This is a console-based car maze game which has several different modes and dynamically produced mazes with obstacles. It also has hints to help you find the shortest path to traverse the maze. It also maintains a list of trophies and high scores. It also simulates an AI-controlled car navigating through the maze using the shortest path.

Car Maze Game

Documentation for Explanation

Faraz Hayder

Table of Contents

1. Source.cpp

Main Function (main()):

- Dynamically allocates an integer array arr of size 3 using new.
- Enters an infinite loop to repeatedly play the car maze game.
- Calls carMazeGame() to obtain game parameters.
- Constructs a Graph object (Game) with dimensions and difficulty level.
- Invokes the main menu of the game (Game.MainMenu()).
- The loop continues indefinitely until the game is manually exited.
- Returns 0 upon successful execution.

Car Maze Game Function (carMazeGame()):

- Dynamically allocates an integer array arr of size 3 using new.
- Enters a loop to display the game menu until a valid option is selected.
- Uses system("cls") to clear the console screen.
- Prompts the user to select the difficulty level (Easy, Medium, Hard, or Quit).
- Validates the user input and sets values in the arr array accordingly.
- Returns the array arr with the selected difficulty level.
- Utilizes a recursive call to itself if an invalid option is chosen.

Data Structures Used:

- Dynamic Array (arr):
- Type: Integer array.
- Purpose: Stores game parameters - width, height, and difficulty level.
- Graph (Game):
- Type: User-defined class.
- Purpose: Represents the game board with dimensions specified by the user.

2. Header.h

Header Files:

- <iostream>: Input/output stream handling.
- <vector>: Standard template library for dynamic arrays.
- <stack>: Standard template library for the stack data structure.
- <conio.h>: Console input/output functions (used for getch()).
- <queue>: Standard template library for the queue data structure.

- <cstdlib>: Standard library for general-purpose functions (used for rand() and srand()).
- <ctime>: Standard library for handling time functions (used for seeding random number generation).
- <algorithm>: Standard template library for algorithmic operations (used for sort()).
- <windows.h>: Windows-specific header for Sleep function.
- <fstream>: Input/output stream class to operate on files.
- <list>: Standard template library for doubly-linked lists.

ANSI Escape Codes:

- Defines ANSI escape codes for text color to enhance console output.

Constants:

- INT_MAX: Maximum integer value, initialized to 2147483647.

Function Prototypes:

- int *carMazeGame(): Function to obtain game parameters, returning an array of integers.

Data Structures:

- **Node Structure (struct Node):**

  Represents a node in the maze grid with a character type.

- **Graph Class (class Graph):**

  Represents the maze and game logic.

  <u>**Attributes:**</u>

  - rows, columns: Dimensions of the maze.
  - cRow, cColumn: Current row and column of the car (player).
  - numVertices: Total number of vertices in the graph.
  - menu: Helps traverse to the Main Menu from the game.
  - maze: 2D vector representing the maze grid with nodes.

- adjacencyMatrix: 2D vector representing the adjacency matrix of the maze.
- pathNodes: Stack to store nodes in the shortest path.
- score: Player's score.
- powerUps: List of power-ups in the maze.

Methods:

- Graph(int r, int c): Constructor to initialize the graph with specified dimensions.
- MainMenu(int difficulty): Main menu method with difficulty selection.
- Continue(): Method to continue the game.
- StartGame(int difficulty): Method to start a new game with the specified difficulty.
- initializeMazeEasy(), initializeMazeMedium(), initializeMazeHard(): Methods to initialize the maze based on difficulty.
- addEdges(): Method to add edges to the graph.
- addEdge(int source, int destination, char type): Method to add an edge between two vertices.
- pathExists(int index): Method to check if the node is a valid node in the maze.
- ManualMode(): Method to play the game in manual mode.
- gamePlay(): Method to move the player in manual mode.
- moveUp(), moveDown(), moveLeft(), moveRight(): Methods to handle player movement.
- pauseMenu(): Method to display a pause menu.
- AutomatedMode(): Method to play the game in automatic mode.
- automatedGamePlay(): Method to move the player automatically.
- displayMaze(bool shortcut): Method to display the maze grid.
- displayAdjacencyMatrix(): Method to display the adjacency matrix.
- dijkstraShortestPath(int start, int end): Method to find the shortest path using Dijkstra's algorithm.

