**Title: System Implementation – Driverless Car**

**README file**

## Description

This project implements software to support the operation of a driverless car using OOP principles in Python. It supports three key operations, allowing user interaction through a frontend interface, while the backend simulates data generation and collection. Other required functions like sensors, cameras, and control panels, are assumed to be imported from other subsystems. The corresponding UML diagrams modified in response to feedback, are presented in the **Appendix 9-13** for reference.

## Key Operations

- **Lane Detection:** Continuously monitors and corrects the car's position within the lane.
- **Obstacle Detection:** Detects obstacles in the path and takes appropriate actions.
- **Emergency Brake:** Activates emergency braking when a critical obstacle is detected.

## Module and Class Overview

To better organize the code, classes and functions are defined in separate modules and imported into the main program as needed.

| **Appendix** | **Module** | **Class** | **Description** |
|---|---|---|---|
| **1** | vehicle.py | Vehicle | Base class for generic vehicle with basic functionalities like starting and stopping engine. |
| **2** | driverless_car.py | DriverlessCar | Inherits from Vehicle and includes methods for moving, turning, and braking. Integrates lane detection, obstacle detection, and emergency braking. |
| **3** | lane_detection.py | LaneDetect | Handles lane deviation detection and correction. |
| **4** | obstacle_detection.py | ObstacleDetect | Manages obstacle detection and initiates slowing down or emergency braking. |
| **5** | emergency_brake.py | EmergencyBrake | Activates emergency brake when needed. |
| **6** | backend.py | Backend | Collects and stores action history. |

|  |  |  |  |
|---|---|---|---|
| **7** | frontend.py | Frontend | Provides text-based interface for interacting with the car and viewing the history. |
| **8** | test.py |  | Contains automated tests to validate the functionality of the driverless car system. |

## Data Structures Used

- **List:** Used in the Backend class to store the history of actions. Lists allow efficient appending, suitable for maintaining history log.

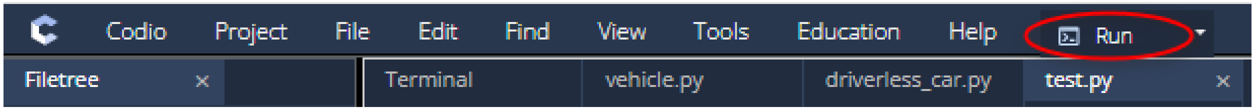
## OOP Features Used

- **Inheritance:** The DriverlessCar class inherits from the Vehicle class, allowing it to use and extend the functionalities of the base class.
- **Abstraction:** The system abstracts complex behaviors into simplified interfaces and method calls.
- **Polymorphism:** Methods like move, turn, and brake in the DriverlessCar class exhibit polymorphic behavior, responding differently based on the context.

### Execution Instructions

This software is designed to support the operation of a driverless car. It can be executed with additional support from other functions, such as sensors, cameras, and control panels. The Test Module (test.py) is executed to test the driverless car system and ensure that all components work correctly and interact as expected:

1. Ensure Python 3.x is installed on your IDE.
2. Ensure all modules are in the same directory.
3. Execute the test.py script to perform a series of automated tests validating the system's functionality.



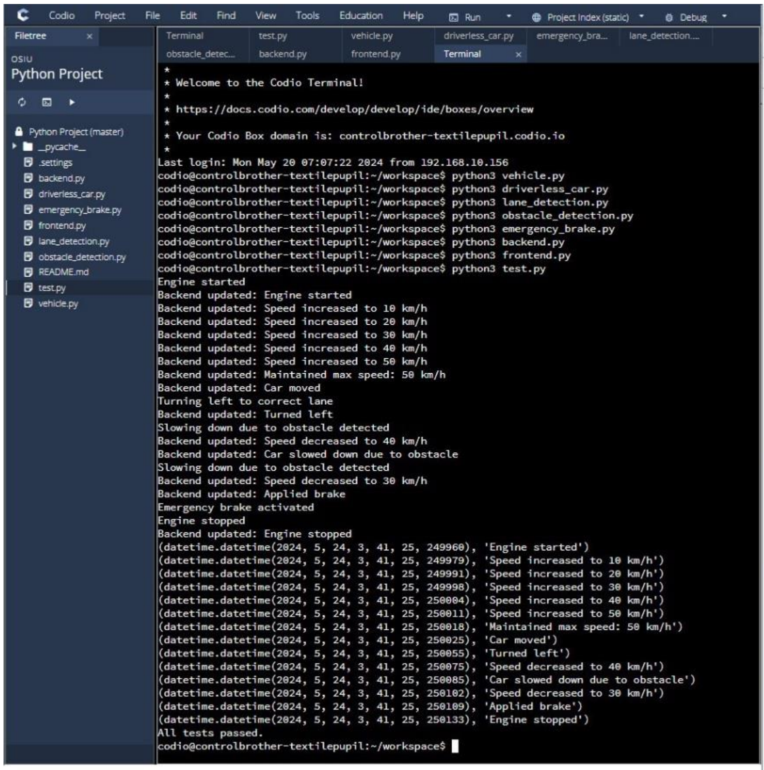
### Automated Test Result

The test.py imports all modules and uses Python’s assert statements to achieve automated testing. The tests simulate various actions of the driverless car to ensure all components interact correctly.

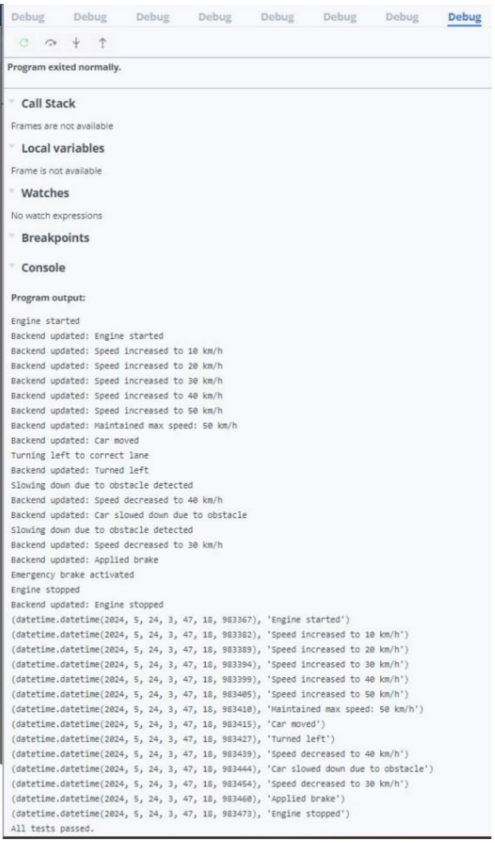
1. **Start car:** Verifies the engine starts and the speed is initialized to zero.
2. **Move car forward:** Ensures the car reaches and maintains the maximum speed.
3. **Simulate lane deviation and correction:** Tests the detection and correction of lane deviations.
4. **Simulate obstacle detection and slowing down:** Verifies the car slows down when detecting obstacles at a safe distance.
5. **Simulate further obstacle detection and emergency brake:** Ensures the car applies the emergency brake if the obstacle is too close.

6. **Stop car:** Verifies the car stops and the engine is turned off.
7. **View history:** Retrieves and displays the history of all actions taken by the car.

