Assignment Presentation

Introduction to Artificial Intelligence

GROUP 12

Metrics

Student ID, Full name, Email, Assigned tasks, Complete percentage.

GROUP 12

FULL NAME	ID	EMAIL
VÕ VĂN LUÂN	2152744	luan.vo2003@hcmut.edu.vn
LÊ QUANG VIỆT	2153095	viet.leseirah145@hcmut.edu.vn
TÔ CHÂU HẢO NHÂN	2152829	nhan.tonhan0812@hcmut.edu.vn
NGUYỄN KIỀU BẢO KHÁNH	2152654	khanh.nguyen1701@hcmut.edu.vn
NGUYỄN TRẦN KHÔI	2152691	khoi.nguyen2701@hcmut.edu.vn

GROUP 12

FULL NAME	%	ASSIGNED TASKS
VÕ VĂN LUÂN	100%	8-Puzzle BFS, Advanced Task
LÊ QUANG VIỆT	100%	Pacman, Advanced Task
TÔ CHÂU HẢO NHÂN	100%	8-Puzzle A*, Advanced Task
NGUYỄN KIỀU BẢO KHÁNH	100%	Report, Advanced Task
NGUYỄN TRẦN KHÔI	100%	Pacman, Advanced Task

TABLE OF COMPLETE PERCENTAGES

TASK NAME	%
TASK 1 - BFS	100%
TASK 1 - A*	100%
TASK 2 - PACMAN - UCS	100%
TASK 2 - PACMAN - A*	100%
ADVANCED TASK	100%

Table of contents

01

8-PUZZLE

BFS, A* Algorithms

02

PACMAN

UCS, A* Algorithms

03

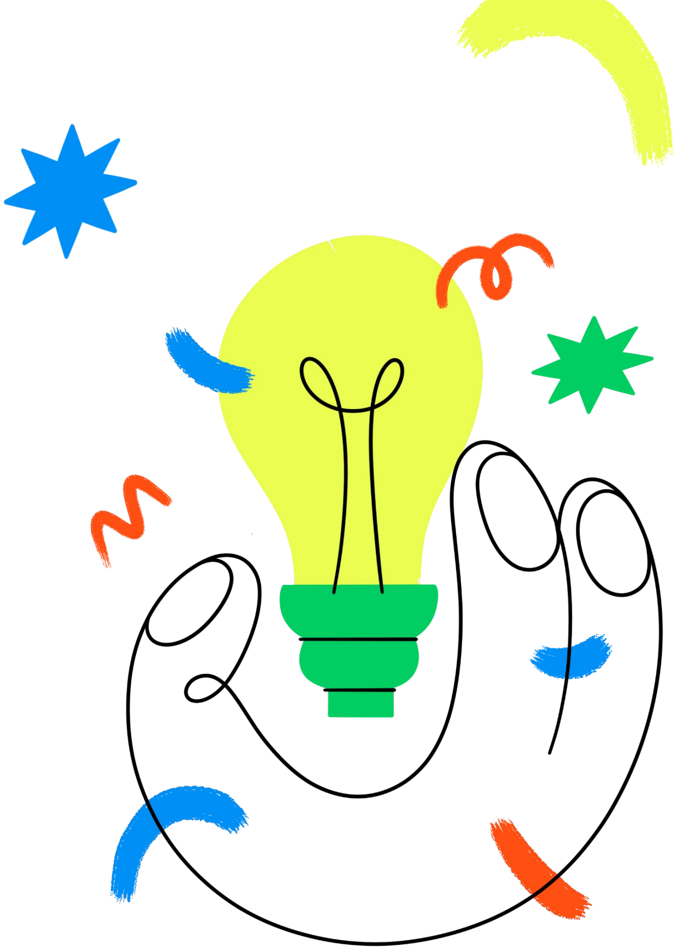
GRAPHICAL PACMAN

Run the game with GUI

01

8 - PUZZLE

THEORIES AND IMPLEMENTATION OF BFS, A*



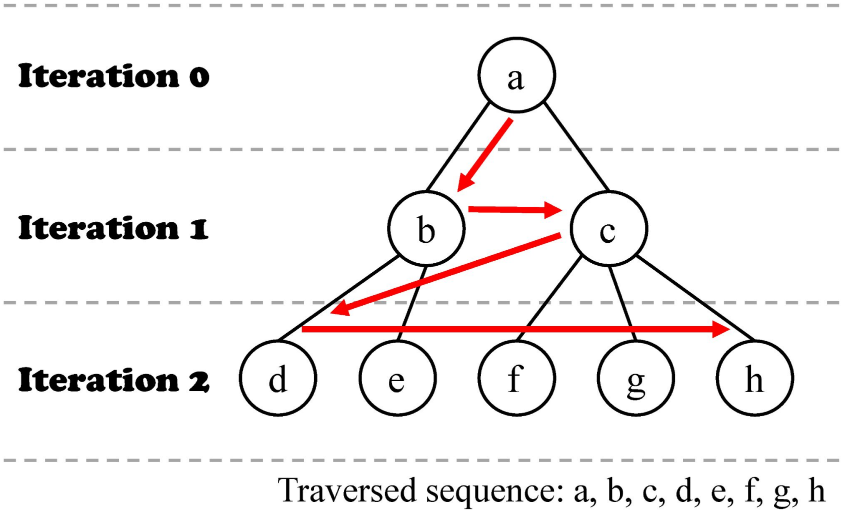
THEORY

BREADTH-FIRST SEARCH (BFS)

Breadth First Search (BFS) is a graph traversal algorithm that explores all the vertices in a graph at the current depth before moving on to the vertices at the next depth level.

