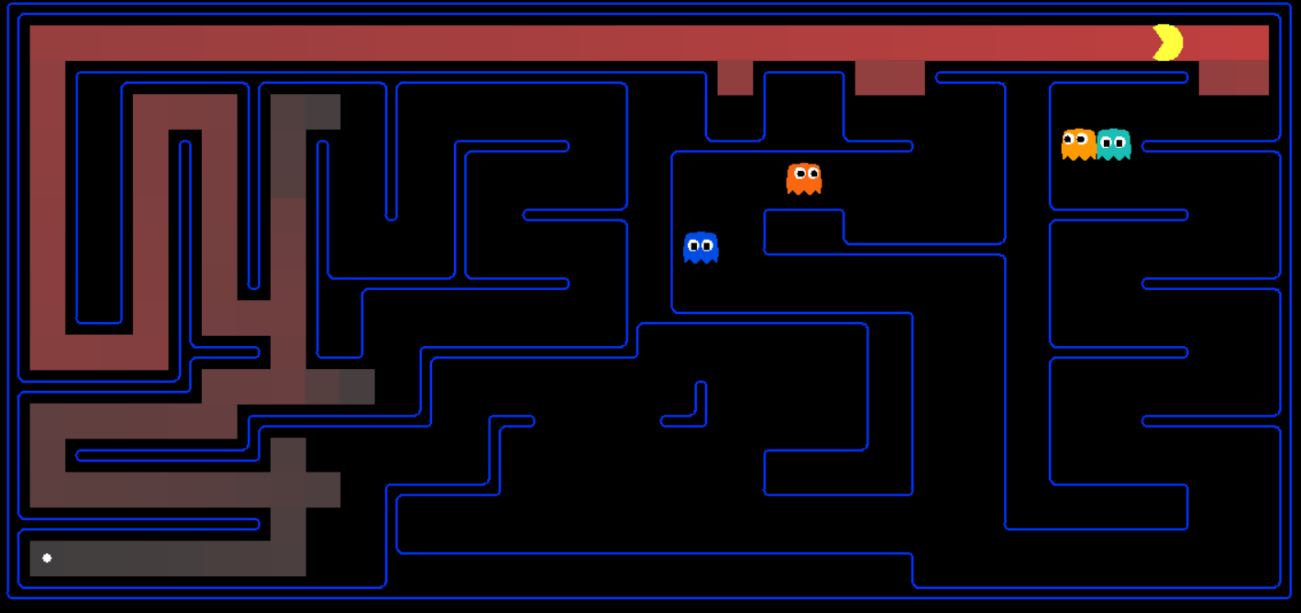


Figure 3.4: python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

**4. PROBLEM 4: FINDING A FIXED FOOD SPOT USING A\* SEARCH**

*4.1 A\* SEARCH:*

A-Star search(A\*) is an Informed search i.e. it used problem specific knowledge beyond definition of problem. In a general case of Informed search, the node to expanded is decided by the value of evaluation function f(n) which is an estimate of cost, so the node with lowest evaluation is expanded first. In case of A\*, the value of f(n) is the sum of g(n), which is the path cost from start node to n, and h(n) called the heuristic function is the estimated cost of cheapest path from node n to goal. So :

f(n) = g(n) + h(n)

In the given problem, h(n) is already implemented as Manhattan heueristic function.

