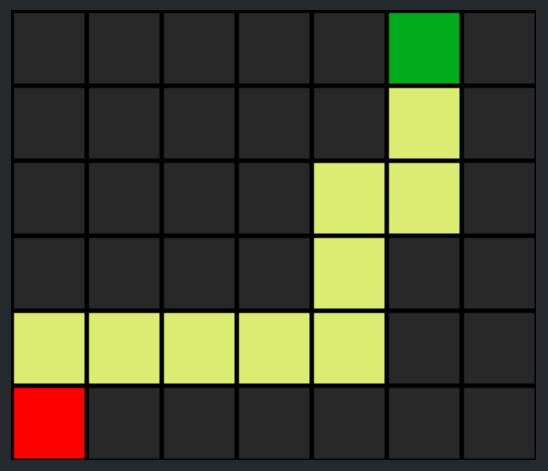
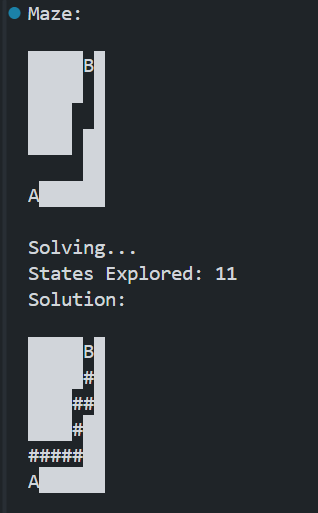
**Explanation of python maze solving code:**

This program implements a maze-solving algorithm using depth-first search (DFS). It starts by loading a maze from a text file and initializes the maze structure, walls, start point ('A'), and goal point ('B'). The solve function employs DFS, where it explores paths by pushing states onto a stack (using the StackFrontier class) and marking visited nodes to prevent revisiting. It continues exploring paths until the goal is found or all possibilities are exhausted. Once the goal is reached, it reconstructs the path and marks it in the maze. The solution is then displayed both in text form and visually using the output\_image method, which colors the maze to represent walls, the start, the goal, explored paths, and the final solution.

The Maze class also contains methods to find neighboring cells and print the maze. The program outputs a visual representation of the solved maze using the Python Imaging Library (PIL), where walls, paths, and the solution are color-coded. This combination of depth-first search for problem-solving and visual output gives a clear and comprehensive way to understand the maze-solving process, making the algorithm's steps easy to follow both in text and visual formats.  
  
**Example 1:**

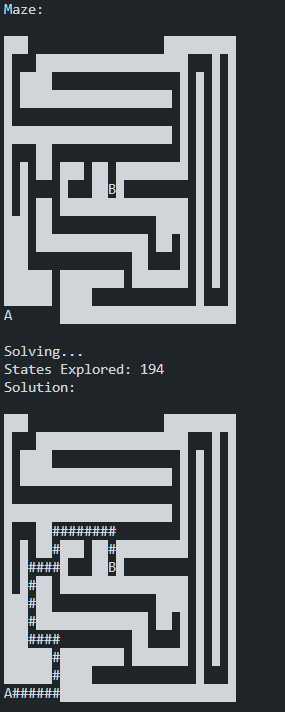
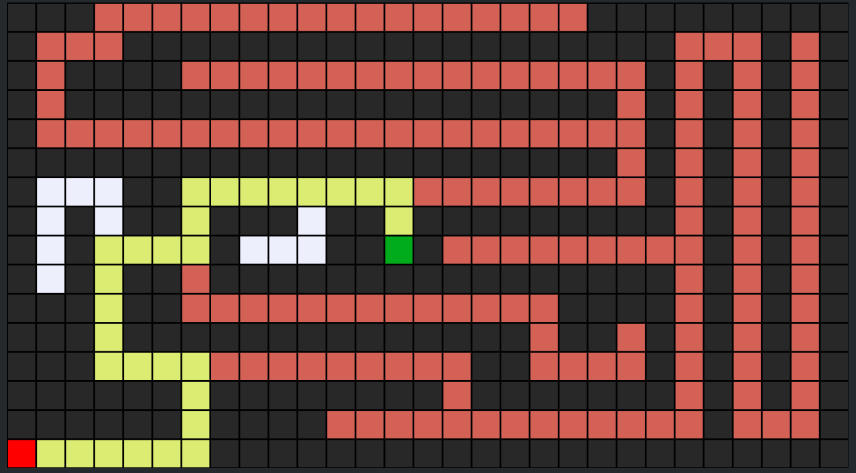


In the first example, the maze is represented by a 6x7 grid where the start ('A') is in the bottom-left corner and the goal ('B') is in the top-right corner. The walls of the maze are depicted by solid block symbols (█), and open spaces are indicated by empty spaces. The maze layout is relatively simple but includes several obstacles that block the direct path from 'A' to 'B'.

