CS-2001 Data Structures (FALL-2023)

Semester Project: Design and Implement a Race Car Game

Contents

[Project Description 3](#_Toc152708958)

[Project Structure 3](#_Toc152708959)

[Classes 3](#_Toc152708960)

[Game Class: 3](#_Toc152708961)

[Data members: 3](#_Toc152708962)

[Functions: 4](#_Toc152708963)

[Binary Search Tree Class: 10](#_Toc152708964)

[Node Class: 10](#_Toc152708965)

[Menu: 10](#_Toc152708966)

[Data Structures 11](#_Toc152708967)

# Project Description

**Introduction: Race Car Game Project**

Greetings, In this semester's exciting project, we embark on a thrilling journey to apply our knowledge of data structures in a practical and engaging way. Our mission? Designing and implementing a captivating race car game through a maze using the power of C++.

**Project Objective:** The primary goal of this project is to craft an interactive and entertaining race car game that not only showcases our programming prowess but also leverages various data structures. Throughout the development process, we'll delve into the efficient management of game elements, exploring the application of data structures such as graphs, binary search tress, and queues.

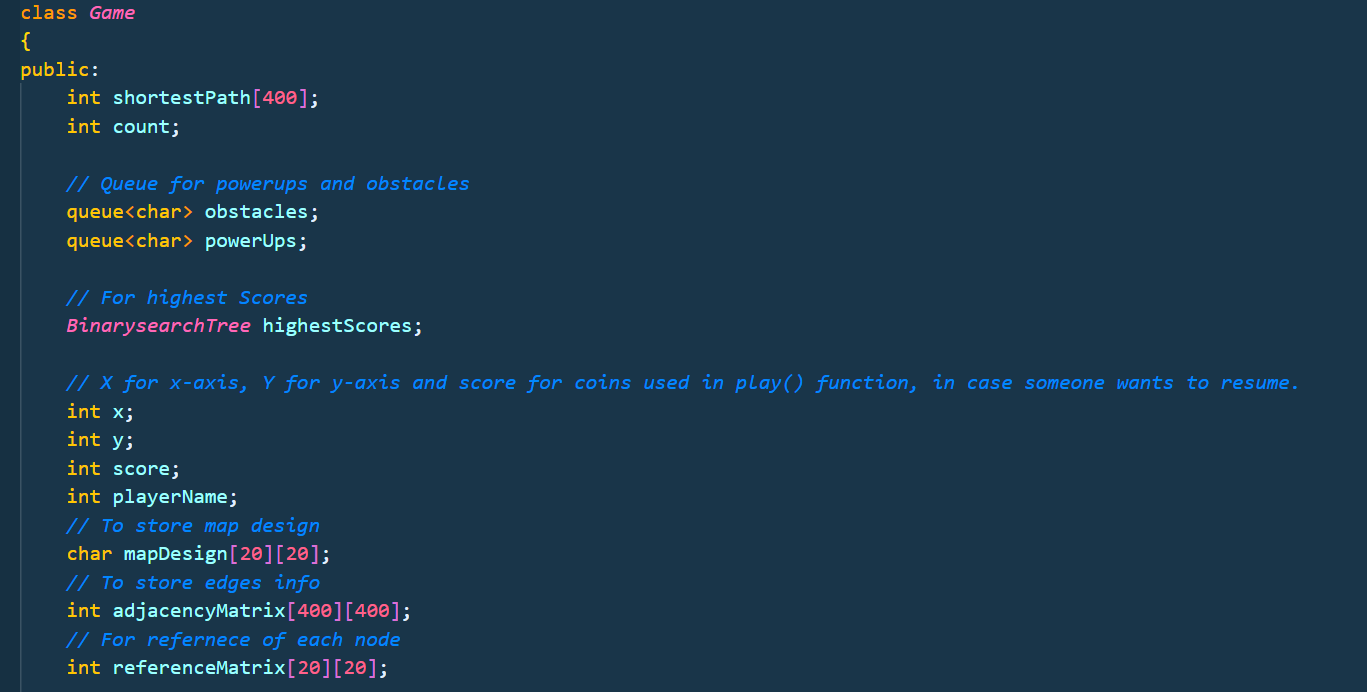
# Project Structure

## Classes

## Game Class:

The main class of this project is game class through which the whole game is being managed it has different functions for different features of our Car Game Project. Lets talk about the members and functions of game class.

