DSA Project Report

Department of Computer Science

Iqra University Islamabad

Report

Maqsood Ahmed	38186
Muhammad Shahzaib	38306

Final Semester Project Report

Design and Implementation of a Race Car Game:

1. Introduction

The semester project for the CS-2001 Data Structures course involved designing and implementing a 2D console-based race car game using C++. The primary objective of the project was to apply data structures such as graphs, linked lists, queues, stacks, and binary trees to manage game elements efficiently. The game features a text-based user interface (TUI), player-controlled race cars, automatic cars, obstacles, tracks, and a scoring system. This report documents the design, implementation, and testing of the game, along with the usage of data structures to enhance its functionality.

2. Project Objectives

The project aimed to achieve the following objectives:
1. Develop a 2D Console-Based Game: Create a race car game using C++ without external graphics libraries. The game features a text-based user interface (TUI) with ASCII characters representing game elements.
2. Integrate Data Structures: Use data structures such as queues, stacks, binary trees, and linked lists to manage game elements such as maps, obstacles, scoring, and leaderboards.
3. Implement Game Logic: Develop player controls, collision detection, scoring, and win/lose conditions.
4. Design a User-Friendly Interface: Create an intuitive and visually appealing TUI for the game.
5. Document and Test: Provide comprehensive documentation and thoroughly test the game for bugs and performance issues.

3. Data Structures Used

3.1 Queues for Obstacle and BFS Management

- BFSQueue.h: A queue data structure was used to implement Breadth-First Search (BFS) for navigation and pathfinding in the game. This helped in finding the shortest path for the player-controlled race car.
- Queue.h: A standard queue was used to manage the generation of obstacles. New obstacles were enqueued as they were generated, and the game loop dequeued them to introduce obstacles at appropriate times.

3.2 Priority Queue for Game Management

- PriorityQueue.h: A priority queue was implemented to manage game events or tasks based on their priority. This was particularly useful for handling time-sensitive events in the game.

3.3 Stack for Game State Management

- Stack.h: A stack data structure was used to manage the game state, such as tracking the history of player moves or managing undo operations.

3.4 Binary Tree for Leaderboard Management

- LeaderBoard_Binary_Tree.h: A binary tree was implemented to manage the leaderboard. This allowed for efficient insertion, deletion, and retrieval of player records, ensuring that the leaderboard was always sorted and up-to-date.

3.5 Linked List for Scoring Management

- **Scoring_List.h:** A linked list was used to track the player's score and collected items. Each node in the linked list represented a score update or a collected item, allowing for easy tracking and updating of the player's progress.

4. Game Logic

The game logic was implemented to ensure smooth gameplay and an engaging experience for the player. Key features included:

- **Player Controls:** The player could control the race car using the following keys:
  - **W: Accelerate**
  - **A: Turn Left**
  - **D: Turn Right**
  - **O: Quit**
- **Automatic Cars:** The **AutomaticCar_CameManager.h** file managed the behavior of automatic cars in the game, ensuring they followed predefined paths and interacted with the player.
- **Collision Detection:** Collision detection was implemented to handle interactions between the race car and obstacles or collected items.
- **Scoring System:** A scoring system was implemented to track the player's progress. Points were awarded for collecting coins and completing laps.
- **Win/Lose Conditions:** The game ended when the player either completed the race or collided with too many obstacles.

5. User Interface Design

The game featured a text-based user interface (TUI) designed to be intuitive and visually appealing. Key components of the TUI included:

- **Main Menu:** Displayed options to start the game, view instructions, or quit.
- **Game Screen:** Used ASCII characters to represent the race car, obstacles, and track.
- **Scoreboard:** Displayed the player's current score and progress.
- **Controls:** Provided on-screen instructions for player controls.

6. Implementation Details

6.1 Code Structure

The project was implemented in C++ and organized into the following files:

- **Main.cpp:** Contains the main game loop and logic.
- **Game_Manager.h:** Manages the overall game logic and state.
  Manages game objects such as cars, obstacles, and collectibles.
- **BFSQueue.h:** Implements a queue for BFS navigation.
- **PriorityQueue.h:** Implements a priority queue for game event management.
- **LeaderBoard_Binary_Tree.h:** Implements a binary tree for leaderboard management.
- **Scoring_List.h:** Implements a linked list for scoring management.
- **Stack.h:** Implements a stack for game state management.

6.2 Key Algorithms

- **Breadth-First Search (BFS):** Used for pathfinding and navigation.
- **Collision Detection**: Implemented using bounding box checks.
- **Queue Operations:** Enqueue and dequeue operations for obstacle management.
- **Binary Tree Operations:** Insertion, deletion, and traversal for leaderboard management.
- **Linked List Operations:** Insertion and traversal for tracking collected items.

7. Testing and Optimization

The game was thoroughly tested to ensure it ran smoothly and without bugs. Key testing phases included:

- **Unit Testing:** Individual components (e.g., queue, stack, binary tree) were tested for correctness.
- **Integration Testing:** The game logic and data structures were tested together to ensure proper interaction.
- **Performance Testing:** The game was optimized for performance, with a focus on reducing memory usage and improving response times.

8. Challenges Faced

- **Pathfinding:** Implementing BFS for navigation was challenging due to the need to handle dynamic obstacles.
- **Collision Detection:** Ensuring accurate collision detection in a text-based environment required careful implementation.
- **Leaderboard Management:** Implementing an efficient binary tree for the leaderboard required careful handling of insertions and deletions.

9. Conclusion

The race car game project successfully applied data structures such as queues, stacks, binary trees, and linked lists to create an engaging and interactive game. The project demonstrated the importance of data structures in managing game elements efficiently and improving performance. The game was well-received during testing, and the objectives of the project were met.

10. Future Work

- **Enhanced Graphics:** Implement a graphical user interface (GUI) using external libraries.
- **Multiplayer Mode:** Add support for multiple players to compete in real-time.
- **Advanced AI:** Implement AI-controlled opponents with more sophisticated pathfinding algorithms.
- **Additional Features:** Introduce power-ups, different track designs, and varying difficulty levels.

The End