# C++ PROGRAMMING

# (OCSE-306T)

# PROJECT REPORT

SUBMITTED BY:

Samarth Khanna (108) , Mohit Rana (120) , Sonu Sharma (127)

6th Semester ECE-B

Batch: 2022-2026

SUBMITTED TO:

Dr. Tanima Ghosh

Assistant Professor

Department of Electronics and Communication Engineering

**Department of Electronics and Communication Engineering**

**Bhagwan Parshuram Institute of Technology**

# Car Racing Game

By: Samarth Khanna (108) , Mohit Rana (120) , Sonu Sharma (127)

## Abstract

The "Car Racing Game" is a 2D racing simulation developed using C++ and SFML (Simple and Fast Multimedia Library). It simulates a simple car racing environment where multiple AI-controlled cars navigate a race track defined by a set of checkpoints, while the player controls one car using keyboard inputs. The game emphasizes movement physics, path-following AI, and basic collision detection, providing an interactive simulation of autonomous and player-driven cars in a racing context.

## Objectives

- Develop a functional 2D car racing game using C++ and SFML.
- Implement autonomous movement for AI-controlled cars using pathfinding logic.
- Integrate player controls for a manually driven car using real-time keyboard inputs.
- Simulate realistic movement and turning mechanics including acceleration, deceleration, and angular motion.
- Create a visually interactive environment with smooth animations and sprite handling.
- Introduce simple collision detection to avoid car overlapping during motion.

## Project Overview and Description

This project is a 2D car racing simulation game developed in C++ using the SFML (Simple and Fast Multimedia Library). The game allows a player to control a car in a top-down environment, racing alongside AI-driven opponents who follow a path marked by predefined checkpoints.

### Game Overview

This game simulates a basic 2D racing scenario where:

- One player-controlled car is maneuvered using keyboard arrow keys.
- Four AI-controlled cars follow a series of predefined checkpoints (waypoints).
- The race track is rendered using a 2x scaled background image and dynamic sprite rendering.
- The game runs in a window using SFML's rendering engine, supporting frame rate limiting and real-time updates.

## Features

- Realistic motion using physics-based calculations
- AI cars with dynamic pathfinding through checkpoints
- Keyboard controls for intuitive player input
- Collision handling to maintain spatial realism
- Camera tracking for enhanced player experience
- Distinct sprite rendering with unique colors for each car

## Core Mechanics

- Smooth 2D car movement using realistic physics with sin and cos for angle-based motion.
- Player controls for acceleration, deceleration, and turning (red car starts at a fixed position).
- AI cars use findTarget() to compute angles to the next waypoint and move accordingly.
- Collision detection and resolution prevent car overlaps using positional adjustments.
- Camera smoothly follows the player's car to keep it centered on the screen.
- Cars have individual speed, turning radius, and angular movement

## How the game works ?

- Start: Player launches the game; a 640x480 window opens.
- Initial Scene: A road map background appears, with 5 colored cars.
- Player Control:
  - Up Arrow: Accelerate
  - Down Arrow: Brake/Reverse
  - Left/Right Arrows: Steer left/right
  - Esc: Exit the game window
- AI Behavior: Other cars follow a route defined by 8 waypoints.
- Real-Time Rendering: Cars move and rotate based on angle and speed.
- Collision Handling: Prevents car overlaps via simple spatial corrections.
- Camera Tracking: The screen follows the player’s car, giving a smooth experience.

## System Requirements

- Operating System: Windows / Linux / macOS
- Compiler: C++17 compatible (e.g., g++, MSVC)
- Libraries: SFML 2.5+ (Graphics, Window, System, Audio, Network)
- Build System: CMake 3.10 or higher

## Code Structure Overview

- main.cpp: Core game logic
    - Initializes cars, loads textures
    - Handles input, movement, and rendering
- Car struct:
    - Stores car position, speed, angle, and checkpoint index
    - Handles movement and AI target finding
- CMakeLists.txt:
    - Manages build settings and SFML dependencies

### 1. Header Files

cpp

CopyEdit

```
#include <cmath>

#include <SFML/Graphics.hpp>
```

Used for trigonometric calculations and SFML graphics handling.

