

Source Code Availability

The complete source code is available on GitHub:

 <https://github.com/hslopez/AutoDrivingCar/>

```
git clone https://github.com/hslopez/AutoDrivingCar.git
```

Design & Architecture

Core Components

1. Interfaces-Driven Design

- **ICar**: Defines car operations (movement, rotation, collision handling)
- **IField**: Manages field boundaries and validation
- **ISimulation**: Orchestrates command processing and collision detection

2. SOLID Principles

- Single Responsibility: Each class handles specific domain logic
- Open/Closed: Extensible through interfaces
- Dependency Inversion: Components depend on abstractions

3. Key Features

- Step-by-step command processing
- Collision detection with step tracking
- Field boundary validation
- Immutable collision records

Assumptions

1. Input Validation

- Users provide valid commands (L/R/F)
- Car positions start within field boundaries
- Unique car names enforced

2. Movement Rules

- Invalid moves are ignored
- Rotation happens instantaneously
- Collisions stop further movement

3. Field Definition

- Zero-indexed grid (0,0) to (Width-1, Height-1)
- Minimum field size 1x1

Environment Requirements

Supported Platforms

- **Windows** 10/11 (x64)
- **Linux** (Ubuntu 22.04+, CentOS 7+)
- **macOS** 10.15+

Prerequisites

1. [.NET 8 SDK](#)
 2. Terminal/Command Prompt access
 3. 50MB disk space
-

Installation & Execution

Command Line (All Platforms)

Bash

Clone repository

```
git clone https://github.com/your-repo/auto-driving-car.git  
cd AutoDrivingCar
```

Build and run
dotnet build

For Windows
dotnet AutoDrivingCar.dll

For self-contained apps
.\AutoDrivingCar.exe

Run tests
dotnet test

Visual Studio (Windows)

1. Open `AutoDrivingCar.sln`
 2. Set startup project to `AutoDrivingCar.Console`
 3. Build > Build Solution (Ctrl+Shift+B)
 4. Debug > Start Without Debugging (Ctrl+F5)
-

Application Flow

graph TD

```
A[Start] --> B[Create Field]
B --> C{Add Car?}
C -->|Yes| D[Input Car Details]
C -->|No| E{Run Simulation?}
D --> C
E -->|Yes| F[Process Commands]
F --> G[Detect Collisions]
G --> H[Display Results]
H --> I{Restart?}
```

Code Structure

Project Layout

AutoDrivingCar/

```
|— Core/           # Domain models
| |— Car.cs        # Movement/Rotation logic
| |— Field.cs      # Boundary validation
| |— Simulation.cs # Command processor
|— Interfaces/     # Abstraction contracts
| |— ICar.cs
| |— IField.cs
| |— ISimulation.cs
|— Console/        # CLI implementation
| |— Program.cs    # User interaction
```

└─ Tests/ # Unit tests
 └─ CarTests.cs
 └─ SimulationTests.cs

Testing Strategy

Test Coverage

1. **Car Movement**
 - Valid/invalid moves
 - Rotation sequences
 - Command processing
2. **Field Validation**
 - Boundary checks
 - Invalid position detection
3. **Simulation**
 - Collision scenarios
 - Multi-car interactions
 - Command step execution

Example Test

[Test]

```
public void ThreeCarCollision_AllRegisterImpact()
{
    var field = new Field(10, 10);
    var simulation = new Simulation(field);

    simulation.AddCar(new Car("A", 1, 1, Direction.East, "FF"));
    simulation.AddCar(new Car("B", 3, 1, Direction.West, "FF"));
    simulation.AddCar(new Car("C", 2, 1, Direction.North, "F"));

    simulation.Run();

    Assert.That(simulation.GetCars().All(c => c.Collision != null));
}
```

Readability & Maintenance

Code Quality

1. Naming Conventions

- Clear, descriptive names (`ProcessCommand`, `RegisterCollision`)
- Consistent casing (PascalCase for methods, camelCase for locals)

2. Documentation

- XML comments for public members
- Error code documentation

3. Encapsulation

- Private fields/methods for internal logic
- Immutable DTOs for data transfer

Maintenance Features

1. Null Safety

- Nullable reference types enabled
- Guard clauses for null checks

2. Error Handling

- Custom exceptions for domain errors
- Input validation at boundaries

3. Extensibility

- Interface-based dependencies
 - Protected virtual methods for overriding
-

Key Execution Scenarios

Sample Input/Output

Welcome to Auto Driving Car Simulation!

Field size: 10x10

Car A: (1,2) N, Commands: FFRFFFRRL

Car B: (7,8) W, Commands: FFLFFFFF

Simulation Results:

A collided with B at (5,4) step 7

B collided with A at (5,4) step 7

The design emphasizes testability and maintainability while adhering to .NET best practices. The interface-based architecture allows easy swapping of components (e.g., different collision detection algorithms) without impacting core functionality.