Pac-man

Introduction

In this project, we explore search algorithms in the context of the classic Pac-man game. Pac-man lives in a maze world full of corridors and dots to eat, and our task is to guide him efficiently using different search techniques. This project demonstrates how various search algorithms, such as Depth First Search (DFS), Breadth First Search (BFS), Uniform Cost Search (UCS), and A* Search, work in solving pathfinding problems.

Table of Contents

1. Instructions
2. Search Algorithms
3. Commands
4. Observations

Instructions

1. Clone or download the Pac-man package.
2. Ensure you have Python 3.7 or higher installed on your system.
3. To start the game or test various features, run the provided commands from your terminal.

Search Algorithms (DFS, BFS, UCS, A*)

The Pac-man agent finds paths through the maze world to reach specific locations or collect food using the following algorithms:
- Depth First Search (DFS): Explores deeply without guaranteeing the shortest path.
- Breadth First Search (BFS): Explores level by level and guarantees the shortest path.
- Uniform Cost Search (UCS): Considers path costs and explores the lowest-cost path first.
- A* Search: Combines UCS and heuristics to find the most efficient path.

Code Structure:

- `search.py`: Contains all search algorithms.
- `searchAgents.py`: Contains search-based agents.
- `pacman.py`: Main file to execute Pac-man with different layouts, agents, and search techniques.

Commands

General Commands:

1. Start the game with the default agent:
   python pacman.py

2. Run the GoWestAgent:

python pacman.py --layout testMaze --pacman GoWestAgent

3. Run the GoWestAgent on a more complex maze:
python pacman.py --layout tinyMaze --pacman GoWestAgent

4. Display help for all available options:
python pacman.py -h

Testing Search Algorithms:

1. Solve `tinyMaze` with a specific function:
python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch

2. Solve `mediumMaze` using the default search agent:
python pacman.py -l mediumMaze -p SearchAgent

3. Solve `bigMaze` with a custom zoom level:
python pacman.py -l bigMaze -z .5 -p SearchAgent

Breadth First Search (BFS):

1. Use BFS on `mediumMaze`:
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs

2. Use BFS on `bigMaze` with zoom level adjustment:
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5

Uniform Cost Search (UCS):

1. Use UCS on `mediumMaze`:
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs

Custom Agents:

1. Use the StayEastSearchAgent:
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

2. Use the StayWestSearchAgent:
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent

A* Search:

1. Run A* with the Manhattan heuristic:
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic

Corners Problem:

1. Solve `tinyCorners` using BFS:
python pacman.py -l tinyCorners -p SearchAgent -a fn=bfs,prob=CornersProblem

2. Solve `mediumCorners` using A*:
   python pacman.py -l mediumCorners -p AStarCornersAgent -z 0.5

Food Search Problem:

1. Solve `testSearch` with A*:
   python pacman.py -l testSearch -p AStarFoodSearchAgent

2. Solve `trickySearch` with A*:
   python pacman.py -l trickySearch -p AStarFoodSearchAgent

Closest Dot Search Agent:

1. Solve `bigSearch` with the ClosestDotSearchAgent:
   python pacman.py -l bigSearch -p ClosestDotSearchAgent -z .5

Observations

Depth First Search (DFS):

Explores deeply without guaranteeing the shortest path.
Example:
   python pacman.py -l tinyMaze -p SearchAgent