Search Agent for the Pacman World

Navigating the Pacman world using AI Search Techniques

Yash Pradhan (YPP170130)
Department of Computer Science
The University of Texas at Dallas
Richardson, Dallas
Email: ypp170130@utdallas.edu

***Abstract*—The Pacman is an Arcade Game developed by Namco. In this Project, various search techniques studied in the CS 6364 Artificial Intelligence course are used to navigate the pacman in order to search for the Goal(food) in the Pacman world. This report will explain the techniques used and results obtained.**

***Keywords*—pacman; intelligent search; agent; heuristic search**

I. Introduction

The Pacman is an arcade game which was developed by Namco and released in Japan in May 1980. Upon its release it became a social phenomenon. The aim of the player is to navigate the Pacman through a maze containing food items, and four ghosts. The ghosts roam the maze and try to eat the Pacman. Score is assigned based on the number of food items eaten, time spent in navigation and other factors. The project that has been used was developed for UC Berkeley's course on Artificial Intelligence. The project doesn't focus on building intelligent agents for Atari games, but on teaching foundational AI concepts like state-space, graph search and heuristics.

II. Problem Description

"All those colored walls, Mazes give Pacman the blues, so teach him to search". The part of the pacman world, on which my project work focuses on is implementing search techniques to help pacman reach efficiently to the goal. The problem is that the pacman is at some initial location and the goal is to reach to a particular location(goal) and to collect food.

The pacman is yellow colored in the pacman world, which consists of mazes, there are ghosts which try to kill the pacman, the goal of the pacman is to collect the food and avoid the ghosts along the way. If the pacman touches a ghost, he loses a life. There are power pills in the world which gives special powers to pacman that he can eat the ghosts and earn bonus scores, the eyes of the ghosts return to the center and the ghost is reborn.

A perfect play occurs when the player achieves highest score on collecting every food in the maze and not being eaten by the ghost. My project focuses only on the First part i.e. implementing search techniques: uninformed and informed namely depth first search, breadth first search, uniform cost search and A star search using Manhattan distance as heuristics. This report explains the approach taken and results of the experiments

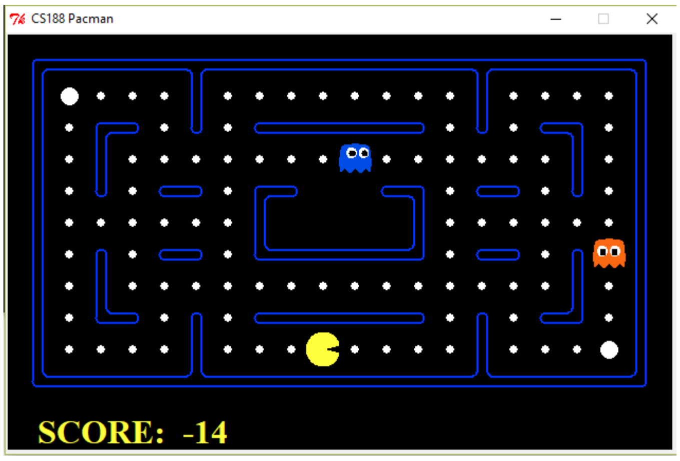

Fig. 1. The Pacman World, game play area.

III. Related Work

The entire scaffolding of the pacman project is provided by UC Berkeley the link to their website is http://ai.berkeley.edu/search.html. The project code is in python 2.7 and do not require any additional dependency.

This project has been incorporated in the coursework of Artificial Intelligence by many professors. I have implemented search algorithms learnt in our AI courses, mainly depth first serach, breadth first search, uniform cost search and A star search.

IV. Approach

This problem required the implementation of search techniques in order to enable pacman to reach to food location and collect it., hence helping the pacman to navigate the world of mazes.

The approach for solving the problem with different search techniques is almost the same, except the management of the fringe or openlist.