### Data members:



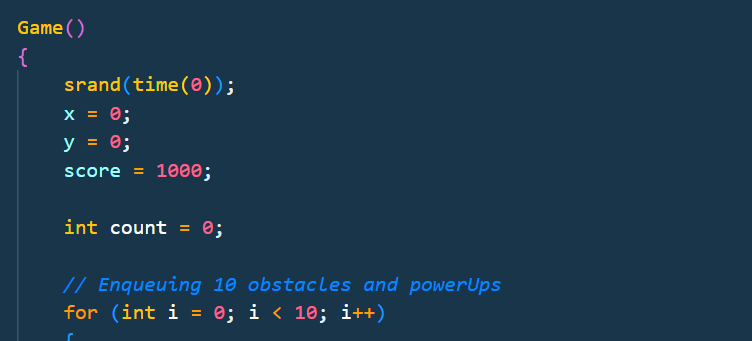
The game class have following members:

* ShortestPath: Its an array of integer type to store the shortest path vertices on the graph obtained from Dijsktra Algrithm.
* Obstacles: Its an queue data structures object used to store obstacles of the game.
* Powerups: it’s an queue data structures object used to store powerups on the map.
* highestScores: its an BinarySearchTree object keeps the record of the score of different players and also represent them in descending order for comparison.
* X & Y: There are integers that stores the current coordinates of car on the map.
* Score: Integer to store the score of the player.
* PlayerName: To keep the track of score along with its player name.
* MapDesign: A 20x20 char array that stores the map of the car racing game.
* AdjacencyMatrix: A integer array to store the information of the map in the from of a graph.
* ReferenceMatrix: A matrix to keep track of vertice’s number on mapDesign.

### Functions:

#### Game Constructor:

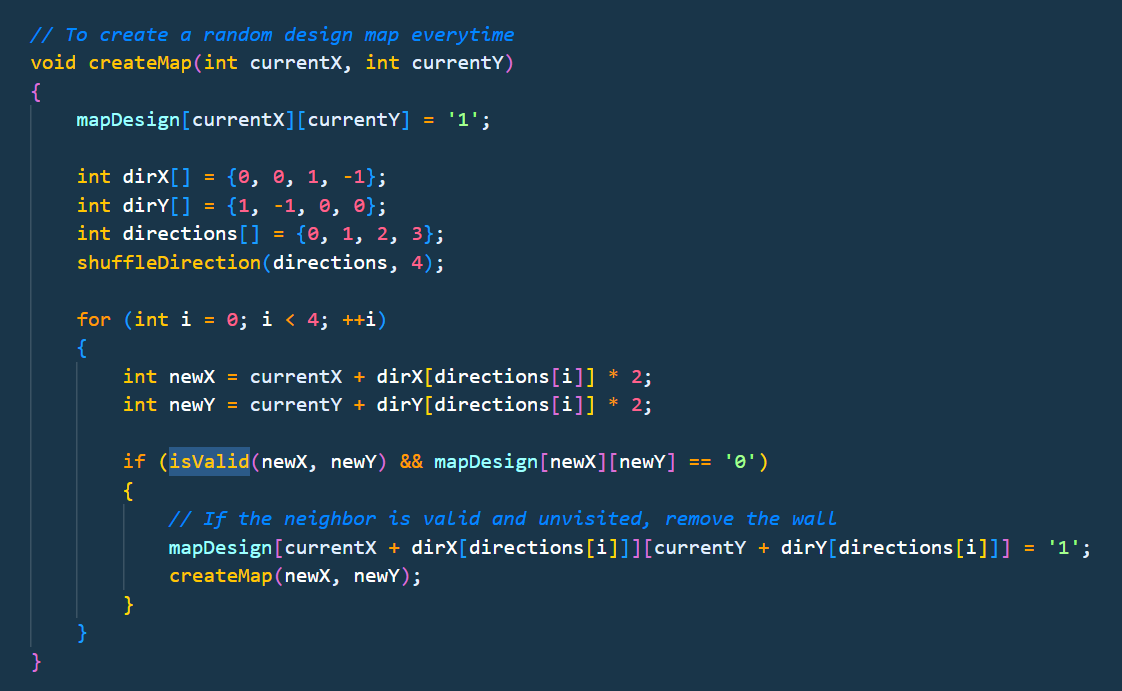
First of all we have Game class constructor to initialize data members like mapdesign, adjacencyMatrix etc.