All modules exited normally and below is the expected output of the test:



```
*
* Welcome to the Codio Terminal!
*
* https://docs.codio.com/develop/develop/ide/boxes/overview
*
* Your Codio Box domain is: controlbrother-textilepupil.codio.io
*
Last login: Mon May 20 07:07:22 2024 from 192.168.10.156
codio@controlbrother-textilepupil:~/workspace$ python3 vehicle.py
codio@controlbrother-textilepupil:~/workspace$ python3 driverless_car.py
codio@controlbrother-textilepupil:~/workspace$ python3 lane_detection.py
codio@controlbrother-textilepupil:~/workspace$ python3 obstacle_detection.py
codio@controlbrother-textilepupil:~/workspace$ python3 emergency_brake.py
codio@controlbrother-textilepupil:~/workspace$ python3 backend.py
codio@controlbrother-textilepupil:~/workspace$ python3 frontend.py
codio@controlbrother-textilepupil:~/workspace$ python3 test.py
Engine started
Backend updated: Engine started
Backend updated: Speed increased to 10 km/h
Backend updated: Speed increased to 20 km/h
Backend updated: Speed increased to 30 km/h
Backend updated: Speed increased to 40 km/h
Backend updated: Speed increased to 50 km/h
Backend updated: Maintained max speed: 50 km/h
Backend updated: Car moved
Turning left to correct lane
Backend updated: Turned left
Slowing down due to obstacle detected
Backend updated: Speed decreased to 40 km/h
Backend updated: Car slowed down due to obstacle
Slowing down due to obstacle detected
Backend updated: Speed decreased to 30 km/h
Backend updated: Applied brake
Emergency brake activated
Engine stopped
Backend updated: Engine stopped
(datetime.datetime(2024, 5, 24, 3, 41, 25, 249960), 'Engine started')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 249979), 'Speed increased to 10 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 249991), 'Speed increased to 20 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 249998), 'Speed increased to 30 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250004), 'Speed increased to 40 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250011), 'Speed increased to 50 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250018), 'Maintained max speed: 50 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250025), 'Car moved')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250055), 'Turned left')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250075), 'Speed decreased to 40 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250085), 'Car slowed down due to obstacle')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250102), 'Speed decreased to 30 km/h')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250109), 'Applied brake')
(datetime.datetime(2024, 5, 24, 3, 41, 25, 250133), 'Engine stopped')
All tests passed.
codio@controlbrother-textilepupil:~/workspace$
```



Program exited normally.

Frames are not available

Frame is not available

No watch expressions

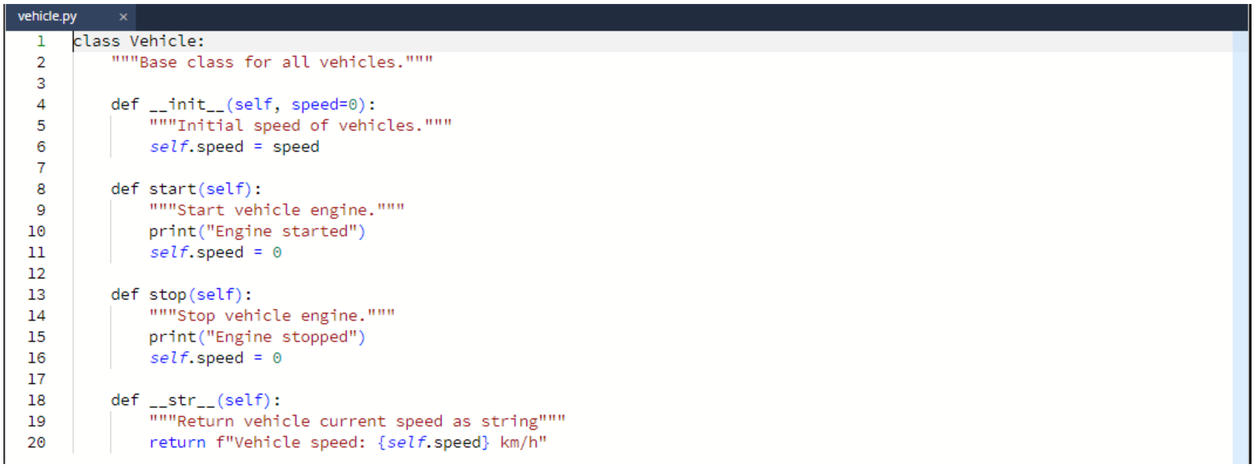
Program output:

```
Engine started
Backend updated: Engine started
Backend updated: Speed increased to 10 km/h
Backend updated: Speed increased to 20 km/h
Backend updated: Speed increased to 30 km/h
Backend updated: Speed increased to 40 km/h
Backend updated: Speed increased to 50 km/h
Backend updated: Maintained max speed: 50 km/h
Backend updated: Car moved
Turning left to correct lane
Backend updated: Turned left
Slowing down due to obstacle detected
Backend updated: Speed decreased to 40 km/h
Backend updated: Car slowed down due to obstacle
Slowing down due to obstacle detected
Backend updated: Speed decreased to 30 km/h
Backend updated: Applied brake
Emergency brake activated
Engine stopped
Backend updated: Engine stopped
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983367), 'Engine started')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983382), 'Speed increased to 10 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983389), 'Speed increased to 20 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983394), 'Speed increased to 30 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983399), 'Speed increased to 40 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983405), 'Speed increased to 50 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983410), 'Maintained max speed: 50 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983415), 'Car moved')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983427), 'Turned left')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983439), 'Speed decreased to 40 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983444), 'Car slowed down due to obstacle')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983454), 'Speed decreased to 30 km/h')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983460), 'Applied brake')
(datetime.datetime(2024, 5, 24, 3, 47, 18, 983473), 'Engine stopped')
All tests passed.
```