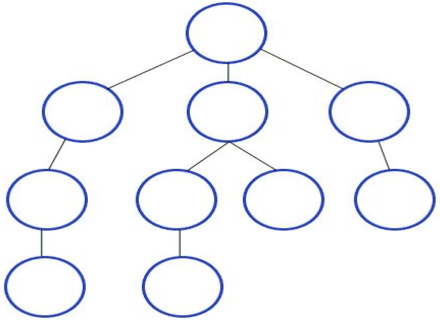
- The root node is expanded first → all the successors of the root node → their successors → ...
- Implementation: *frontier* is a **FIFO** queue

In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.



BREADTH-FIRST SEARCH (BFS)

```
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
        /* chooses the shallowest node in frontier */
        node ← POP(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)
```

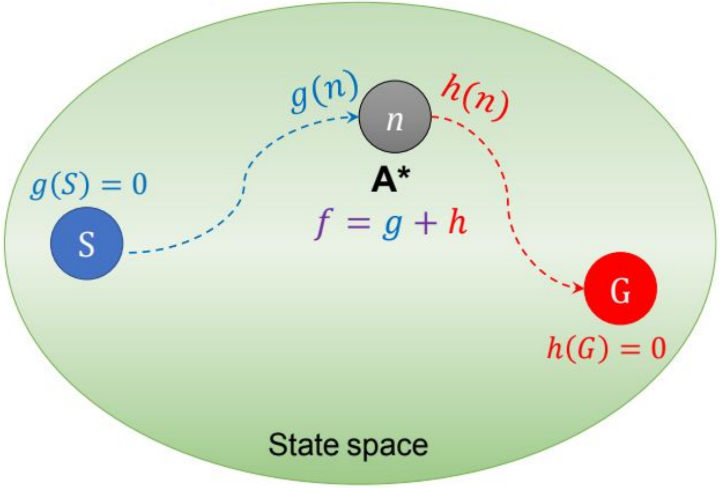


THEORY

A* SEARCH

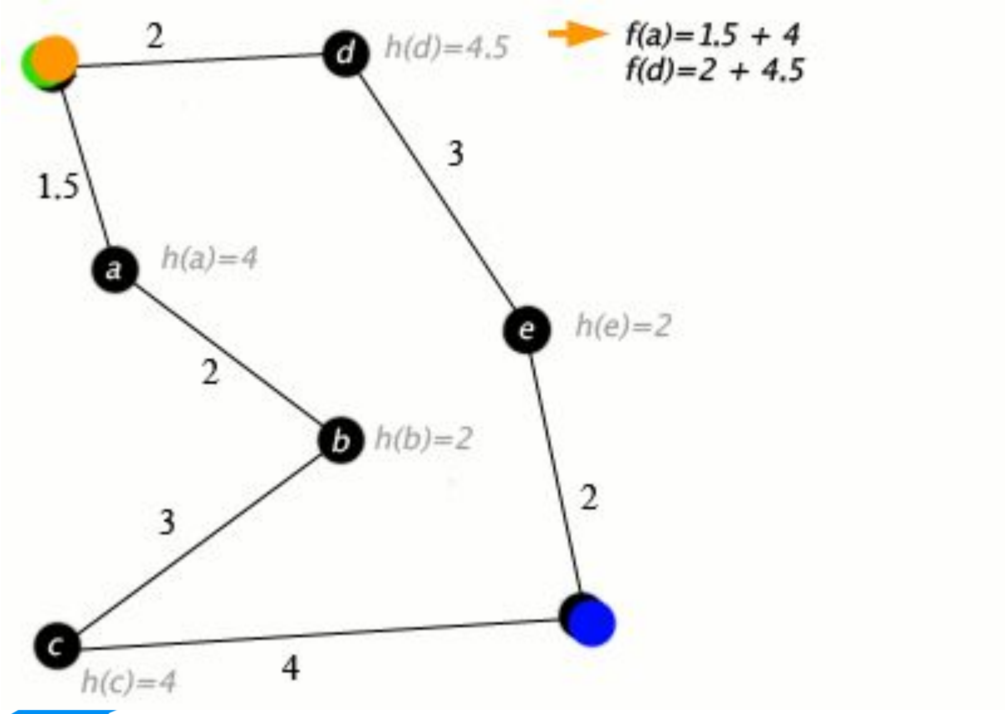
The most widely known form of best-first search.

- Use heuristic to guide search, but not only
- Avoid expanding paths that are already expensive
- Ensure to compute a path with minimum cost



Evaluate nodes by $f(n) = g(n) + h(n)$

A* SEARCH ALGORITHM



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

Evaluate:

f(n) = estimated cost of the cheapest solution through n.

g(n) = the cost to reach the node n

h(n) = the cost to get from n to the goal

8-PUZZLE IMPLEMENTATION

States

The location of each of tiles and the blank.