TABLE I. Data Structures

Sr.	Search Technique	Data Structure for frontier
1	Depth First Search	Stack
2	Breadth First Search	Queue
3	Uniform Cost Search	Priority Queue with priority as cost to the node
4	A star search	Priority Queue with priority as the sum of cost to node and estimate of code to the goal state.

The code written is generic i.e. can be used to solve other problems like 8 puzzle which are similar to the pacman problem, in terms of state space, successors, actions, cost and goal.

As the entire project is using python 2.7, the implementation of the search algorithms which resides in the search.py file is also using python 2.7.

The data structures used for the frontier are as shown in table 1, the closed list or visited list is implemented using the set data structure in python as it supports hasing hence the lookup for already visited states becomes efficient.

The representation for the node in the state space is using tuples, and each tuple for node stores its state configuration, cost to the node, path from the initial state and other information as required by each search strategy.

The functions to generate successors of the current node is given as implementation in the project itself, hence a call to getSuccessors( ) returns all the valid successors of the given state, hence avoiding the pacman to bang into walls.

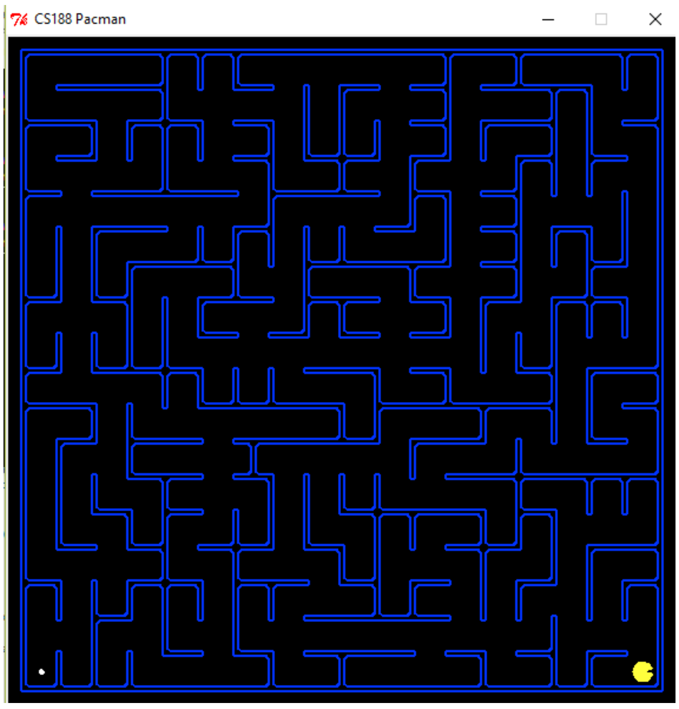

Fig. 2. The Pacman World, goal is to reach to the white dot on bottom left, navigating through the maze efficienlty.

The data structures required to implement the search techniques is given in the util.py file.

V. UNDERSTANDING

The fully implemented search agent is given in searchAgents.py, the agent will using search techniques that's implemented by me will plan a path through the pacman world, through the mazes to reach the goal, and then upon find the path to the goal, executes the path step by step.

In order for the search agent to plan out the path, search algorithms are required to be implemented, this shall reside in the search.py file.

The graph search version of searching is to be implemented inorder to make the search complete. The algorithms that are implemented are similar and differ in the details of how the fringe is constructed (data structure) and managed (queueing strategies).

When the python environment is set up, the pacman game can be played by an human agent by the command python pacman.py and the game screen appears as shown in figure 2.

VI. IMPLEMENTATION OF ALGORITHMS

Abbreviations:
DFS: Depth First Search
BFS: Breadth First Search
UCS: Uniform Cost Search
A* : A star Search

The following sections explains the working of these algorithms in order to help pacman navigate the world.

This has been done using graph search, as many repeated structures exists and the search tree becomes infinitely huge.