### 2. Constants and Waypoints

cpp

CopyEdit

```
const int num=8;

int points[num][2] = {...};
```

Defines 8 checkpoints for AI navigation.

### 3. Car Structure

cpp

CopyEdit

```
struct Car {

  float x,y,speed,angle;

  int n; // Current checkpoint index

  Car() {speed=2; angle=0; n=0;}

  void move();        // Moves the car forward
```

void findTarget(); // Adjusts angle towards the next checkpoint };

Encapsulates car data and behavior. Each car has:

- Position (x, y)
- Speed
- Angle of direction
- Checkpoint tracking

## 4. Main Function

cpp

CopyEdit

int main() {

RenderWindow app(VideoMode(640, 480), "Car Racing Game!");

...

}

## Key components:

- Window Initialization: Sets frame rate and loads background/car textures.
- Car Initialization:

cpp

CopyEdit

const int N=5;

Car car[N];

for (int i = 0; i < N; i++) ...

Initializes 5 cars at different starting positions and speeds.

- User Input: Keyboard arrow keys control the player's car.
- Movement Update:
    - Applies physics (acceleration, deceleration).
    - Updates position using move() method.
    - AI cars use findTarget() to follow checkpoints.
- Collision Detection:

  cpp

  CopyEdit

  for(int i=0;i<N;i++)

```
for(int j=0;j<N;j++)
```

Prevents cars from overlapping using a basic repulsion method.

- Camera Offset: Keeps the player's car centered by adjusting the background and car drawing positions.
- Drawing: Renders the background and car sprites with different colors.

## CMake Configuration

cmake

CopyEdit

```
project(Racing)

find_package(SFML REQUIRED network audio graphics window system)

file(COPY "${CMAKE_CURRENT_SOURCE_DIR}/images" DESTINATION
"${CMAKE_CURRENT_BINARY_DIR}/")

add_executable(racing main.cpp)

target_link_libraries(racing PRIVATE sfml-system sfml-window sfml-graphics sfml-network
sfml-audio)
```

This configuration:

- Sets the project name as Racing.
- Finds and links the SFML libraries required for multimedia handling.
- Ensures the images directory is copied to the build directory.
- Compiles and links the main.cpp file to build the final executable.

## How to Build and Run

- **Step 1: Clone or download the project.**

  ```
  git clone <repository_url>
  cd Racing
  ```

- **Step 2: Install SFML**

  - On Linux: sudo apt-get install libsfml-dev
  - On Windows: Download binaries from SFML Downloads and configure paths accordingly

- **Step 3: Build the project using CMake**

```
mkdir build && cd build
cmake ..
make
./racing
```

## Deployment Guide

- **Windows:**
  1. Bundle racing.exe with the images/ folder
  2. Include SFML .dll files in the same directory
  3. Use a tool like Inno Setup to create an installer

- **Linux:**
  1. Make the binary executable
  2. Package it with images/ in a tarball:
  ```
  tar -czvf racing_game.tar.gz racing images/
  ```
  3. Include SFML runtime libraries or list them in dependencies