*4.2 BIG MAZE – with Manhattan Heuristic:*

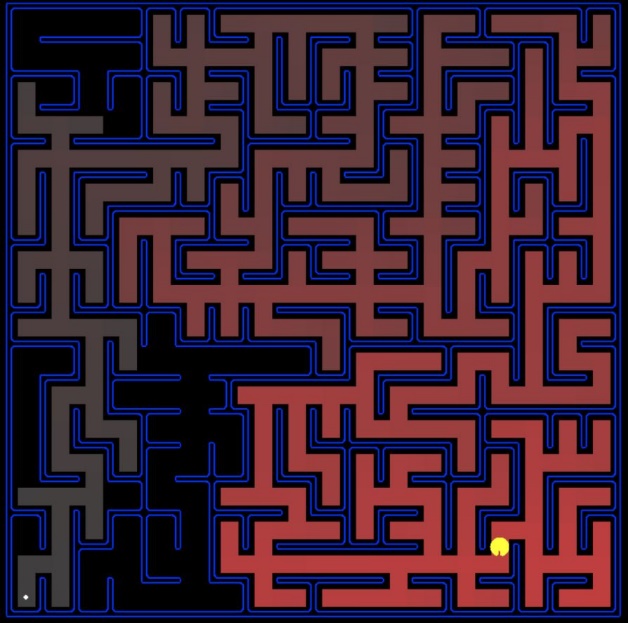


Figure 4.2: python pacman.py -l bigMaze -z .5 -p SearchAgent -a fn=astar,heuristic=manhattanHeuristic

*4.3 HOW MANY NODES DID A\* EXPAND WHEN DOING THE BIG MAZE?*

Total 549 search nodes expanded.

*4.4 HOW DOES IT COMPARE TO UCS ?*

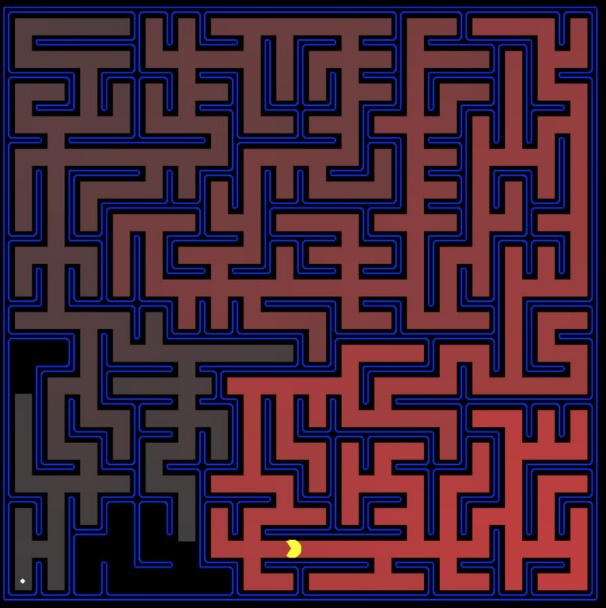
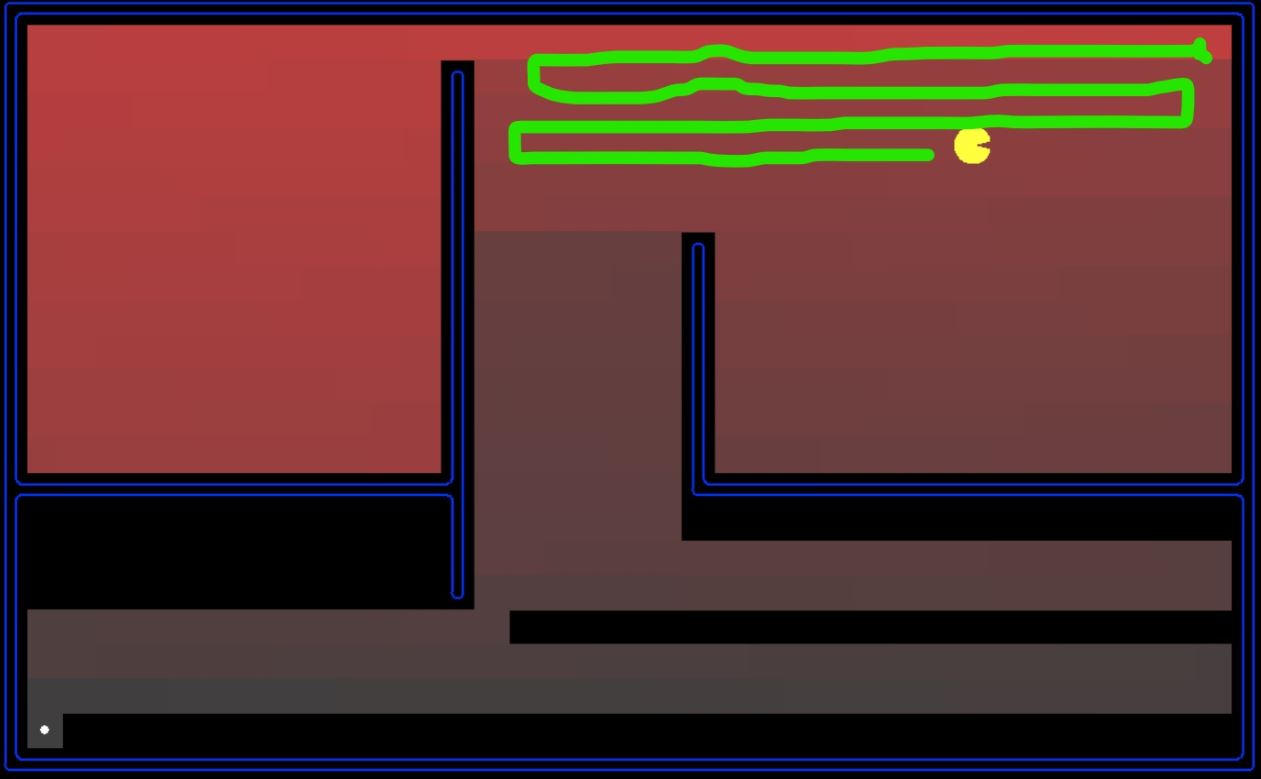


