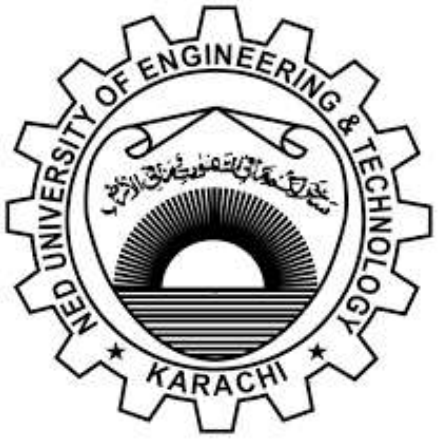

NED UNIVERSITY OF ENGINEERING & TECHNOLOGY
Department of Computer Science & Information Technology

TRAFFIC RACER GAME

PROJECT REPORT

Submitted By:

Ali Muhammad Salim - CT-24083

Abdul Rahim - CT-24088

Muhammad Fouzan Abdul Aziz - CT-24090

Syed Irtaza Shahid Zaidi - CT-24072

1. Introduction

This report presents the design and implementation of a Traffic Racer game developed using C++ and Raylib. The project demonstrates object-oriented programming, real-time rendering, game state management, collision detection, threading, and multiple data structures essential for efficiency and smooth gameplay.

2. Project Overview

The Traffic Racer game is a lane-based racing simulation where the player avoids incoming vehicles, collects power-ups, earns points, and progresses through increasingly challenging levels. The game features animated backgrounds, multiple dynamic scenes, power-up effects, particle systems, sound effects, and a scoring system with high score tracking.

3. System Architecture

The project is built around modular classes such as Car, EnemyManager, PowerUpManager, ScoreManager, SceneManager, and TrafficRacingGame. Each class manages a specific set of responsibilities, improving readability, scalability, and maintainability.

4. Data Structures Used

Vector (std::vector)

Used for dynamic lists such as enemies, power-ups, particles, and jobs. Offers fast iteration and amortized O(1) insertion.

Quadtree (Custom Implementation)

A spatial partitioning tree used to improve collision-detection efficiency by limiting checks to nearby objects.

Priority Queue (std::priority_queue)

Used in the event scheduler to handle timed events like spawning enemies or power-ups at future frame counts.

Thread-Safe Job Queue

Implemented using mutex and condition_variable to perform asynchronous file I/O without freezing the main game loop.

CollisionBox Struct

Supports fast axis-aligned collision detection between the player, enemies, and power-ups.

Enums

Used for game states, scene types, and power-up categories to keep the code clean and readable.

```
enum GameState { MENU, PLAYING, PAUSED, GAME_OVER, SCORES };
enum SceneType { CITY, HIGHWAY, DESERT, NIGHT, FOREST, SNOW, SUNSET, RAIN };
enum PowerUpType { SHIELD, SLOW_MOTION, SCORE_MULTIPLIER, EXTRA_LIFE };
```

5. Algorithms Implemented

• Lane selection algorithm ensures enemies are spawned in safe lanes.
• Power-up spawn logic avoids conflict with enemy lanes.
• Collision detection uses Quadtree query operations.
• Event scheduler uses timed callbacks for smooth gameplay control.
• Level progression algorithm increases game difficulty dynamically.

6. Game Features

• 8 dynamic background scenes
• Smooth lane switching animation
• Power-ups (Shield, Slow Motion, Score Multiplier, Extra Life)
• Camera shake effects on collision
• Particle explosion effects
• High score saving using asynchronous jobs
• Optimized collision system with Quadtree

7. Conclusion

This Traffic Racer project demonstrates how data structures, algorithms, and programming principles combine to build a complete, interactive real-time game. The use of advanced structures such as Quadtrees and asynchronous job queues shows the importance of performance optimization in game development.

8. References

• Raylib Documentation
• C++ Standard Library
• Object-Oriented Programming Concepts
• Data Structures and Algorithms theory