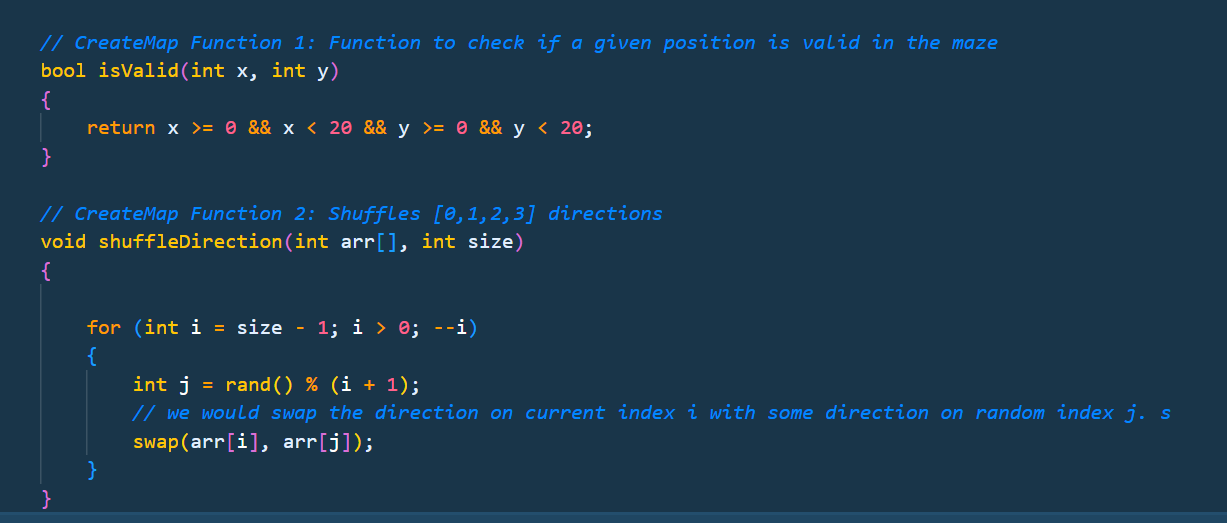


#### Depth First Search Algorithm for map:

* In order to generate a random map for car racing we have used depth first search algorithm.
* In this algorithm three functions are involved.
  + - createMap: which is the main function of this algorithm that generate a random map on mapDesign array.
    - shuffleDirection: which is used to shuffle the 4 directions (up, down, left, right) in order to get a random map design.
    - isValid: The third function to check if the position that we are trying to visit is inside the 2d

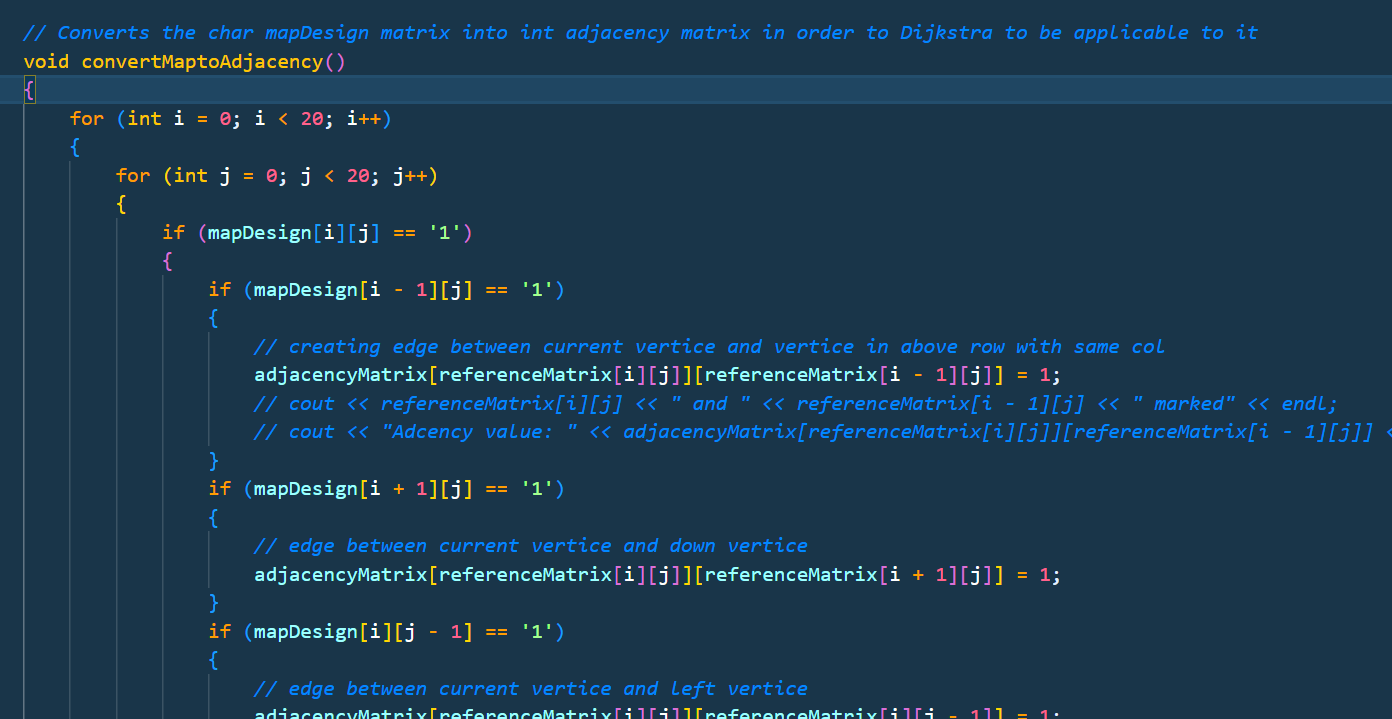
array mapDesign.