## Conclusion

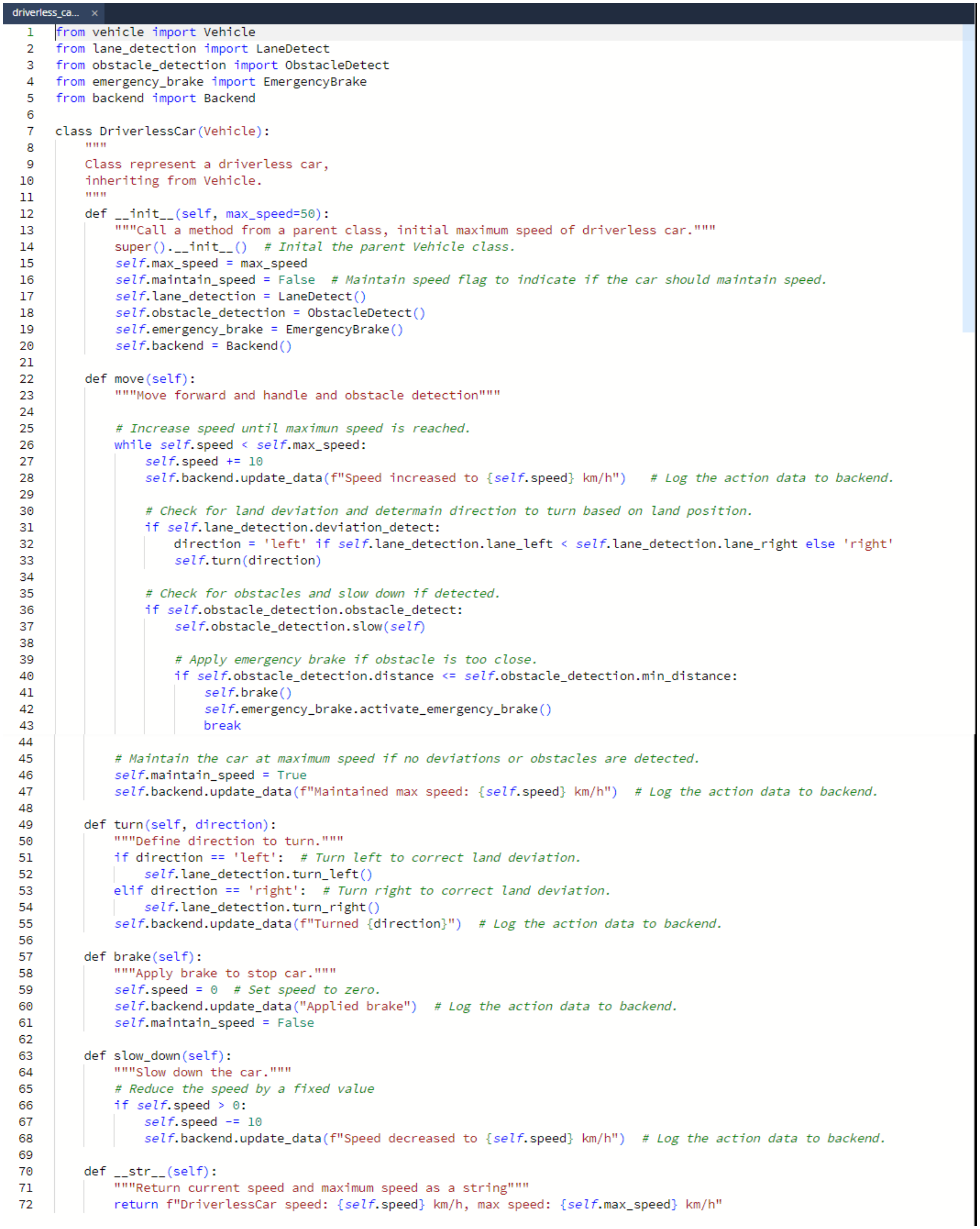
This project demonstrates implementation of software that supports the operation of driverless car using OOP principles. The detailed testing and validation processes ensure that each component performs as intended, contributing to the overall reliability and safety of the system. By modularizing the system into distinct components, the project ensures reusability, maintainability, and scalability.

## Appendix 1 vehicle.py



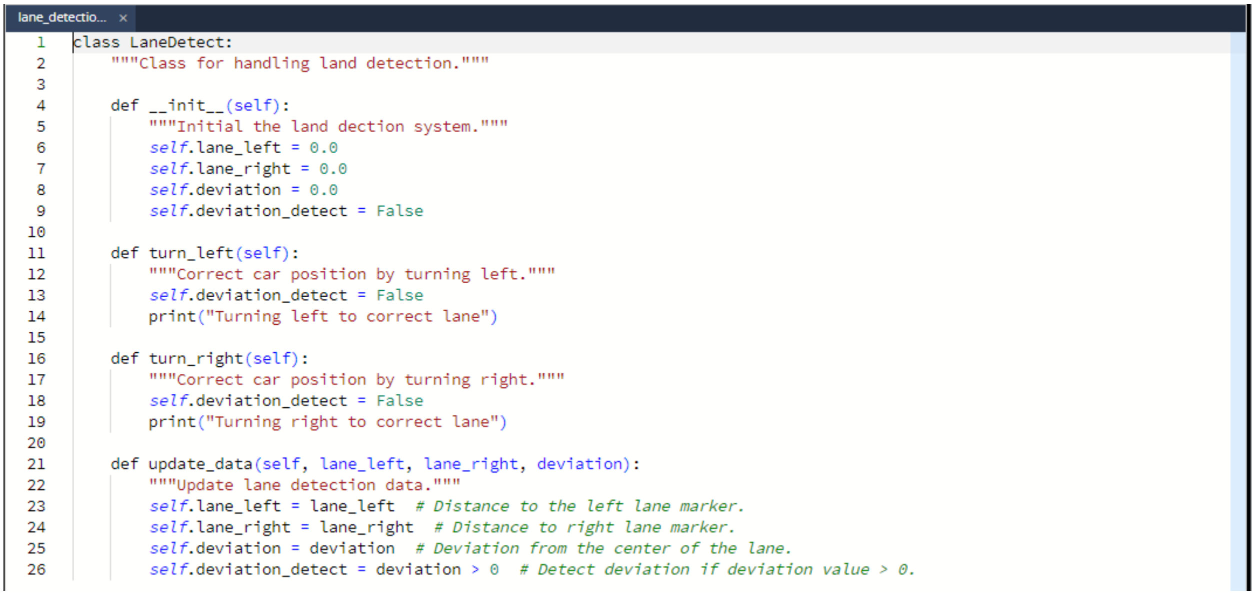
```python
class Vehicle:
    """Base class for all vehicles."""

    def __init__(self, speed=0):
        """Initial speed of vehicles."""
        self.speed = speed

    def start(self):
        """Start vehicle engine."""
        print("Engine started")
        self.speed = 0

    def stop(self):
        """Stop vehicle engine."""
        print("Engine stopped")
        self.speed = 0

    def __str__(self):
        """Return vehicle current speed as string"""
        return f"Vehicle speed: {self.speed} km/h"
```