Initial State

Any state can be the initial state.

Actions

Left, Right, Up, or Down. Different subsets depending on where the blank is.

Transition Model

Return a resulting state given a state and an action.

Goal Test

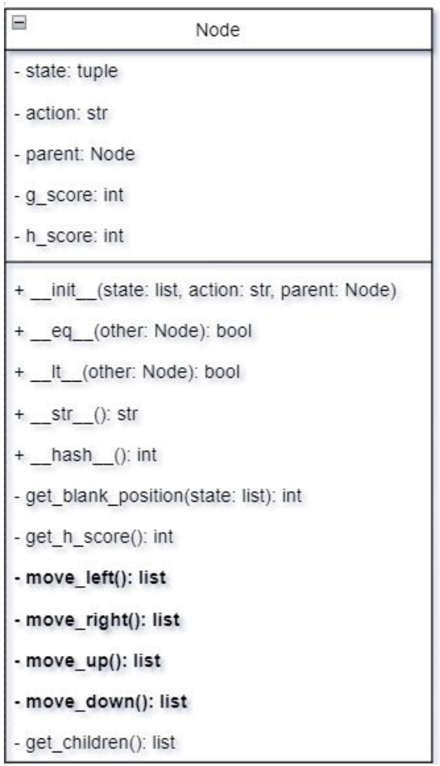
Goal Config:

```
Node([0,1,2,3,
4,5,6,7,8])
```

Path Cost

Each step costs 1.

8-PUZZLE IMPLEMENTATION

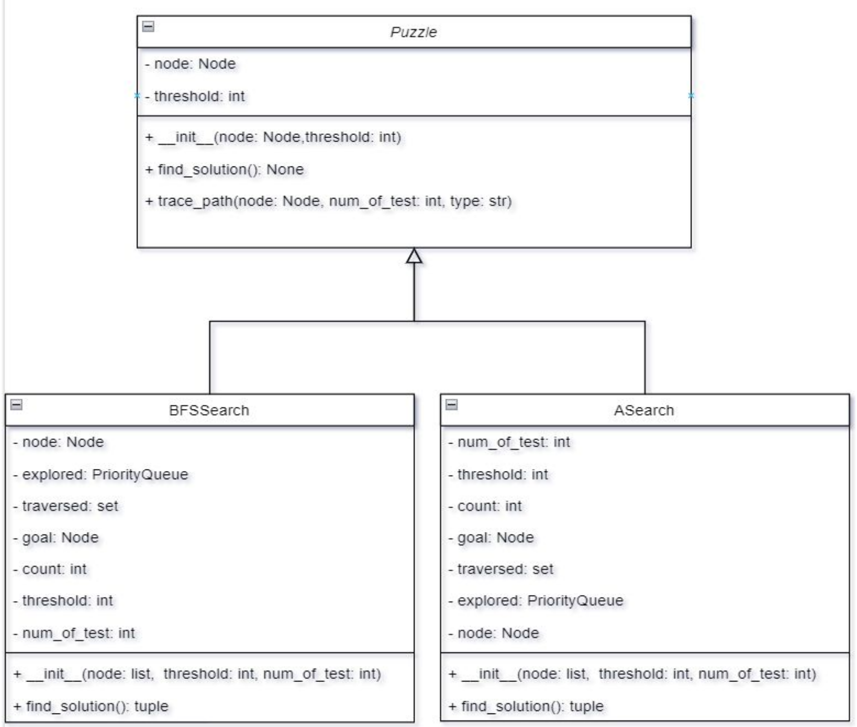


OOP STRUCTURES - NODE

Structure of a node.

- State: Tuple of Current State
- Action: Up, Down, Left, Right
- Parent Node
- G-Score: g(n)
- H-Score: h(n)