The node in the search graph consists of not only the current state i.e. location of the pacman but it also has other information like cost to the current node (path length), path to the current node etc.

Also it should be clear why the search strategies are required to navigate the world, because the entire exploration the world using the British Museum algorithms (the most naïve search algorithm) results into very large search space, which is not feasible to explore. Exponentially large number of states.

Example:
Consider small pacman world:
120 location for pacman
30 food items
12 ghost positions
4 Directions: North, South, East, West

World States = $120\text{x}(2^{30})\text{x}(12^{2})\text{x}4$

A. *Depth First Search*

The depth first search expands deepest node first, frontier is managed using a Last In First Out structure Stack. It is a blind search technique and not suitable when there are blind allays in the search space. For this problem DFS succeeds to find a path to the goal.

But this strategy is not good enough, consider the food item is to your north but the algorithm goes south instead and explores all the states in that path before finding the goal state.

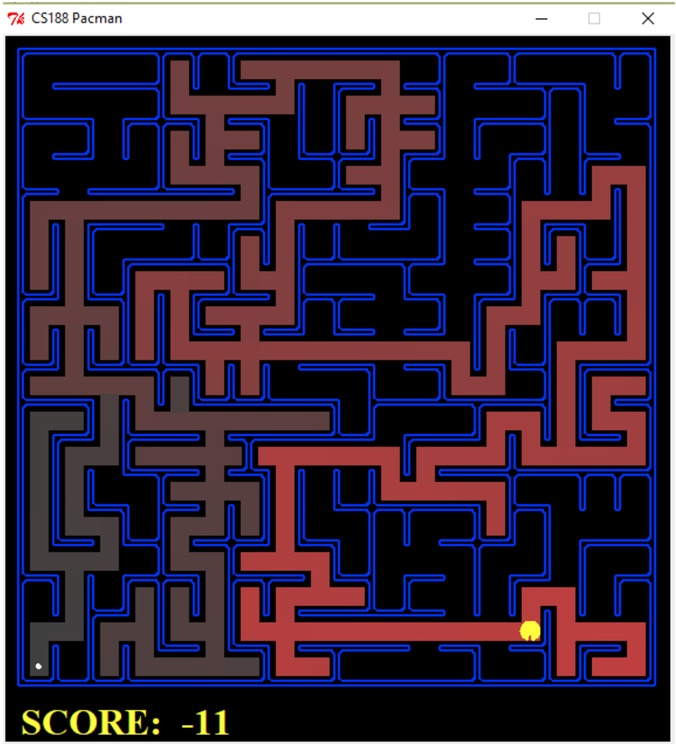

Fig. 3. The Pacman World, as explored by DFS.

As seen the deepest node is explored first. Brighter read means earlier exploration. DFS fails to find least cost solution.

B. *Breadth First Search*

The breadth first search expands shallowest node first, frontier is managed using a First In First Out structure Queue. It is a blind search technique and not suitable when the breadth of the search space is too wide. For this problem BFS succeeds to find a path to the goal, with least cost, but the work that it does to reach to the goal is high, i.e. is explores a large number of nodes.

The path that BFS outputs is a least cost path, the cost of each action North, South, East and West is same.