#### Converting Map Design to Adjacency Matrix

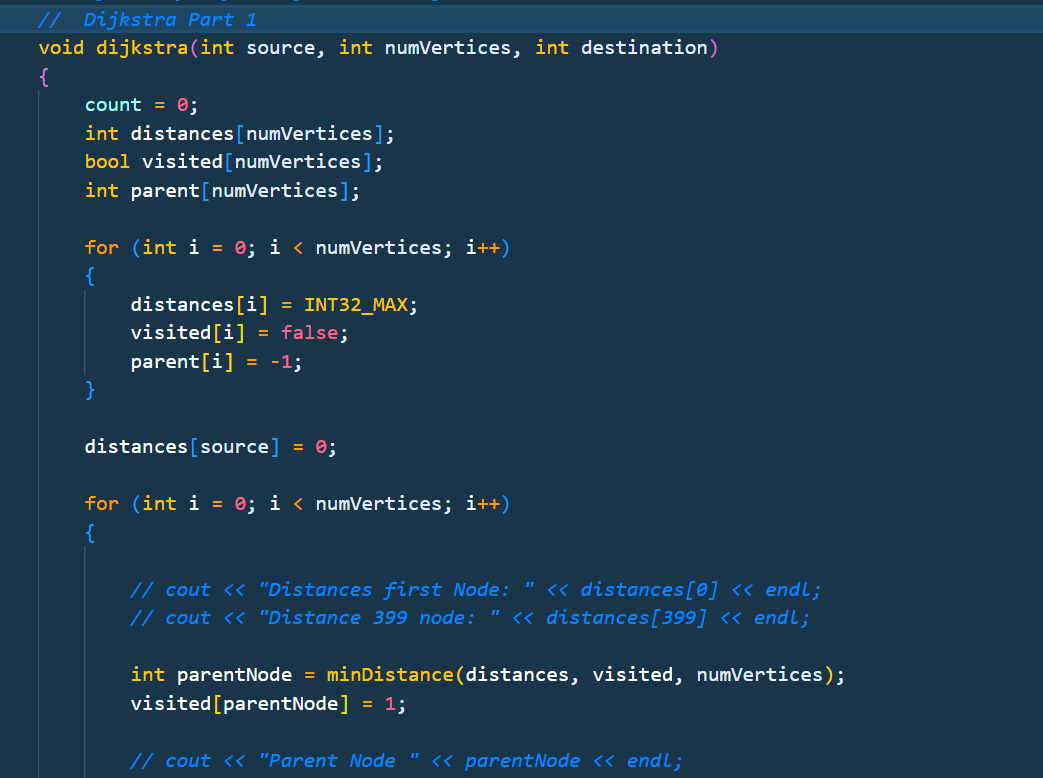
* In order to apply graph functions/algorithm on the mapDesign we first have to convert it into adjacency Matrix for that we have used void convertMaptoAdjacency() function.