8-PUZZLE IMPLEMENTATION

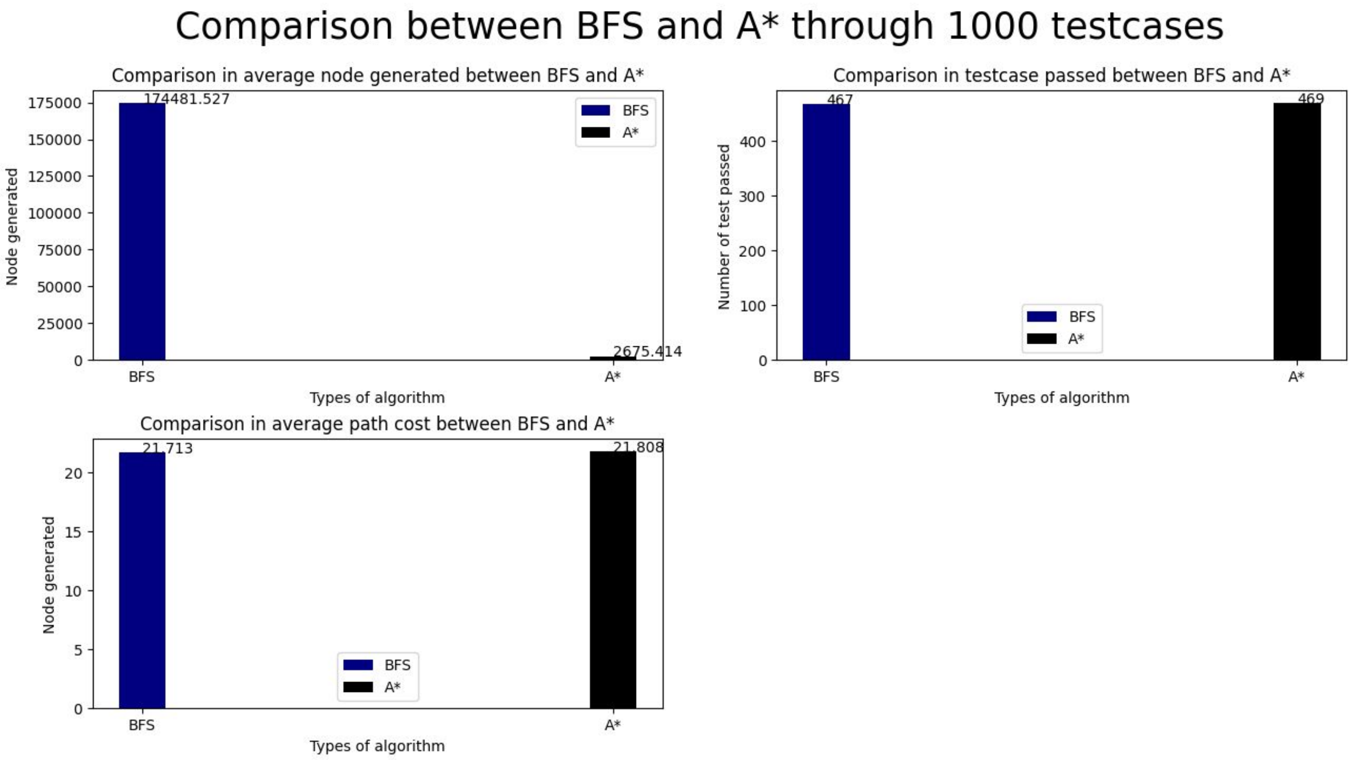


OOP STRUCTURES - PUZZLE

BFSSearch: Implement BFS Search algorithm for 8-Puzzle

ASearch: Implement A* Search algorithm for 8-Puzzle

8-PUZZLE IMPLEMENTATION



Pros and cons - Solving with BFS

Pros

- BFS is guaranteed to find a solution if one exists.
- BFS guarantees an optimal solution in terms of the number of moves required to reach the goal state.
- The implementation of BFS is relatively straightforward compared to more complex search algorithms like A* or IDA* (Iterative Deepening A*).

Cons

- BFS may require a significant amount of memory, especially for problems with large state spaces.
- BFS explores all possible states at each depth level before moving to the next level. → High time complexity
- BFS becomes increasingly inefficient as the size of the state space grows.

Pros and cons - Solving with A*

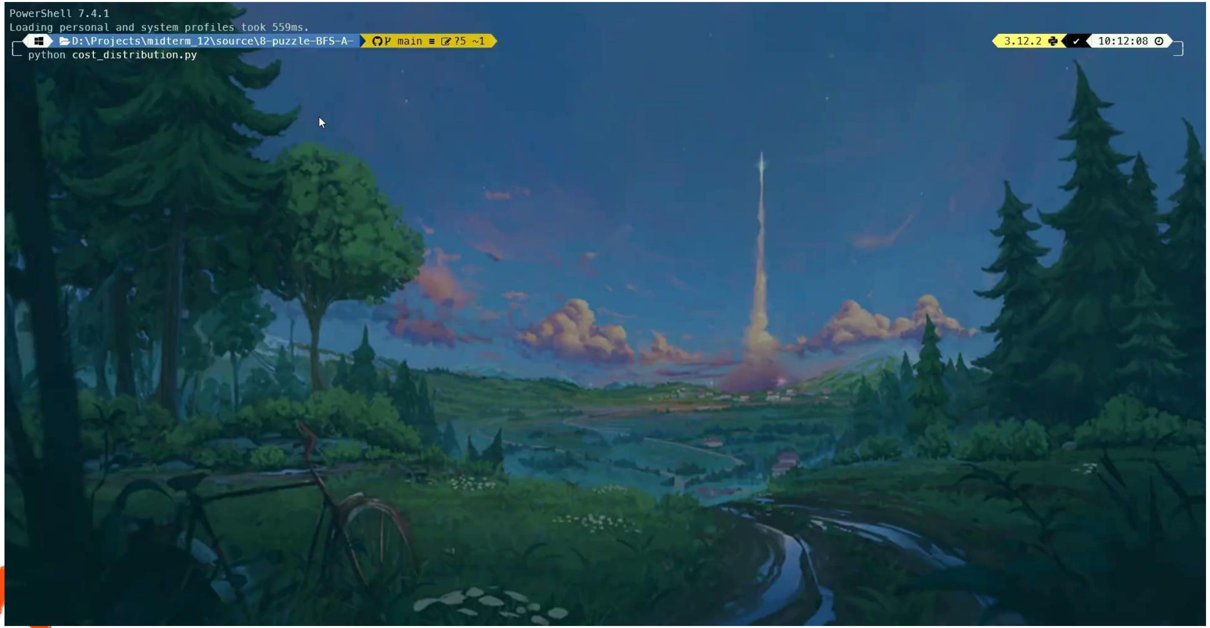
Pros

- A* is complete, meaning it will always find a solution if one exists.
- A* guarantees an optimal solution in terms of the number of moves required to reach the goal state.
- A* is generally more efficient than uninformed search algorithms like BFS because it uses heuristics to guide the search towards the goal.