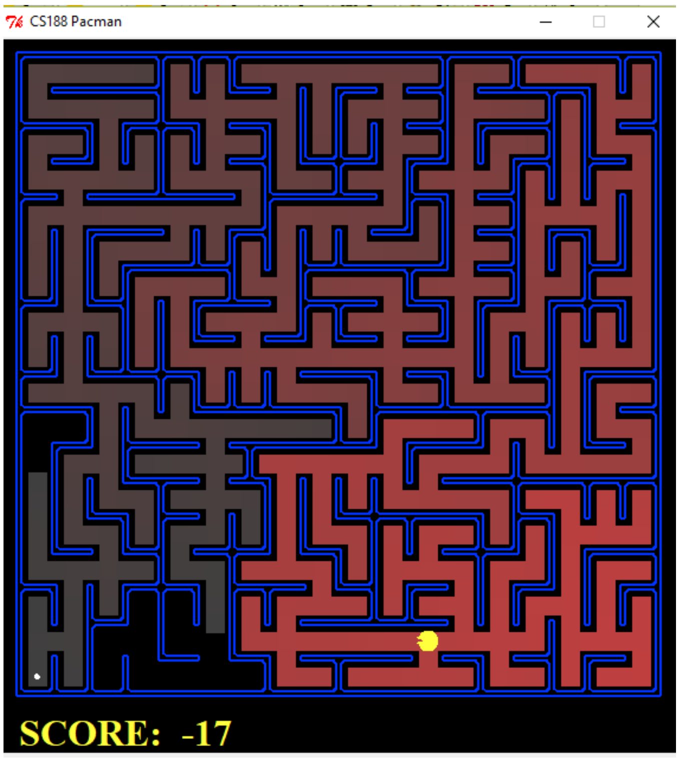

Fig. 4. The Pacman World, as explored by BFS.

It is clear from the figure, how BFS explored mostly all of the search space before reaching the goal.

C. *Uniform Cost Search*

The strategy employed by UCS is to expand the least cost unexpanded node, by using a priority queue ordered by g(n) where g(n) is the path cost to reach that particular node.

UCS examines all nodes at goals depth to see if some node has lower cost. Uniform cost search is optimal in sense that it returns the least cost path to the goal.

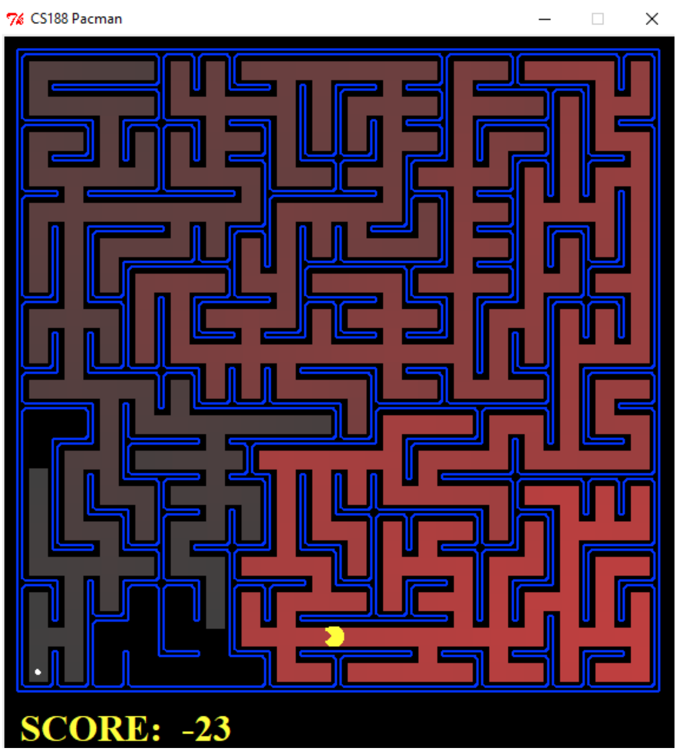

Fig. 5. The Pacman World, as explored by UCS.

In this instance the number of states explored by UCS is same as BFS, it returns the least cost path to goal.

Relaxation when a node which is already in open list is found but now with a lower(better cost).

```
RELAX_NODE:
IF CURRENT_NODE IN OPEN_LIST:
    IF COST(CURRENT_NODE) < NODE_IN_OPEN_LIST:
        DEL NODE_IN_OPEN_LIST
        ADD CURRENT_NODE TO OPEN_LIST
```

This is also used in A* search algorithm when it finds already explored node with better cost than what it had found earlier, and hence replaces with the node having less cost.