The solving process, which explores 11 states, successfully finds a path from the start to the goal. The solution is marked by '#' symbols that show the path taken by the depth-first search algorithm. The path moves upward and to the right. It also provides a colorful representation of the solved maze, which is created as an image after solving the maze.

In the solution:

* The path starts at 'A' and moves rightward, then up through the empty spaces of the maze, avoiding obstacles.
* After moving through a series of explored cells, the path successfully reaches 'B' at the top.

**Example 2:**

In this example, the maze is much more complex, showcasing the maze-solving algorithm's ability to handle more challenging scenarios. The maze is represented by a larger 16x31 grid, with the start ('A') in the bottom-left corner and the goal ('B') positioned near the center-right of the maze. The layout consists of numerous walls and corridors, making the path to the goal more difficult.

The solving process explored 194 states before finding the solution, which is a significant increase compared to simpler mazes. This reflects the complexity of the maze and the thorough search process conducted by the depth-first search algorithm. The solution path is marked by the '#' symbols, showing the algorithm's winding route through the maze’s narrow corridors and avoiding dead ends.

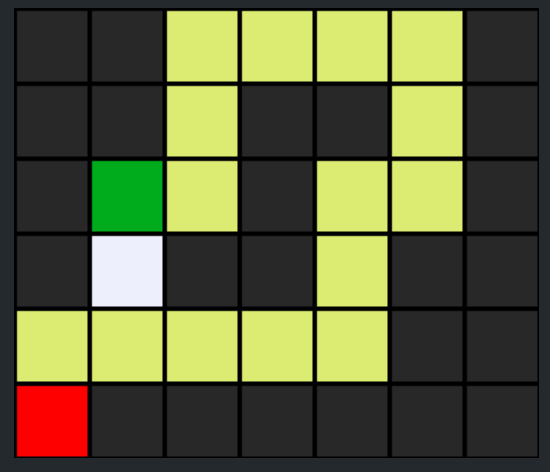
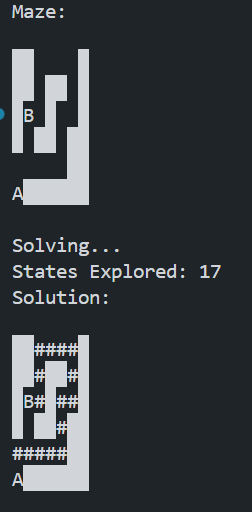
**Key features of the maze:**

* The path starts at 'A' and follows a complex route filled with obstacles before finally reaching 'B.'
* The '#' symbols illustrate the explored and correct path the algorithm followed to solve the maze.

Additionally, the algorithm also creates a visual output of the maze solution in the form of an image. In this image:

* The white areas represent open paths.
* The red squares show walls and obstacles.
* The green square represents the goal 'B'.
* The yellow path indicates the solution discovered by the algorithm, clearly showing how it navigates through the maze from start to goal.

This example effectively shows how the maze-solving algorithm can efficiently navigate through more complex structures, exploring a large number of possibilities before finding the correct path. The visual image generated helps users easily understand the solution route and how the algorithm overcomes the maze's complexity.

**Example 3:**

This output presents a medium-difficulty maze where the depth-first search algorithm finds the path from the starting point 'A' to the goal 'B'. The maze is represented in a 6x7 grid, with several walls and obstacles that make the path slightly more challenging than simpler mazes, but still manageable.

In the text representation:

* The start ('A') is located at the bottom-left corner.
* The goal ('B') is near the top-left.
* The solid block symbols (█) represent walls that the path must navigate around, while the empty spaces are open areas.

The algorithm explored 17 states before finding the solution, indicating a moderate level of complexity. The path, represented by '#' symbols, weaves through the maze from 'A' to 'B' while avoiding walls.

In the visual image of the solution:

* The starting point 'A' is represented by a red square in the bottom-left.
* The goal 'B' is marked by a green square near the top-left.
* Black squares represent walls and obstacles.
* The yellow line represents the solution path, which moves rightward, then upward, and finally leftward to reach the goal.

The algorithm efficiently navigates around the walls, demonstrating its capability to solve mazes with moderate complexity. The visual output provides an intuitive understanding of how the solution was found and how the path avoids obstacles to reach the goal efficiently.