Cons

- The effectiveness of A* heavily relies on the quality of the heuristic function used.
- A* may require significant memory resources, especially for problems with large state spaces.
- While A* is generally more efficient than uninformed search algorithms, it can still be computationally expensive, especially for problems with large state spaces and complex heuristics.

Demo


PowerShell 7.4.1
Loading personal and system profiles took 559ms.
D:\Projects\midterm_12\source\8-puzzle-BFS-A-
main ≡ ?5 ~1
3.12.2
10:12:08
python cost_distribution.py

02

PACMAN

THEORIES AND IMPLEMENTATION OF UCS, A*

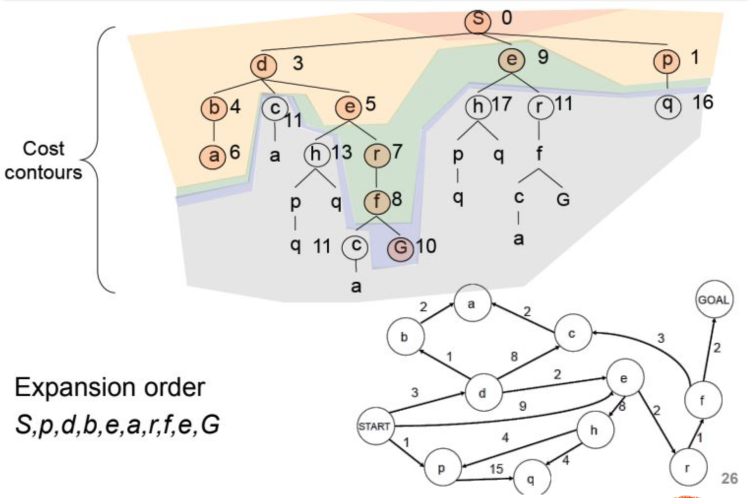
THEORY

UNIFORM-COST SEARCH (UCS)

The Uniform Cost Search Algorithm is a search algorithm to find the minimum cumulative cost of the path from the source node to the destination node. It is an uninformed algorithm.

The goal test is applied to a node when it is selected for expansion.

A test is added in case a better path is found to a node currently on the frontier.



UNIFORM-COST SEARCH (UCS)

```
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 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
```

PACMAN IMPLEMENTATION

States

Pacman's position and the locations of food points.

Initial State

Pacman is positioned at a specified location and food points are scattered throughout the maze.

Actions

North, South, East, or West

Transition Model

Return a resulting state given a state and an action.

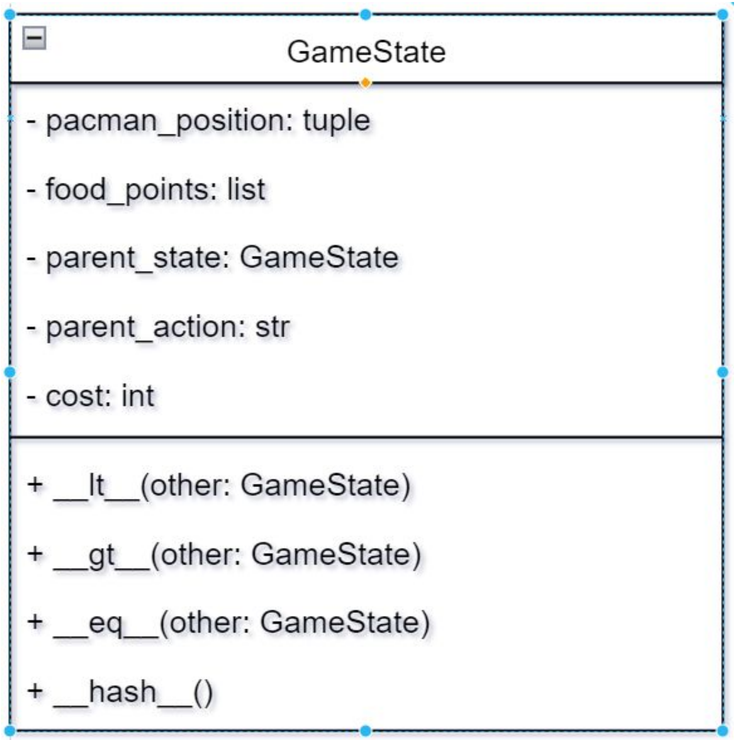
Goal Test

Successful completion of the game.

Path Cost

Each step costs 1.