## Appendix 2 driverless_car.py



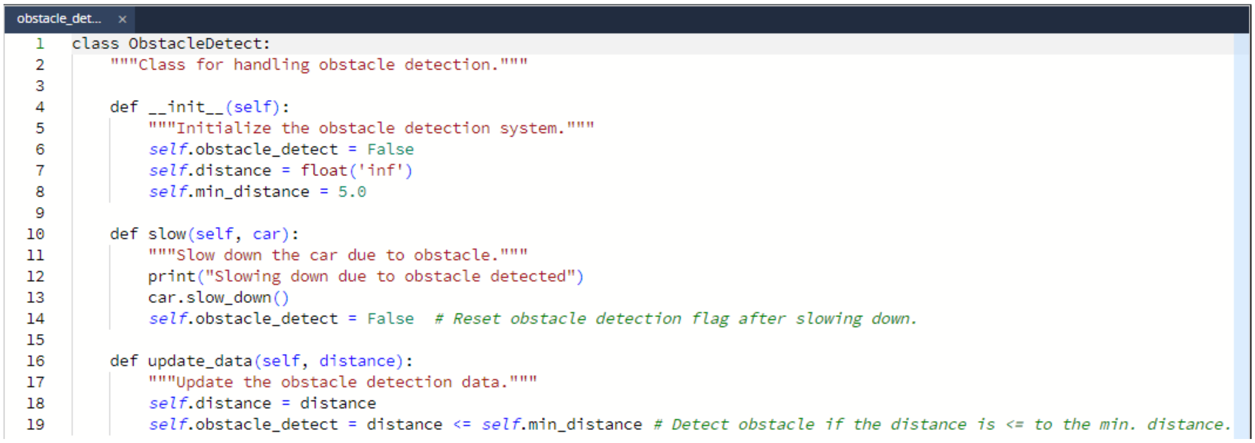
```python
from vehicle import Vehicle
from lane_detection import LaneDetect
from obstacle_detection import ObstacleDetect
from emergency_brake import EmergencyBrake
from backend import Backend

class DriverlessCar(Vehicle):
    """
    Class represent a driverless car,
    inheriting from Vehicle.
    """
    def __init__(self, max_speed=50):
        """Call a method from a parent class, initial maximum speed of driverless car."""
        super().__init__()  # Inital the parent Vehicle class.
        self.max_speed = max_speed
        self.maintain_speed = False  # Maintain speed flag to indicate if the car should maintain speed.
        self.lane_detection = LaneDetect()
        self.obstacle_detection = ObstacleDetect()
        self.emergency_brake = EmergencyBrake()
        self.backend = Backend()

    def move(self):
        """Move forward and handle and obstacle detection"""

        # Increase speed until maximun speed is reached.
        while self.speed < self.max_speed:
            self.speed += 10
            self.backend.update_data(f"Speed increased to {self.speed} km/h")   # Log the action data to backend.

            # Check for land deviation and determain direction to turn based on land position.
            if self.lane_detection.deviation_detect:
                direction = 'left' if self.lane_detection.lane_left < self.lane_detection.lane_right else 'right'
                self.turn(direction)

            # Check for obstacles and slow down if detected.
            if self.obstacle_detection.obstacle_detect:
                self.obstacle_detection.slow(self)

                # Apply emergency brake if obstacle is too close.
                if self.obstacle_detection.distance <= self.obstacle_detection.min_distance:
                    self.brake()
                    self.emergency_brake.activate_emergency_brake()
                    break

        # Maintain the car at maximum speed if no deviations or obstacles are detected.
        self.maintain_speed = True
        self.backend.update_data(f"Maintained max speed: {self.speed} km/h")  # Log the action data to backend.

    def turn(self, direction):
        """Define direction to turn."""
        if direction == 'left':  # Turn left to correct land deviation.
            self.lane_detection.turn_left()
        elif direction == 'right':  # Turn right to correct land deviation.
            self.lane_detection.turn_right()
        self.backend.update_data(f"Turned {direction}")  # Log the action data to backend.

    def brake(self):
        """Apply brake to stop car."""
        self.speed = 0  # Set speed to zero.
        self.backend.update_data("Applied brake")  # Log the action data to backend.
        self.maintain_speed = False

    def slow_down(self):
        """Slow down the car."""
        # Reduce the speed by a fixed value
        if self.speed > 0:
            self.speed -= 10
            self.backend.update_data(f"Speed decreased to {self.speed} km/h")  # Log the action data to backend.

    def __str__(self):
        """Return current speed and maximum speed as a string"""
        return f"DriverlessCar speed: {self.speed} km/h, max speed: {self.max_speed} km/h"
```

## Appendix 3 lane_detection.py



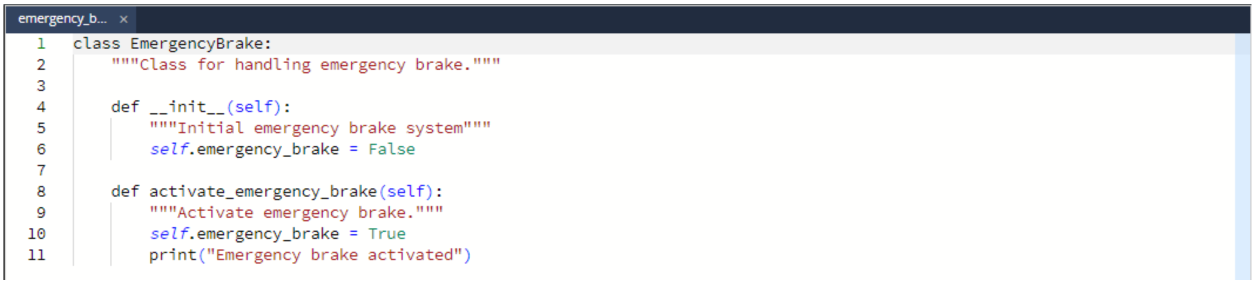
```python
class LaneDetect:
    """Class for handling land detection."""

    def __init__(self):
        """Initial the land dection system."""
        self.lane_left = 0.0
        self.lane_right = 0.0
        self.deviation = 0.0
        self.deviation_detect = False

    def turn_left(self):
        """Correct car position by turning left."""
        self.deviation_detect = False
        print("Turning left to correct lane")

    def turn_right(self):
        """Correct car position by turning right."""
        self.deviation_detect = False
        print("Turning right to correct lane")

    def update_data(self, lane_left, lane_right, deviation):
        """Update lane detection data."""
        self.lane_left = lane_left  # Distance to the left lane marker.
        self.lane_right = lane_right  # Distance to right lane marker.
        self.deviation = deviation  # Deviation from the center of the lane.
        self.deviation_detect = deviation > 0  # Detect deviation if deviation value > 0.
```

## Appendix 4 obstacle_detection.py



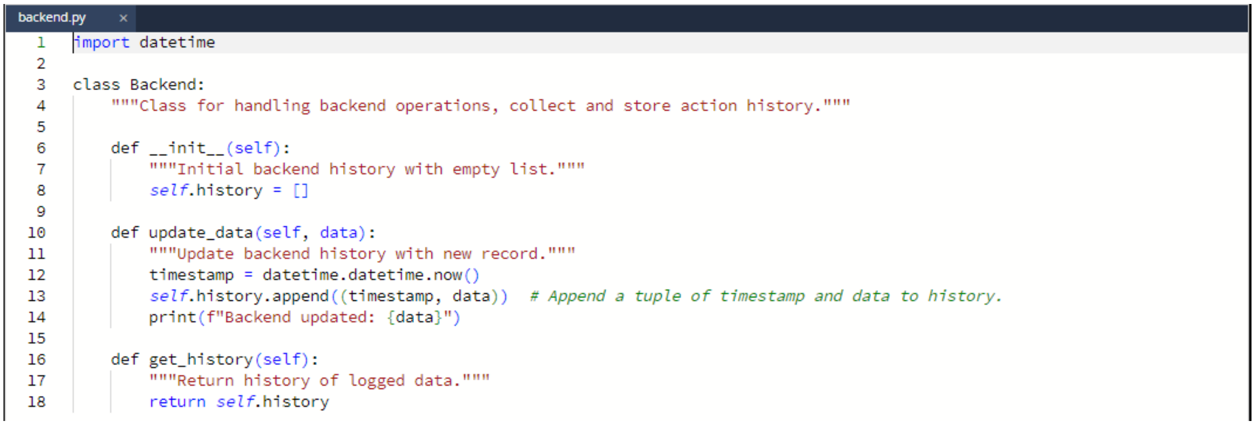
```python
class ObstacleDetect:
    """Class for handling obstacle detection."""

    def __init__(self):
        """Initialize the obstacle detection system."""
        self.obstacle_detect = False
        self.distance = float('inf')
        self.min_distance = 5.0

    def slow(self, car):
        """Slow down the car due to obstacle."""
        print("Slowing down due to obstacle detected")
        car.slow_down()
        self.obstacle_detect = False  # Reset obstacle detection flag after slowing down.

    def update_data(self, distance):
        """Update the obstacle detection data."""
        self.distance = distance
        self.obstacle_detect = distance <= self.min_distance # Detect obstacle if the distance is <= to the min. distance.
```