#### Dijkstra Algorithm:

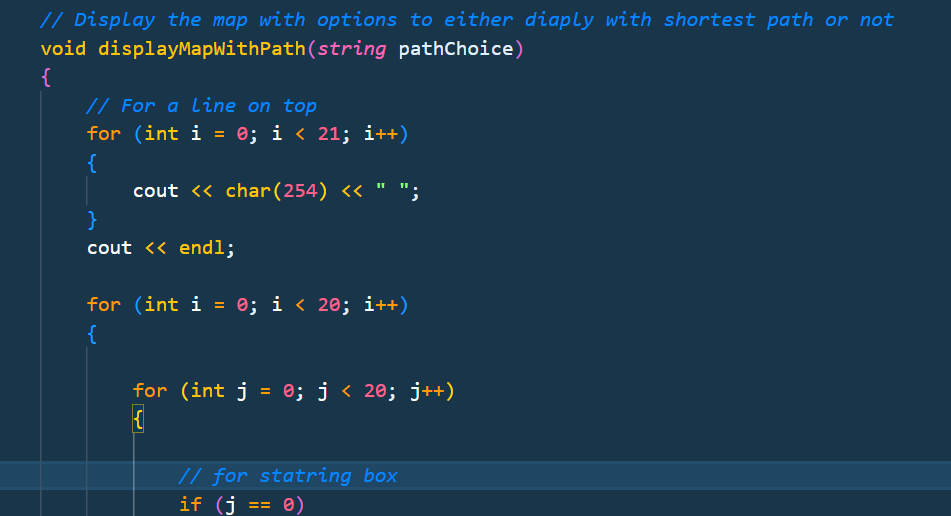
In order to find shortest distance on the random map we choose to use Dijkstra algorithm to find the shortest path. There are three functions involved in this algorithm.

* void dijkstra(int source, int numVertices, int destination): Which is the main function.
* int minDistance(int distances[], bool visited[], int numVertices): It is use to find the min distances among the vertices that are not currently visited.
* void storeShortestPath(int parent[], int j): To store the shortest Distance path.



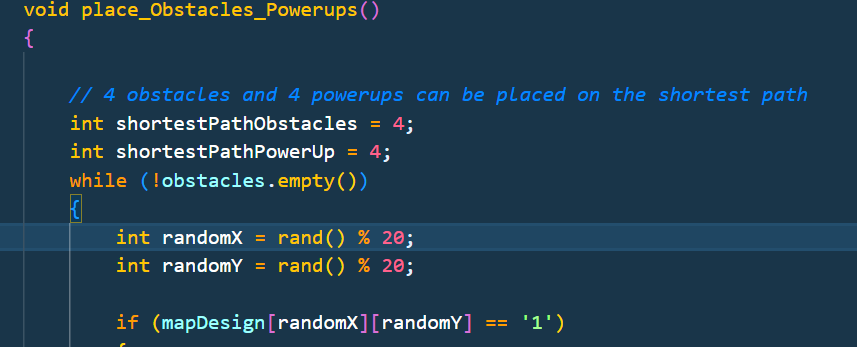
#### Displaying Map:

   void displayMapWithPath(*string* pathChoice): This function is used to display the mapDesign while playing the game. It has choice whether to display map showing the shortest path or without it.



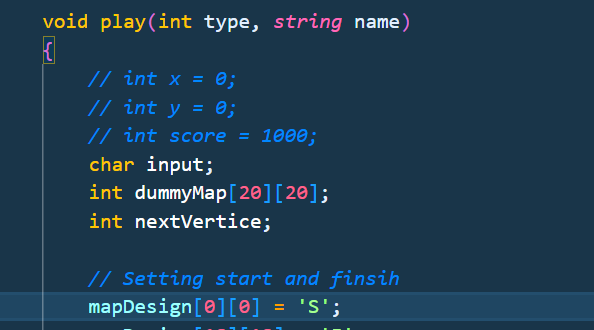
#### Placing Obstacles and Powerups

In order to place Obstacles and powerups randomly on the mapDesign we hace used function     void place\_Obstacles\_Powerups()



#### Playing Game:

We have wrote function     void play(int type, *string* name), which is used for playing the game and managing scores, obstacles, Powerups and displaying the map. It has two parameters first is game type (e.g. automatic, manual, semi-automatic) and second is player name.



## Binary Search Tree Class:

This class is used for storing the scores and the players for the scoreboard displaying highest scorer to lowest scorer. It have Functions:

* *Node* *\**insertR(*Node* *\**temp, *string* name, int score): To insert new Player name and its score.
* void writeToFile(): In order to write scores and player name on file i-e scoreRecord
* void writeToFileRecursive(Node \*temp, ofstream &outFile): Which is also part of writeToFile().
* void readFromFile(): This function is used to read file I-e scoreRecord and store the information of player name and score in binarySearchTree.
* void displayTopScores(Node \*temp, int &ranking): To Display Player name and there scores in descending order.