PACMAN IMPLEMENTATION

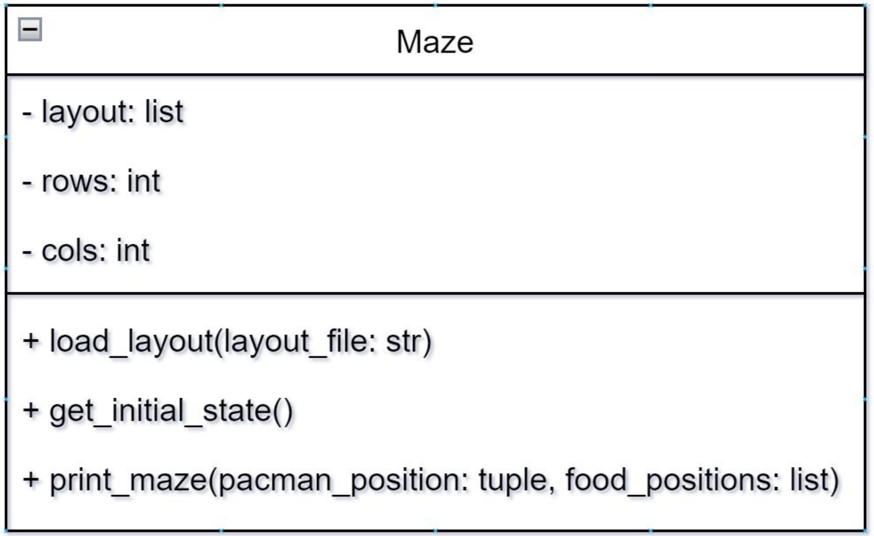


OOP STRUCTURES - Game State

The **GameState** class represents the state of the game in the context of the Pacman game.

The GameState class also provides methods for comparing states based on their costs and positions, as well as a __hash__ method for hashing states.

PACMAN IMPLEMENTATION

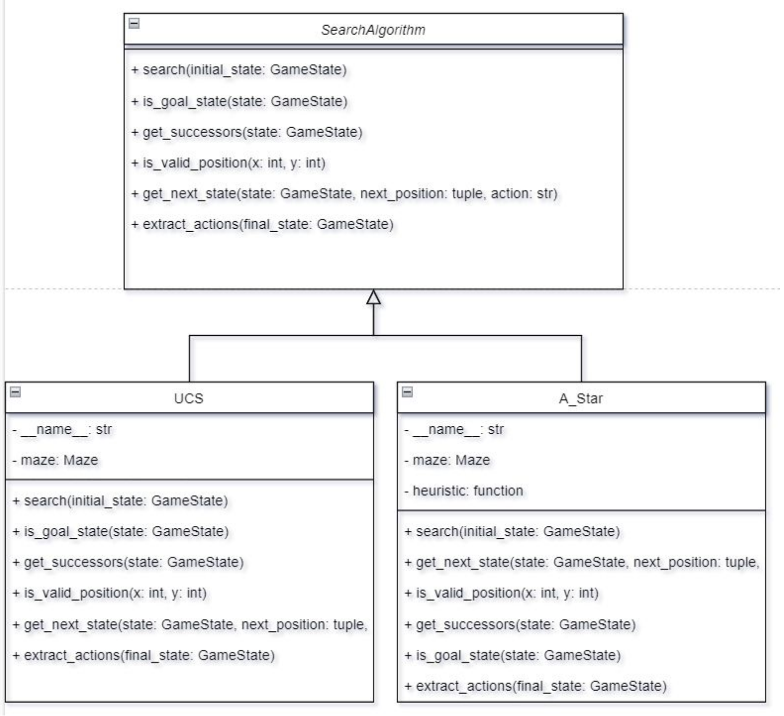


OOP STRUCTURES - Maze

The **Maze** class represents the layout of the game grid in the Pacman game.

1. Load the maze layout from a file.
2. Obtain the initial game state, (starting position and the positions of food points).
3. Print the current state of the maze.

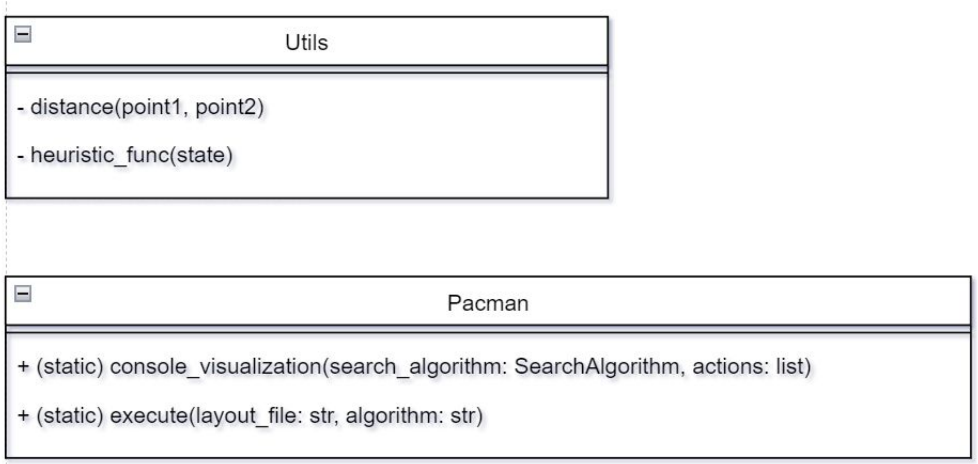
PACMAN IMPLEMENTATION



OOP STRUCTURES - Search Algorithms

SearchAlgorithm is an abstract base class defining the interface for search algorithms.
UCS, A_Star is concrete subclasses of SearchAlgorithm implementing the algorithms.