## Appendix 5 emergency_brake.py



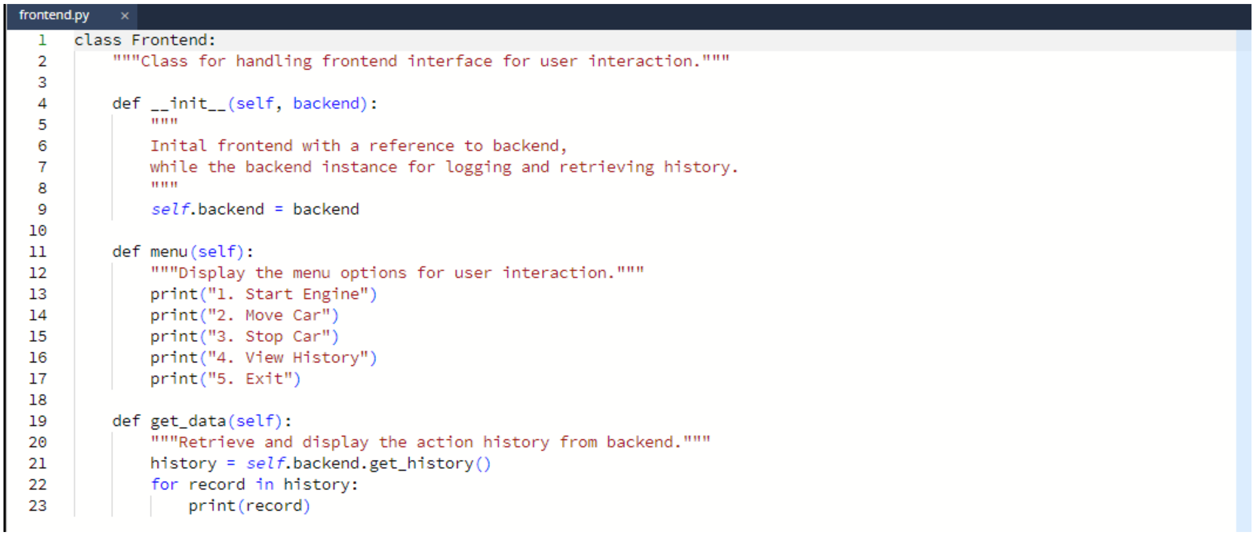
```python
class EmergencyBrake:
    """Class for handling emergency brake."""

    def __init__(self):
        """Initial emergency brake system"""
        self.emergency_brake = False

    def activate_emergency_brake(self):
        """Activate emergency brake."""
        self.emergency_brake = True
        print("Emergency brake activated")
```

## Appendix 6 backend.py



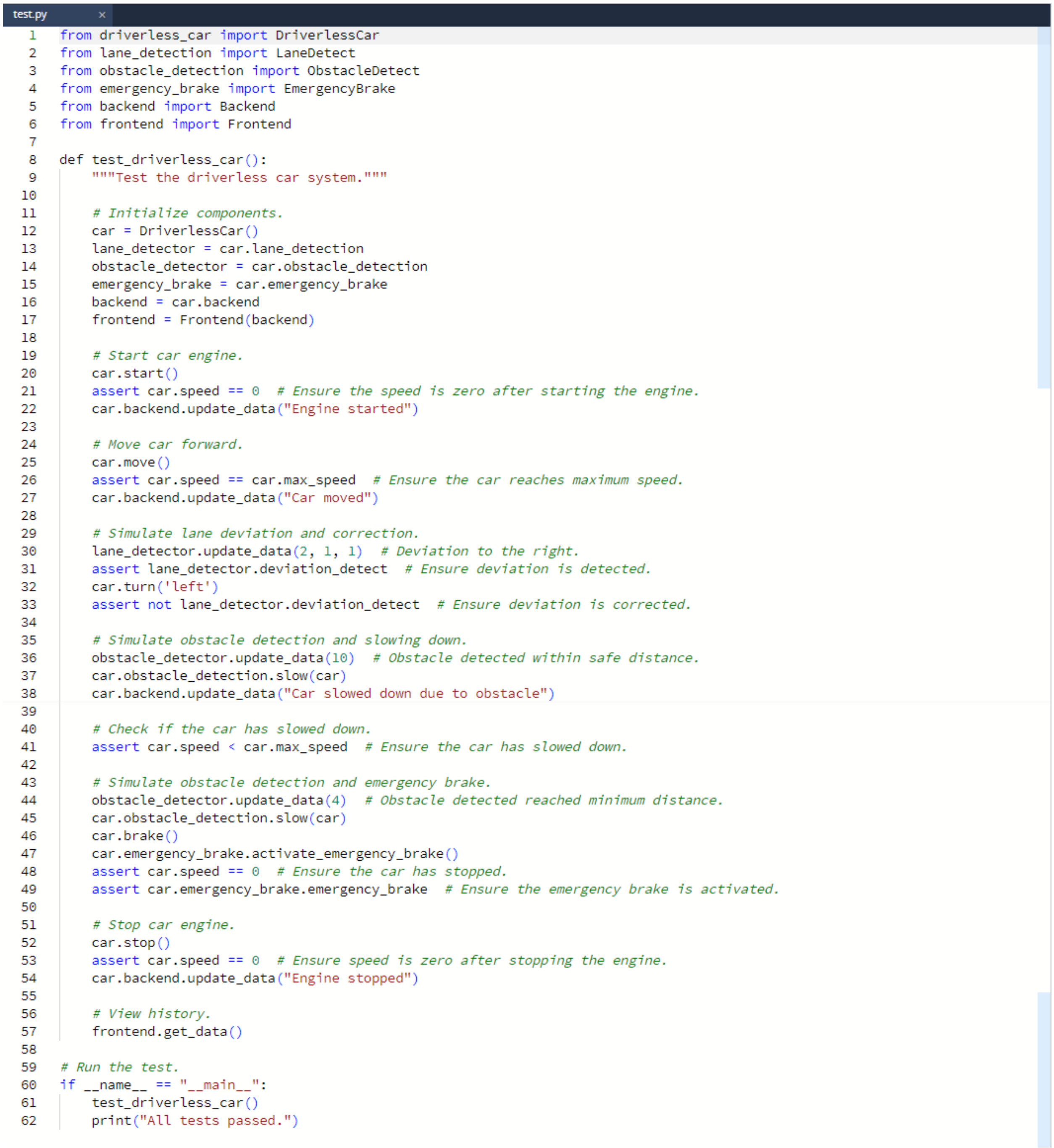
```python
import datetime

class Backend:
    """Class for handling backend operations, collect and store action history."""

    def __init__(self):
        """Initial backend history with empty list."""
        self.history = []

    def update_data(self, data):
        """Update backend history with new record."""
        timestamp = datetime.datetime.now()
        self.history.append((timestamp, data))  # Append a tuple of timestamp and data to history.
        print(f"Backend updated: {data}")

    def get_history(self):
        """Return history of logged data."""
        return self.history
```

