Maze Solver Project

Overview

This project implements a maze-solving algorithm using Policy Iteration and Value Iteration. The goal is to navigate through a maze efficiently by leveraging dynamic programming and reinforcement learning techniques.

Getting Started

Prerequisites

Make sure you have the following installed on your machine:

- Visual Studio

Running the Project

- **Method 1:**
  Run Menu.py file
- **Method 2:**
  If you want policy iteration:
  Run mode1.py and change the input from the code as follows

```
# Define maze parameters
N=5
gamma=0.99
# Define the maze as a 2D grid with rewards and obstacles
grid = [['100', '.', '.', '.', '10'],
        ['.', 'B', '.', '.', '.'],
        ['.', '.', '.', '.', '.'],
        ['.', '.', '.', 'B', '.'],
        ['.', '.', '50', '.', 'B']]

# Call the solve function to find optimal actions for each cell in the maze
policy_iteration(grid,gamma, N)
```

for tracing you can uncomment this printing statements in policy_iteration.py

in line 137:

```
#  PRINT New value functions of each state at each iteration
     print(f'\nNew value functions of each state at iteration {iter}:')
     for i, row in enumerate(new_value_fns):
         print(f'V({i}): {row}')
```

in line 158:

```
# PRINT Action value functions of each state at each iteration
                 print(f'\nAction value functions of each state at iteration {iter}:')
```

in line 178:

```
# PRINT
                    print(f'Q({s},{x}): {action_value_fns[v]}')
```

in line 190:

```
 # PRINT best action of each state at each iteration
                print(f'Best Action at iteration {iter}: Best_A({s}):
{new_policies[s]}')
```

in line 221:

```
 # PRINT
        print(f'Iteration {i} - Value Function Difference:
{value_function_difference}')
```

If you want value iteration:
Run mode2.py and change the input from the code as follows

```
# Define maze parameters
N=5
gamma=0.99
# Define the maze as a 2D grid with rewards and obstacles
grid = [['100', '.', '.', '.', '10'],
        ['.', 'B', '.', '.', '.'],
        ['.', '.', '.', '.', '.'],
        ['.', '.', '.', 'B', '.'],
        ['.', '.', '50', '.', 'B']]
# Call the solve function to find optimal actions for each cell in the maze
solve(N,gamma,grid)
```