Case Study Report

**Title: Car Physics Simulation  
Name: Granth Sidhant Yadav   
Date: 8th June 2025**

# Introduction

This case study presents a simplified car physics simulation in a game environment using Unity and C++. The goal is to model realistic vehicle behavior such as acceleration, braking, friction, suspension, and collisions. As a BCA final-year student with strong knowledge of C++ and databases, this report interprets the case study and presents a plan using my existing technical domain.

# Case Study Interpretation

The simulation models how a car moves and reacts to its surroundings using physics principles. Unity is used for rendering and physics interactions, while C++ is used for implementing core physics calculations. Although I have no prior experience with Unity, I understand the physics involved, such as Newton's laws, friction, and collisions.

# Database Schema Design

This schema helps log vehicle test results, collision data, and simulation metrics.

**Vehicles Table**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| vehicle\_id | INT (PK) | Unique vehicle ID |
| model\_name | VARCHAR | Name of vehicle model |
| mass | FLOAT | Vehicle mass in kg |
| engine\_power | FLOAT | Power of the engine |

**Simulation\_Logs Table**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| log\_id | INT (PK) | Unique log ID |
| vehicle\_id | INT (FK) | Link to Vehicles |
| start\_time | DATETIME | Start of simulation |
| duration\_sec | INT | Duration in seconds |
| surface\_type | VARCHAR | Asphalt, Dirt, etc. |
| max\_speed | FLOAT | Max speed during simulation |
| collisions | INT | Collision count |

**Collision\_Events Table**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| event\_id | INT (PK) | Unique ID for each collision |
| log\_id | INT (FK) | Link to Simulation Logs |
| impact\_force | FLOAT | Force in Newtons |
| collision\_angle | FLOAT | Angle of impact |
| damage\_level | VARCHAR | Low / Medium / High |

# System Flow Diagram

System flow steps:

* 1. Start Simulation → 2. Load Car Model → 3. Apply Physics (acceleration, friction, collision) → 4. Record Data → 5. Display in UI / Export to DB
* [User Input (Keyboard / Controller)]  
   ↓  
  [C++ Logic Layer]  
  - Calculates acceleration, braking  
  - Applies physics equations  
   ↓  
  [Unity Engine]  
  - Renders car visuals  
  - Handles collisions & suspension  
   ↓  
  [Simulation Outputs]  
  - Shows live car movement and reactions  
   ↓  
  [Logging System (Optional)]  
  - Stores simulation metrics  
   ↓  
  [Database (MySQL or MongoDB)]

# Code Architecture

• C++ Module: Contains core physics logic (force, collision response)  
• Unity Frontend: Handles visuals, interactions, and Unity physics engine (Wheel Collider, Rigidbody)  
• Database Layer: Logs outputs of simulation for testing and analysis

# Ground Work & Benchmarking

\*\*Ground Work:\*\*  
Understanding physics concepts like Newton’s laws, friction types, and basic suspension behavior.

\*\*Benchmarking Insight:\*\*

Benchmarking means comparing your simulation model with existing systems to understand industry best practices and performance expectations.  
  
1. Arcade vs Realistic Simulations:  
Arcade-style games (e.g., NFS, GTA) use simplified physics for fun gameplay. Realistic games (e.g., Assetto Corsa, BeamNG.drive) focus on detailed vehicle dynamics.  
→ Insight: Choose simplicity or realism based on purpose.  
  
2. Game Engine Comparison:  
Unity offers easy tools like Rigidbody and Wheel Collider. Custom engines (via C++) give control but are complex.  
→ Insight: Unity tools offer faster setup, good for learning.  
  
3. Performance vs Accuracy:  
Detailed physics need more computing power. Balancing accuracy and smooth performance is key.  
→ Insight: Realism should be believable, not overly complex.

# Implementation Plan

Step 1: Define Core Physics  
- Apply forces (acceleration, gravity)  
- Implement friction by road surface  
- Use Unity Rigidbody for motion  
  
Step 2: Wheel Physics  
- Unity’s Wheel Colliders simulate tire movement  
- Adjust for road changes  
  
Step 3: Collision System  
- Detect collisions using Raycasting  
- Apply impact forces and bounce-back  
  
Step 4: Suspension System  
- Adjust spring settings for terrain  
- Improve balance with anti-roll bars

Step 5: Testing  
- Run stress tests  
- Optimize performance and debug edge cases

# Conclusion

This case study helped understand the fundamentals of building a physics simulation. While I'm new to Unity, my strong foundation in C++ and systems design enables me to learn and contribute effectively. I’m eager to apply these skills in real-world simulation environments like this.