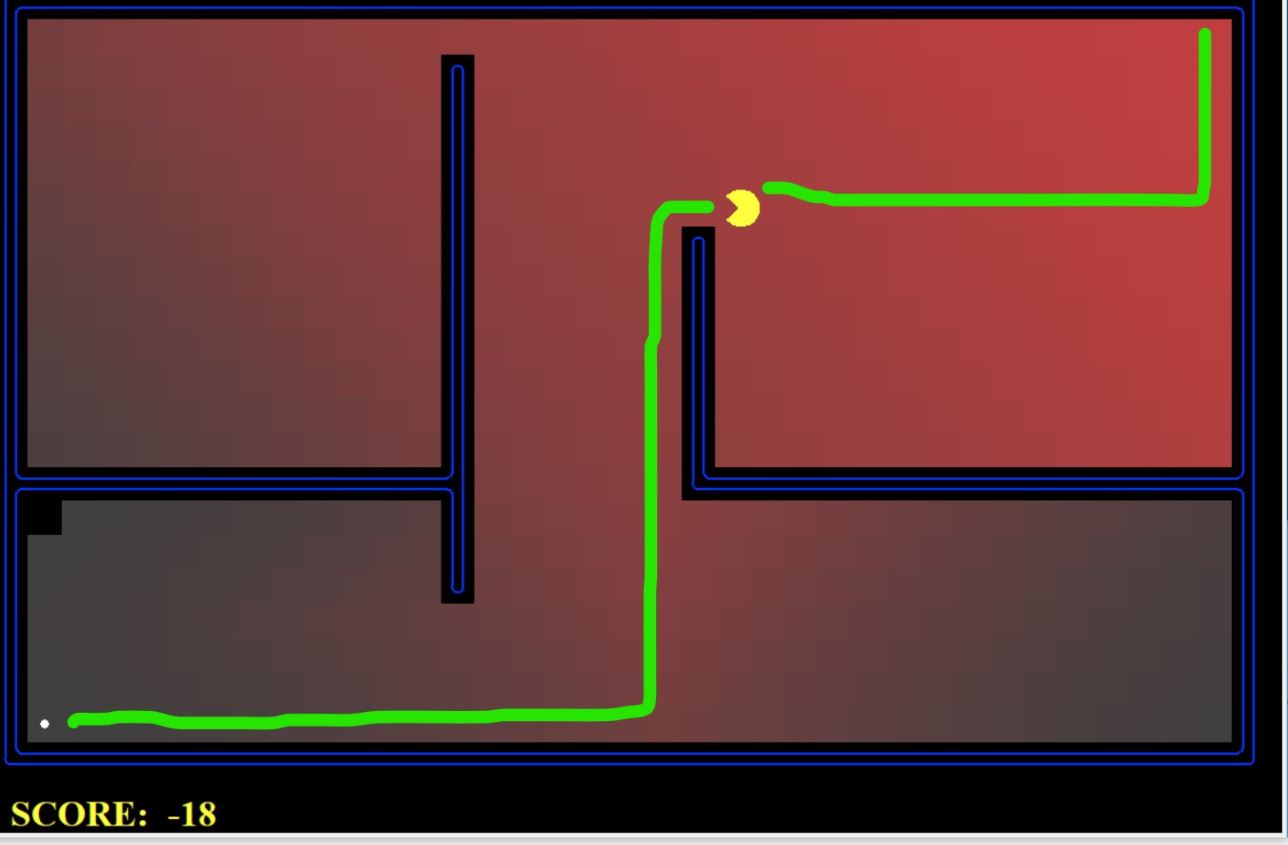
Figure 4.4: python pacman.py -l bigMaze -z 0.7 -p SearchAgent -a fn=ucs