PACMAN IMPLEMENTATION

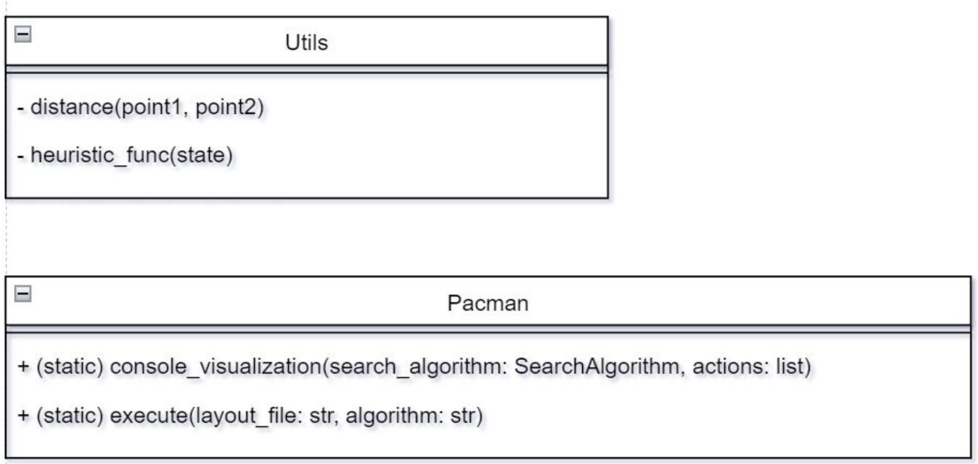


OOP STRUCTURES - Utils

The **Utils** class serves as a utility toolbox in the Pacman game.

- *distance(point1, point2)*: which calculates the Manhattan distance between two points.
- *heuristic_func(state)*: computes a heuristic value for a given game state.

PACMAN IMPLEMENTATION



OOP STRUCTURES - Pacman

- <u>*console_visualization(search_algorithm, actions):*</u>
  method for displaying the game's progression in the console.
- <u>*execute(layout_file, algorithm):*</u>
  static method initiates the game, loading the maze layout from a file and executing the specified search algorithm.

Pros and cons - Solving with UCS

Pros

- Ensuring that it finds the shortest path from the initial state to the goal state.
- UCS is complete, meaning it will find a solution if one exists, even in complex environments.
- UCS is straightforward to implement and understand.

Cons

- UCS can be computationally expensive, especially in large search spaces, as it explores all possible paths up to a certain cost.
- **Space Complexity:** UCS requires storing all explored nodes.
- UCS treats all actions equally, which may not be optimal in scenarios where certain actions have different costs or risks.

Pros and cons - Solving with A*

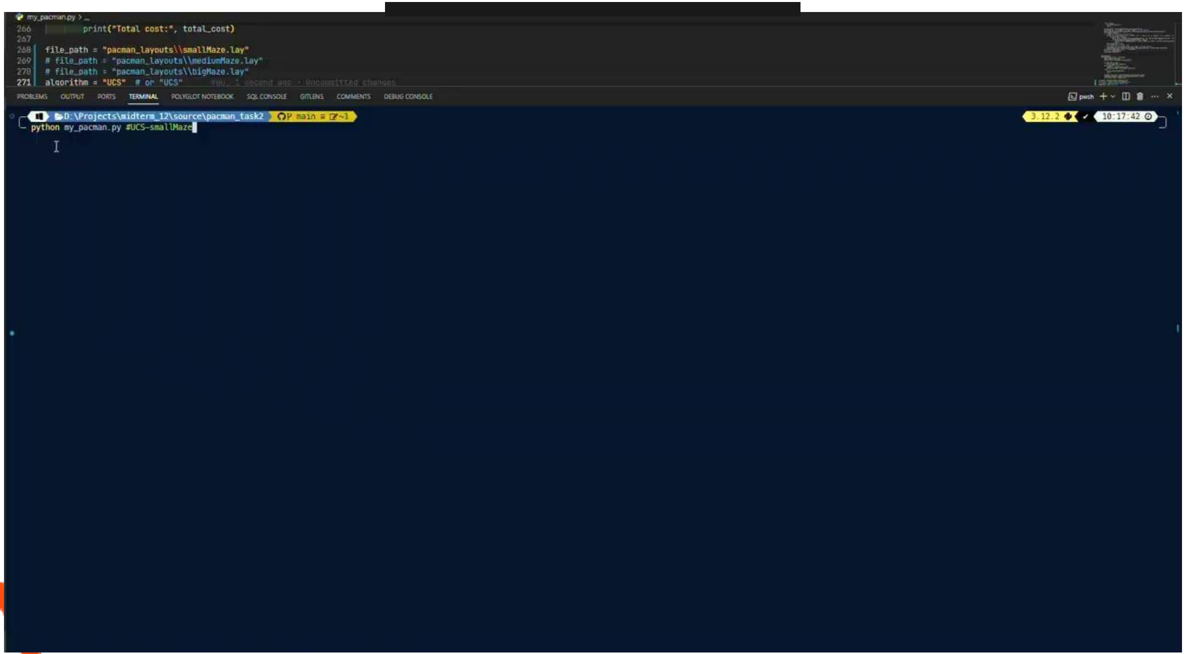
Pros

- A* guarantees optimality when using an admissible heuristic.
- A* is often more efficient than uninformed search algorithms like UCS, as it uses heuristic information to guide the search towards the most promising states
- A* allows for the use of different heuristic functions