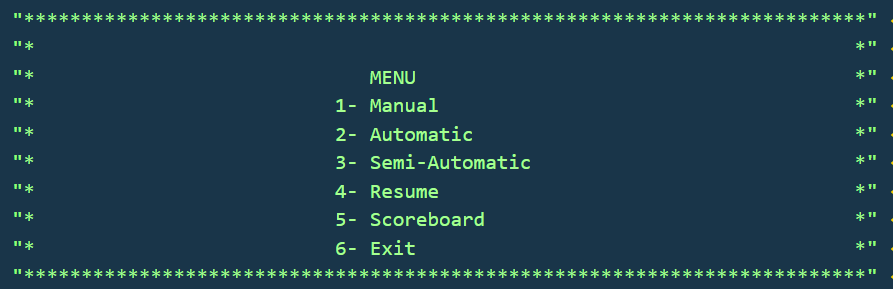
## Node Class:

The Node class is for individual nodes of BinarySearchTree.

## Menu:

We have a user-Friendly menu, which is minimalistic and easy to understand our menu have following options.

1. Manual: That creates a new graph object, and start a game with name map by asking player name.
2. Automatic: This resume that already create game but automatically navigate the car on the shortest path.
3. Semi-Automatic: This is also a new Game which shows the shortest path in form of dots but user have to move the car himself.
4. Resume: This options continue the game from where it was left in Manual or Semi-Automatic Game..
5. Scoreboard to display scores with the player name in descending order.
6. Exit: To exit the game.



# Data Structures

1. **Array:**
   * **Usage:** The array is used for multiple purposes, such as storing the game map design, keeping track of the shortest path vertices, and maintaining the adjacency matrix for graph operations.
   * **Details:**
     + **Map Design:** A 20x20 char array (**MapDesign**) is utilized to represent the layout of the car racing game, where each element of the array corresponds to a specific position on the map.
     + **Shortest Path:** An array (**ShortestPath**) of integers is employed to store the vertices of the shortest path obtained from the Dijkstra algorithm.
     + **Adjacency Matrix:** An integer array (**AdjacencyMatrix**) serves as an adjacency matrix, representing the connectivity information of the map.
2. **Graph :**
   * **Usage:** For Dijkstra Algorithm to find the shortest path on the mapdesign on adjacency matrix. Depth First Search Algorithm is also used to generate a random map for car racing.
   * **Details:**
     + **Functions:**
       - **convertMaptoAdjacency():** Converts the MapDesign array into an adjacency matrix (AdjacencyMatrix) to enable graph-based operations.
       - **dijkstra(int source, int numVertices, int destination):** Main function implementing Dijkstra's algorithm to find the shortest path on the graph.
       - **minDistance(int distances[], bool visited[], int numVertices):** Finds the minimum distance among vertices that are not currently visited.
       - **storeShortestPath(int parent[], int j):** Stores the shortest distance path for later display.
     + **Purpose:** The graph structure facilitates efficient exploration and computation of the shortest path on the randomly generated game map, contributing to the core gameplay mechanics.

This refinement emphasizes the specific role of the graph data structure in supporting Dijkstra's algorithm for calculating the shortest path in the context of the Race Car Game.

1. **Queue:**
   * **Usage:** Queues are used to store obstacles (**Obstacles**) and powerups (**Powerups**) on the game map. The queue data structure is particularly useful for managing these dynamic elements during gameplay.
   * **Details:**
     + **Obstacles:** A queue object (**Obstacles**) is employed to store obstacles randomly placed on the map.
     + **Powerups:** Another queue object (**Powerups**) is used to manage the placement of powerups on the game map.
2. **Binary Search Tree:**
   * **Usage:** The Binary Search Tree (BST) is utilized to store and manage player scores along with their names. It allows for efficient retrieval and display of the top scores.
   * **Details:**
     + **Functions:**
       - **insertR:** Inserts a new player name and score into the BST while maintaining the binary search tree properties.
       - **writeToFile:** Writes scores and player names to a file (**scoreRecord**).
       - **readFromFile:** Reads the file (**scoreRecord**) to populate the BST with player information.
       - **displayTopScores:** Displays player names and scores in descending order.

These data structures collectively contribute to the functionality and efficiency of your Race Car Game, enabling various game features and ensuring a smooth gaming experience.

Top of Form