Software Engineering Projekt

Crazy Car

Author...

22. März 2020

Inhaltsverzeichnis

1 Vision

Nach Projektabschluss ist das CrazyCar über ein Programm am PC mithilfe der Pfeiltasten der Tastatur fernsteuerbar.

Unterstützend wird am PC auch ein Videostream der am Fahrzeug montierten Kamera angezeigt, sodass das Fahrzeug mithilfe des Kamerabildes gesteuert werden kann. Die Übertragung der Daten zwischen dem Fahrzeug und dem PC erfolgt via WLAN.

2 Epic und User Stories

Im folgenden Teil werden User Stories sowie der Epic unseres Projekts dargelegt. Dabei sollen diese die wesentlichen Anforderungen an das Softwareprojekt festlegen, um den Nutzern einen optimalen Funktionsumfang zu bieten.

2.1 Epic

Nr. 1	**Remote Control CrazyCar**
Das sichere Steuern des CrazyCars ohne Sichtkontakt.	

2.2 User Stories

Nr. 1-1	**First Person Camera**
Als Benutzer bzw. Student möchte ich eine verzögerungsfreie Bildübertragung damit ich dank der Kamera mögliche Hindernisse sehe und darauf in meinem Fahrverhalten reagieren kann.	

Nr. 1-2	**User Friendly**

Als Benutzer (Student, Kind)

möchte ich ein einfaches User Interface

damit ich das CrazyCar einfach steuern kann, keine Commandline Befehle notwendig sind und ich mich ohne Fachwissen austoben kann.

Nr. 1-3	**Safety First**

Als zahlender Besitzer des Autos (Lehrstuhl – CIT)

möchte ich einen Notstop

damit vor Hindernissen rechtzeitig gestoppt wird und damit in weiterer Folge das Auto nicht gleich kaputt ist.

3 Test cases

Im folgenden Teil werden Testfälle für die Entwicklung der Software festgelegt, welche die Basisfunktionalitäten abdecken.

3.1 ConnectionTest

```
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;

public class ConnectionTest {

    private Car car;
    private String url;

    @BeforeEach
    public void setUp() {
        car = new Car();
        url = "carIp";
    }

    @AfterEach
    public void tearDown() {
        url = null;
        car = null;
    }

    @Test
    public void testConnectionEstablished() {
        assertNotNull(car.getStatus());
    }

    @Test
    public void testCarResponse() {
        assertNotNull(car.setSpeed(0.0));
    }

    @Test
    public void testWrongUrl() {
        url = "wrongUrl";
        assertNull(car.getStatus());
    }
}
```

3.2 CarTest

```
import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.*;

public class CarTest {
    private Car car;
    private static Connection connection;

    @BeforeAll
    public static void setup() {
        connection = new Connection();
    }

    @BeforeEach
    public void setUp() {
        car = new Car();
        car.setConnection(connection);
    }

    @Test
    public void testSpeedChange() {
        double[] testArray = {-1.0,-0.5,0,0.5,1.0};
        double[] resultArray = new double[testArray.length];
        for (int i = 0; i<testArray.length;i++)
                resultArray[i] = car.setSpeed(testArray[i]);
        assertArrayEquals(testArray,resultArray);
    }

    @Test
    public void testSteerLeft() {
        assertSame(-1.0,car.steer(-1.0));
    }

    @Test
    public void testSteerRight() {
        assertSame(1.0,car.steer(1.0));
    }

    @AfterEach
    public void tearDonw() {
        car = null;
    }


}
```