## Appendix 7 frontend.py



```python
class Frontend:
    """Class for handling frontend interface for user interaction."""

    def __init__(self, backend):
        """
        Inital frontend with a reference to backend,
        while the backend instance for logging and retrieving history.
        """
        self.backend = backend

    def menu(self):
        """Display the menu options for user interaction."""
        print("1. Start Engine")
        print("2. Move Car")
        print("3. Stop Car")
        print("4. View History")
        print("5. Exit")

    def get_data(self):
        """Retrieve and display the action history from backend."""
        history = self.backend.get_history()
        for record in history:
            print(record)
```

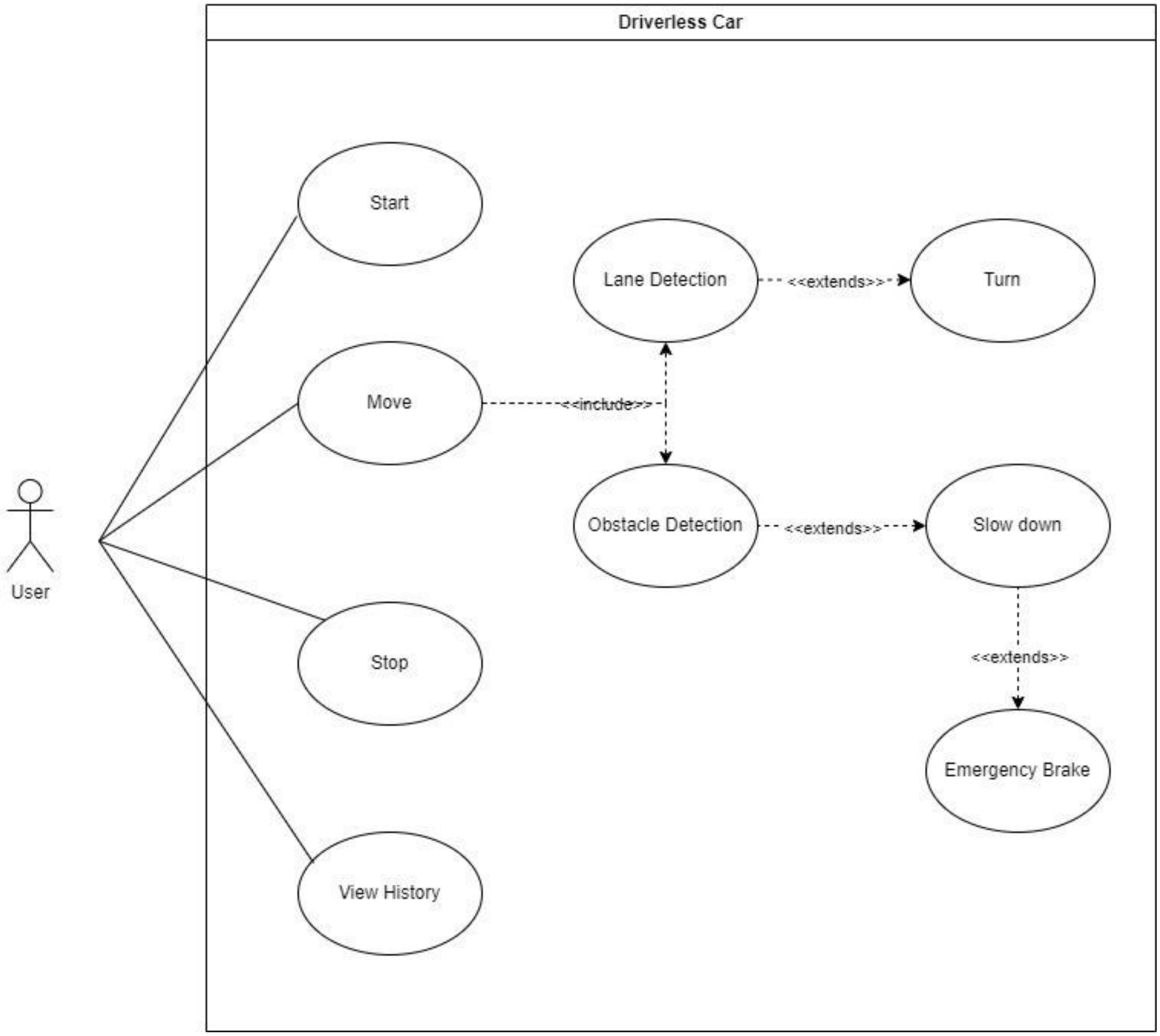
## Appendix 8 test.py



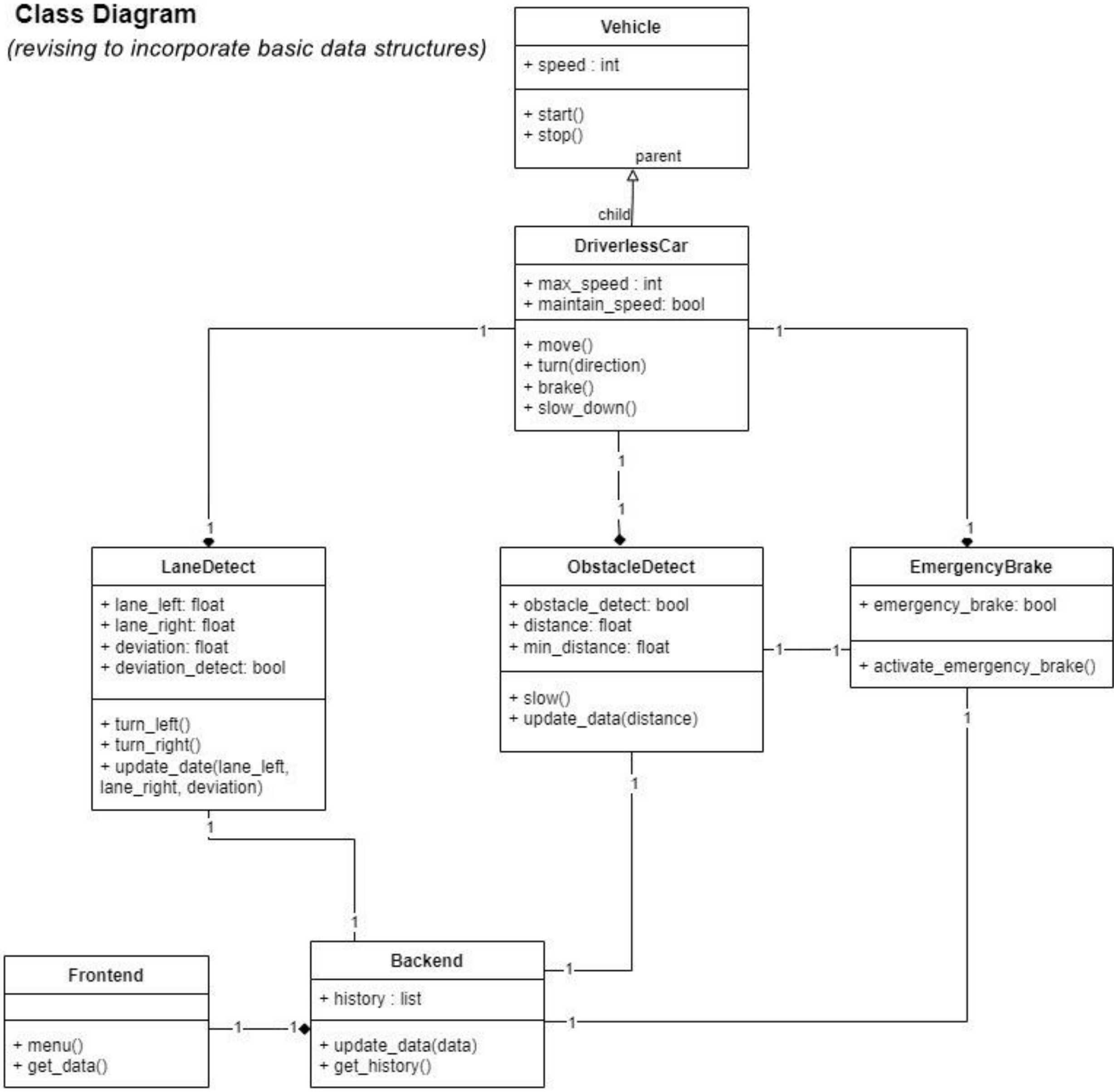
```python
from driverless_car import DriverlessCar
from lane_detection import LaneDetect
from obstacle_detection import ObstacleDetect
from emergency_brake import EmergencyBrake
from backend import Backend
from frontend import Frontend

def test_driverless_car():
    """Test the driverless car system."""

    # Initialize components.
    car = DriverlessCar()
    lane_detector = car.lane_detection
    obstacle_detector = car.obstacle_detection
    emergency_brake = car.emergency_brake
    backend = car.backend
    frontend = Frontend(backend)

    # Start car engine.
    car.start()
    assert car.speed == 0  # Ensure the speed is zero after starting the engine.
    car.backend.update_data("Engine started")

    # Move car forward.
    car.move()
    assert car.speed == car.max_speed  # Ensure the car reaches maximum speed.
    car.backend.update_data("Car moved")

    # Simulate lane deviation and correction.
    lane_detector.update_data(2, 1, 1)  # Deviation to the right.
    assert lane_detector.deviation_detect  # Ensure deviation is detected.
    car.turn('left')
    assert not lane_detector.deviation_detect  # Ensure deviation is corrected.

    # Simulate obstacle detection and slowing down.
    obstacle_detector.update_data(10)  # Obstacle detected within safe distance.
    car.obstacle_detection.slow(car)
    car.backend.update_data("Car slowed down due to obstacle")

    # Check if the car has slowed down.
    assert car.speed < car.max_speed  # Ensure the car has slowed down.

    # Simulate obstacle detection and emergency brake.
    obstacle_detector.update_data(4)  # Obstacle detected reached minimum distance.
    car.obstacle_detection.slow(car)
    car.brake()
    car.emergency_brake.activate_emergency_brake()
    assert car.speed == 0  # Ensure the car has stopped.
    assert car.emergency_brake.emergency_brake  # Ensure the emergency brake is activated.

    # Stop car engine.
    car.stop()
    assert car.speed == 0  # Ensure speed is zero after stopping the engine.
    car.backend.update_data("Engine stopped")

    # View history.
    frontend.get_data()

# Run the test.
if __name__ == "__main__":
    test_driverless_car()
    print("All tests passed.")
```