Cons

- A* relies on the quality of the heuristic function, which must be admissible
- A* may require significant memory resources, especially in large search spaces
- In certain cases, particularly when using a poor heuristic or in environments with complex state spaces

Demo

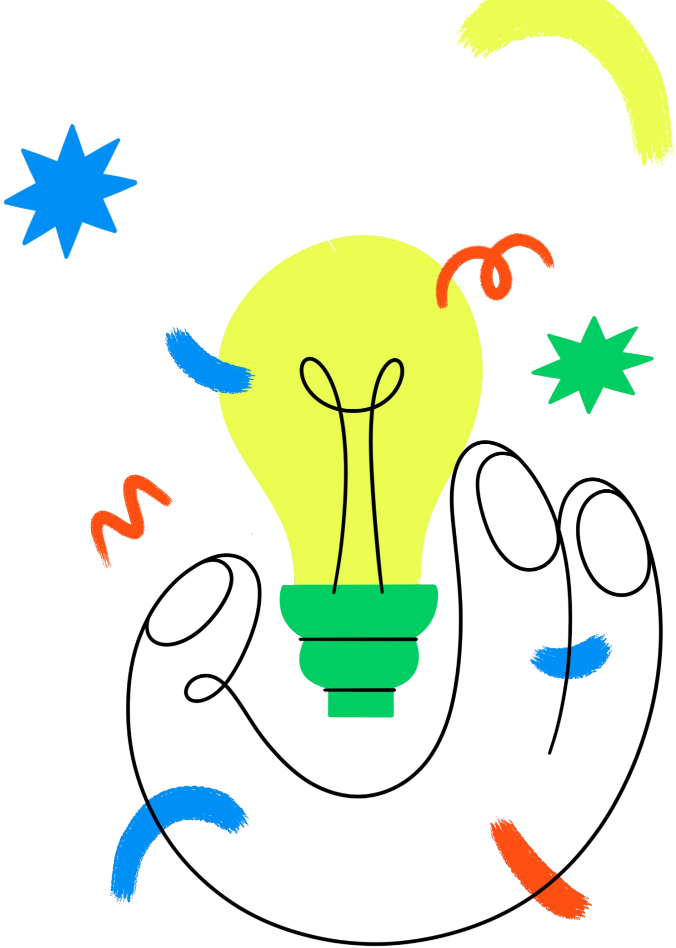


my_pacman.py
print("Total cost:", total_cost)
file_path = "pacman_layouts\\smallMaze.lay"
# file_path = "pacman_layouts\\mediumMaze.lay"
# file_path = "pacman_layouts\\bigMaze.lay"
algorithm = "UCS" # or "UCS"
PROBLEMS OUTPUT PORTS TERMINAL POLYGLOT NOTEBOOK SQL CONSOLE GITLENS COMMENTS DEBUG CONSOLE
D:\Projects\midterm_12\source\pacman_task2
main
python my_pacman.py #UCS-smallMaze
3.12.2
10:17:42

03

GRAPHICAL PACMAN

AWESOME.



USING UNIFORM COST SEARCH

USING A STAR SEARCH

Resources and References

- R. Belwariar, “A* Search Algorithm - GeeksforGeeks,” *GeeksforGeeks*, Sep. 07, 2018. https://www.geeksforgeeks.org/a-search-algorithm
- GeeksforGeeks, “Breadth First Search or BFS for a Graph - GeeksforGeeks,” *GeeksforGeeks*, Feb. 04, 2019. https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/
- “Uniform-Cost Search (Dijkstra for large Graphs),” GeeksforGeeks, Mar. 25, 2019. https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/
- GeeksforGeeks, “Python OOPs Concepts,” *GeeksforGeeks*, May 16, 2016. https://www.geeksforgeeks.org/python-oops-concepts/
- Mre, “English: An animation to visualize the Breadth-First-Search (BFS) Algorithm used in Computer Science.,” Wikimedia Commons, Mar. 26, 2009. https://commons.wikimedia.org/wiki/File:Breadth-First-Search-Algorithm.gif
- “Category:Animations of pathfinding algorithms - Wikimedia Commons,” commons.wikimedia.org. https://commons.wikimedia.org/wiki/Category:Animations_of_pathfinding_algorithms (accessed Mar. 28, 2024).
- C. Birch, “GameInternals - Understanding Pac-Man Ghost Behavior,” *gameinternals.com*, Dec. 02, 2010. https://gameinternals.com/understanding-pac-man-ghost-behavior

Thanks!