D. *A star Search*

The strategy used by A star search is an improvement over the one used by UCS. A* uses a priority queue and orders it by

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

g(n) is path from initial state to current state h(n) is heuristic estimate from current state to goal.

In order for A* to be optimal, the heuristic used should be consistent and admissible.

Heuristic used: Manhattan Distance

Points: (x1, y1), (x2, y2)

ManhattanDist = | x1 – x2 | + | y1 - y2 |

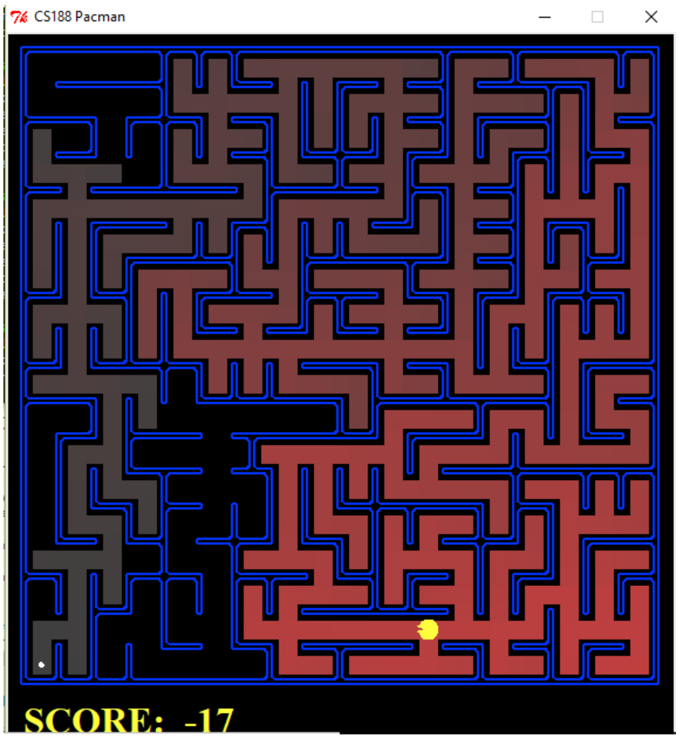

Fig. 6. The Pacman World, as explored by A*.

The performance of A* is better than other algorithms in terms of time and space complexities. As the heuristic used is consistent and admissible, the path returned is optimal.

TABLE II. COMPLEXITIES

Algo	Complete	Time	Space	Optimal
DFS	Yes (graph search)	$O(b^m)$	O(bm)	No
BFS	Yes	$O(b^d)$	$O(b^d)$	Yes
UCS	Yes	$O(b^{1+C*/e})$	$O(b^{1+C*/e})$	Yes
A*	Yes	$O(b*^d)$	$O(b*^d)$	Yes

b*: effective branching in A* search

C*: optimal cost in UCS, stepcost>=e

TABLE III. RESULTS

	DFS	BFS	UCS	A*
Medium Maze				
Nodes Explored	146	269	269	221
Path Length	130	68	68	68
Big Maze				
Nodes Explored	390	620	620	549
Path Length	210	210	210	210
Open Maze				
Nodes Explored	576	682	682	535
Path Length	298	54	54	54

VII. CONCLUSIONS

This project of CS 6364 Artificial Intelligence helped me to visualize the working of search techniques such as DFS, BFS, UCS and A* search, their performance is visualized in the results table. It helped me develop creative solutions in the challenging environment of pacman world and the idea that this can be extended to other Atari games is amazing.

Due to time limitations, though I could complete what was the scope of my project, I couldn't develop the multi agent (adversarial) search which would be more fun to visualize techniques like alpha beta pruning and expecti minimax algorithm.

REFERENCES

[1] The Pacman Project developed by John DeNero, Dan Klein, Pieter Abbeel et. al.

[2] Lecture notes from UC Berkeley for CS 188 Intro to AI for algorithms.

[3] Class Notes by Dr Moldovan