A\* solution is slightly faster than uniform cost search (549 nodes in A\* vs 620 nodes in UCS), hence can be called optimal in comparison

*4.5 WHAT HAPPENS ON OPEN MAZE WITH VARIOUS SEARCH STATEGIES?*



For DFS search in an openMaze scenario, the pacman scans through each line horizontally indicating that it is always expanding the deepest node as in DFS search.



For BFS, A\*, UCS search strategies, the pacman always found out the shortest path to goal quite opposite to what happened with DFS search. In all the three cases (A\*,UCS,BFS) the search algorithm uses queue, which means

**5. PROBLEM 5: FINDING ALL CORNERS**

*5.1 TINY CORNERS:*

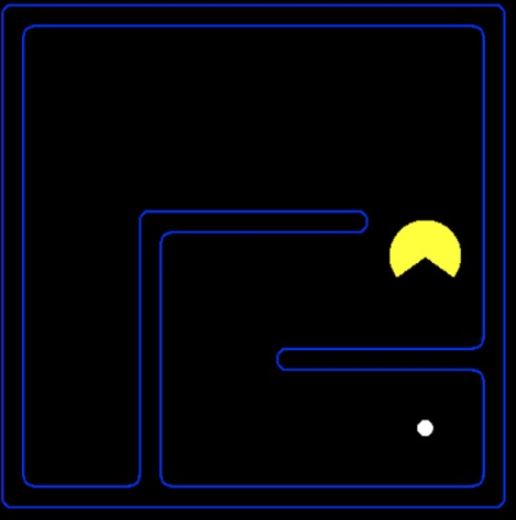


Figure 5.1: python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

*5.2 MEDIUM CORNERS:*

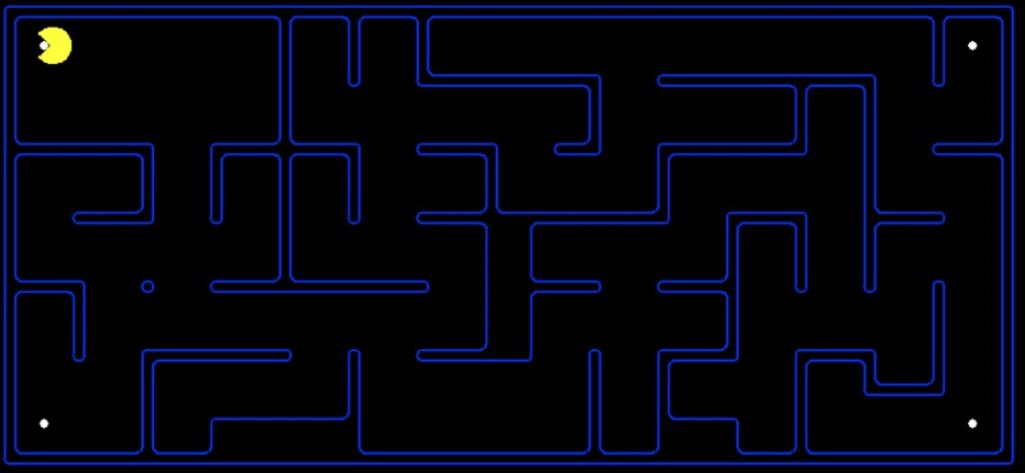


Figure 5.2: python pacman.py -l mediumCorners -p SearchAgent -a fn=bfs,prob=CornersProblem