## Appendix 9

**Use Case Diagram** *(Revising to better reflect the intended functionalities)*



## Appendix 10



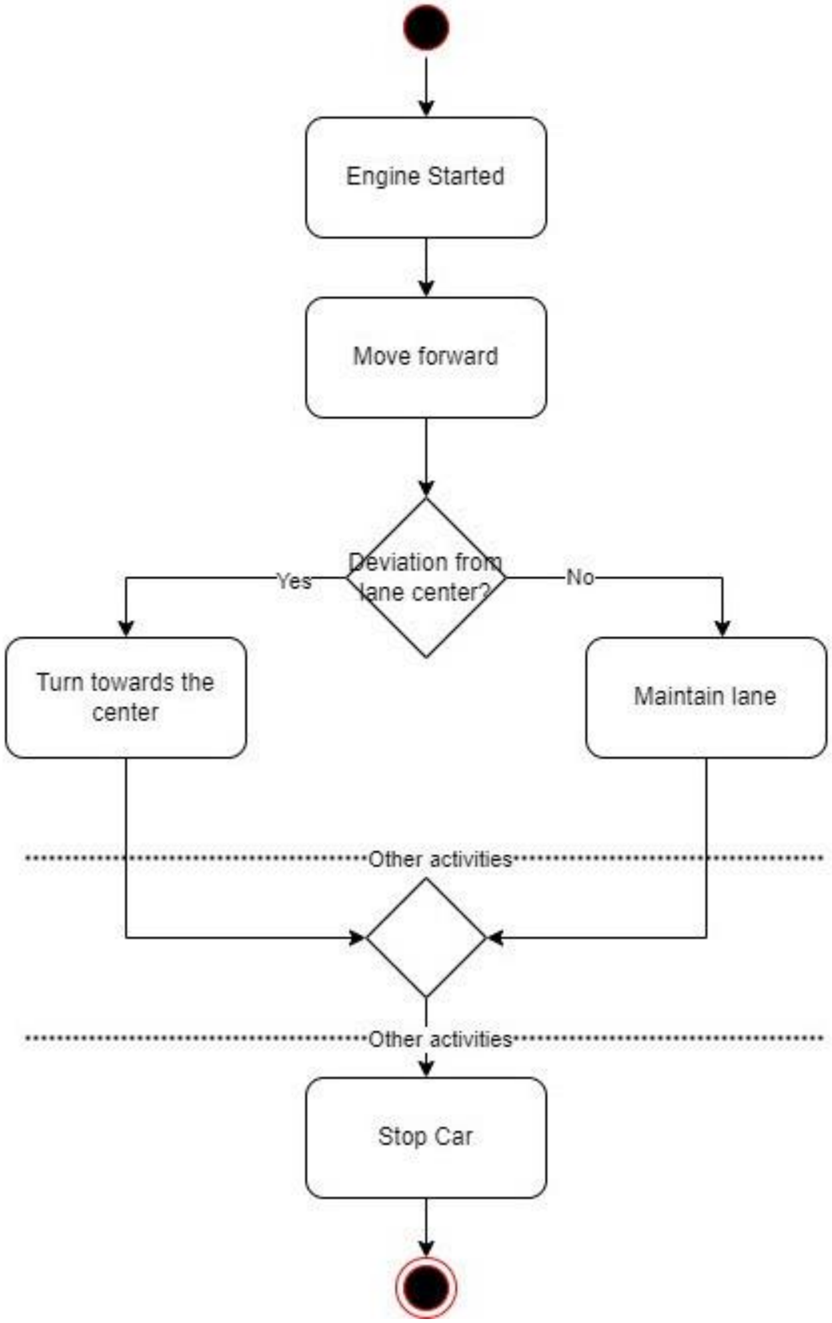
**Class Diagram**

*(revising to incorporate basic data structures)*

## Appendix 11

**Activity Diagram (Land Detection)**

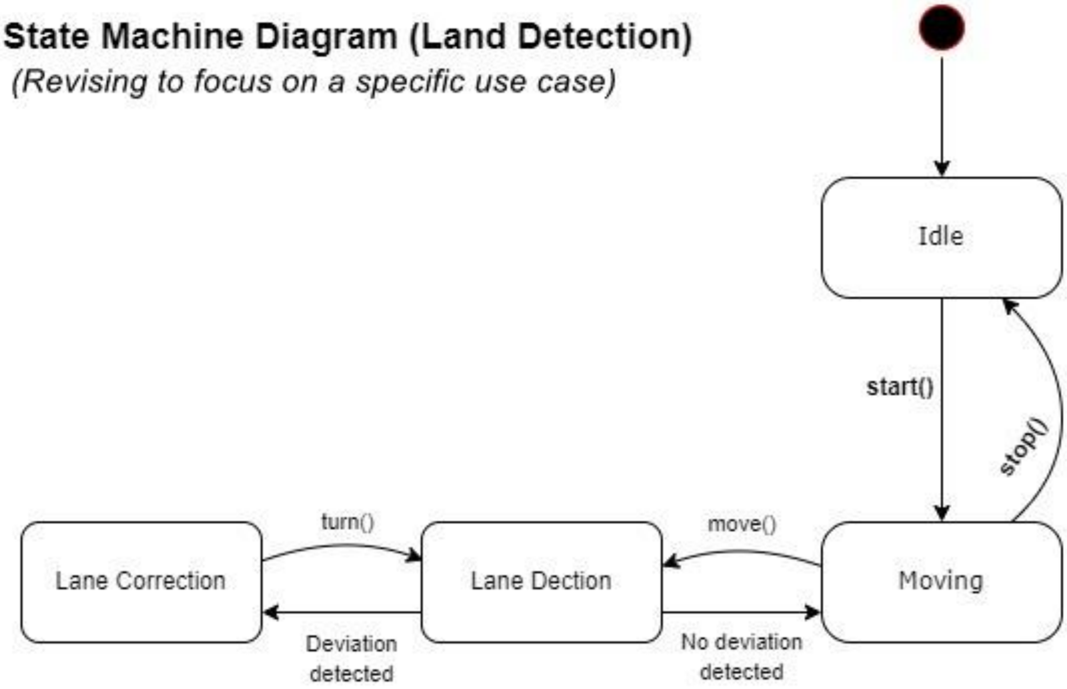
*(Revising to focus on a specific use case)*



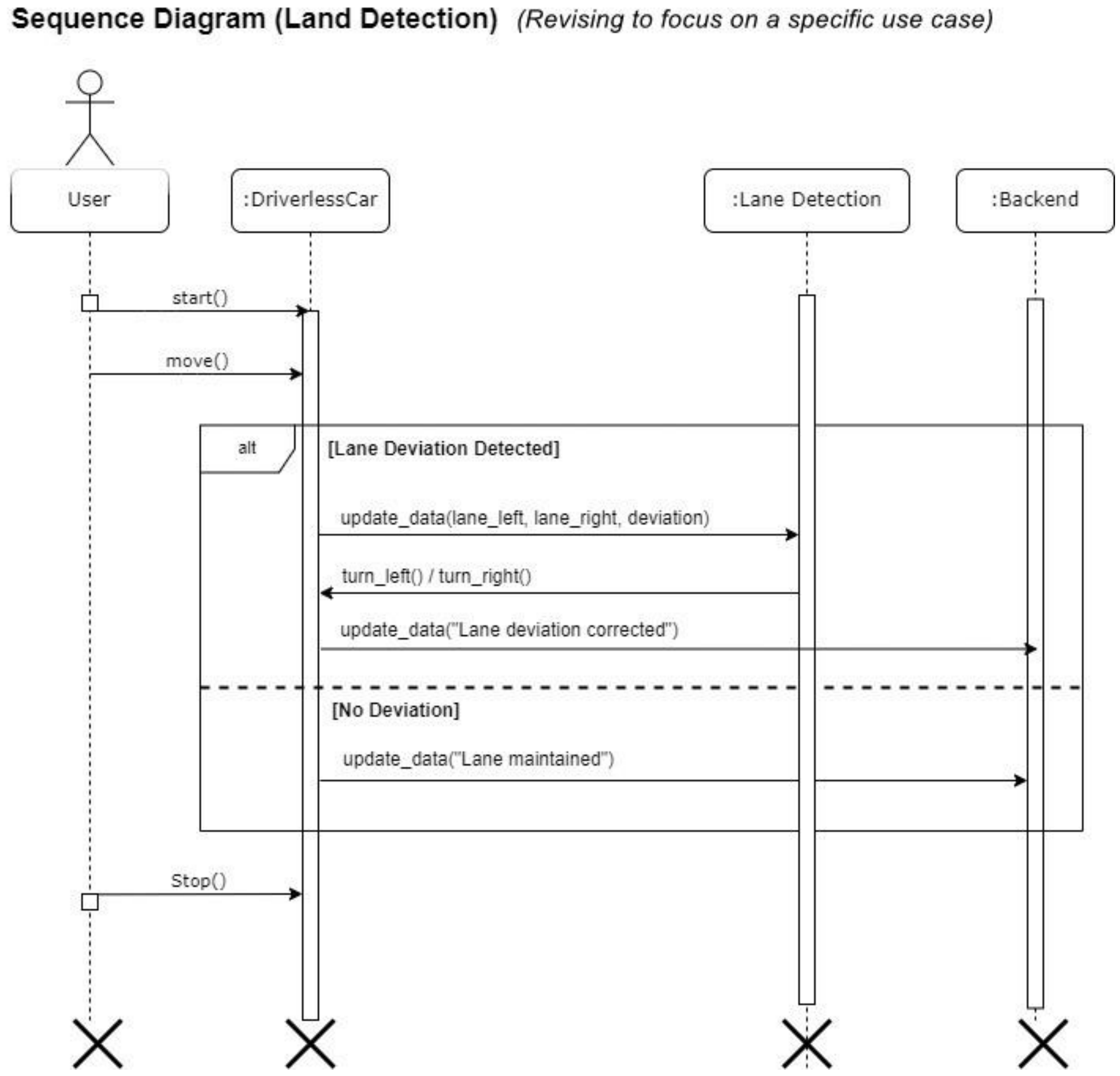
## Appendix 12

**State Machine Diagram (Land Detection)**

*(Revising to focus on a specific use case)*



## Appendix 13


Sequence Diagram (Land Detection)
(Revising to focus on a specific use case)
User
:DriverlessCar
:Lane Detection
:Backend
start()
move()
alt
[Lane Deviation Detected]
update_data(lane_left, lane_right, deviation)
turn_left() / turn_right()
update_data("Lane deviation corrected")
[No Deviation]
update_data("Lane maintained")
Stop()