3. Methods.h

The maze is represented as a 2D grid, and the game involves navigating a car ('C') from the start ('S') to the end ('E') while avoiding obstacles ('#') and collecting power-ups ('*'). The implementation includes manual and automated gameplay modes, a scoring system, and a menu system.

Class: Graph

Attributes:

- rows: Number of rows in the maze.
- columns: Number of columns in the maze.
- numVertices: Total number of vertices in the maze grid.
- cRow, cColumn: Current position of the car in the maze.
- menu: Flag to control the game menu.
- maze: 2D vector representing the maze grid, storing nodes with type information.
- adjacencyMatrix: 2D vector representing the adjacency matrix of the graph.

Methods:

- Constructor (Graph(int r, int c)):
- Initializes attributes based on provided row (r) and column (c) parameters.
- Initializes the maze grid and adjacency matrix.
- initializeMazeEasy():
- Fills the maze grid with predefined elements for an easy-level maze.
- Initializes start ('S'), end ('E'), car ('C'), obstacles ('#'), and power-ups ('*').
- Populates the adjacency matrix based on maze connectivity.
- initializeMazeMedium():
- Similar to initializeMazeEasy() but with a more complex medium-level maze.
- initializeMazeHard():
- Similar to initializeMazeEasy() but with a highly complex hard-level maze.
- addEdges():
- Populates the adjacency matrix by adding edges between connected nodes in the maze grid.
- initializeMazeEasy, initializeMazeMedium, initializeMazeHard: Set up the maze with varying difficulty levels.
- addEdges: Create edges in the graph based on maze paths.
- pathExists: Check if a path exists at a given index in the maze.
- dijkstraShortestPath: Find the shortest path using Dijkstra's algorithm.
- displayMaze: Display the maze with optional shortcuts.
- MainMenu: Main menu to navigate between manual, automated, and exit options.
- Continue: Pause and wait for user input to continue.
- StartGame: Initialize the maze and ensure it is solvable.
- gamePlay: Handle player movements and interactions in manual mode.
- ManualMode: Execute manual gameplay.
- automatedGamePlay: Automatically navigate through the maze using the calculated path.
- moveUp, moveDown, moveLeft, moveRight: Functions to move the player in the respective directions.

- pauseMenu: Display a pause menu during gameplay.
- saveToFile: Save player name and score to a file.
- loadFromFileIntoTreeAndDisplay: Load scores from a file into a binary search tree and display them.

Properties:

- maze: 2D array of nodes representing the maze.
- adjacencyMatrix: Adjacency matrix representing connections between nodes.
- pathNodes: Stack storing nodes in the current path.
- powerUps: List storing collected power-ups.
- scoresTree: Binary search tree storing player scores.

Class Node:

- Represents a cell in the maze.

  Properties:

  type: Character representing the content of the cell ('S' for start, 'E' for end, 'C' for car/player, '*' for power-ups, '#' for obstacles, '-' for empty cells).

Scores:

- Player scores are saved to a file ("scores.txt").
- Scores are loaded into a binary search tree for display.

File Operations:

The program reads and writes player scores to the "scores.txt" file.

Dependencies:

The implementation uses the <iostream>, <cstdlib>, <ctime>, <fstream>, <conio.h>, and <windows.h> libraries for console-based functionality.

Data Structures Used:

- **2D Vector (maze):**

  Represents the maze grid as a 2D vector of nodes, where each node stores type information ('#', '*', ' ', etc.).

- **2D Vector (adjacencyMatrix):**

  Represents the graph's adjacency matrix, where each element indicates the connectivity between nodes in the maze.

- **Queue (gameElements):**

  Used to store game elements ('#', '*') for random placement in the maze grid.

- **Game Initialization:**

  The game initializes with a specified number of rows and columns.

  The maze is populated with obstacles, power-ups, a start point, an end point, and a car.