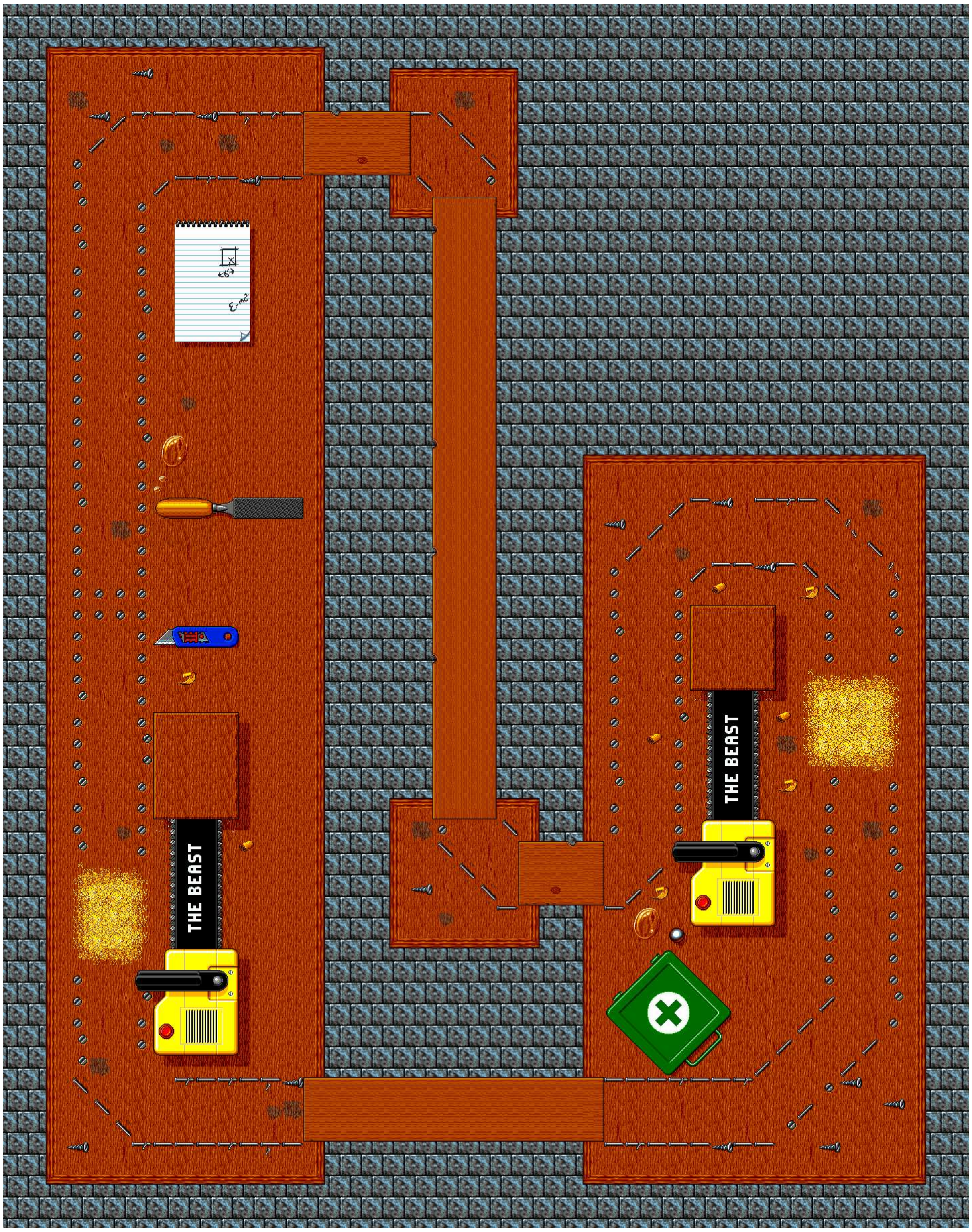
- **Cross-platform:**
  1. Use tools like CMake + CPack to create OS-specific packages

## Sample Output (Gameplay Snapshot)

Imagine the following scene:

- A map with long background track with twists and turns.
- 5 cars of different colors:
  - Red: Player car
  - Green, Magenta, Blue, White: AI cars
- The red car in the center is controlled by you using arrow keys.
- Other cars move smoothly along the path, adjusting angles at checkpoints.
- Camera scrolls and follows as your car moves.
- Smooth sprite rotation based on movement direction.
- Collision between cars causes slight repulsion.

Gameplay feels dynamic, and responsive, and gives a good foundation for further enhancements like scoring, laps, or obstacle placement.


THE BEAST
THE BEAST

## Possible Enhancements

- Add lap timing and scoreboards.
- Implement obstacles and track boundaries.
- Include sound effects and background music.
- Add power-ups, nitro boosts, or damage mechanics.
- Create levels or different maps.
- Add UI (speedometer, lap timer)
- Track boundary limits and crash effects
- Multiplayer support (local or network)

## Conclusion

This project successfully demonstrates the basics of C++ programming of a simple car racing game. It provided hands-on experience with loops, conditionals, and user interaction. The project was useful in practicing both logic implementation and user experience design within the constraints of a terminal-based application.

## License

This project is released under the MIT License.

## Appendix: Full Source Code

The full source code of the game can be found at:

https://github.com/samarthkhanna2741/Car_Racing_game.git