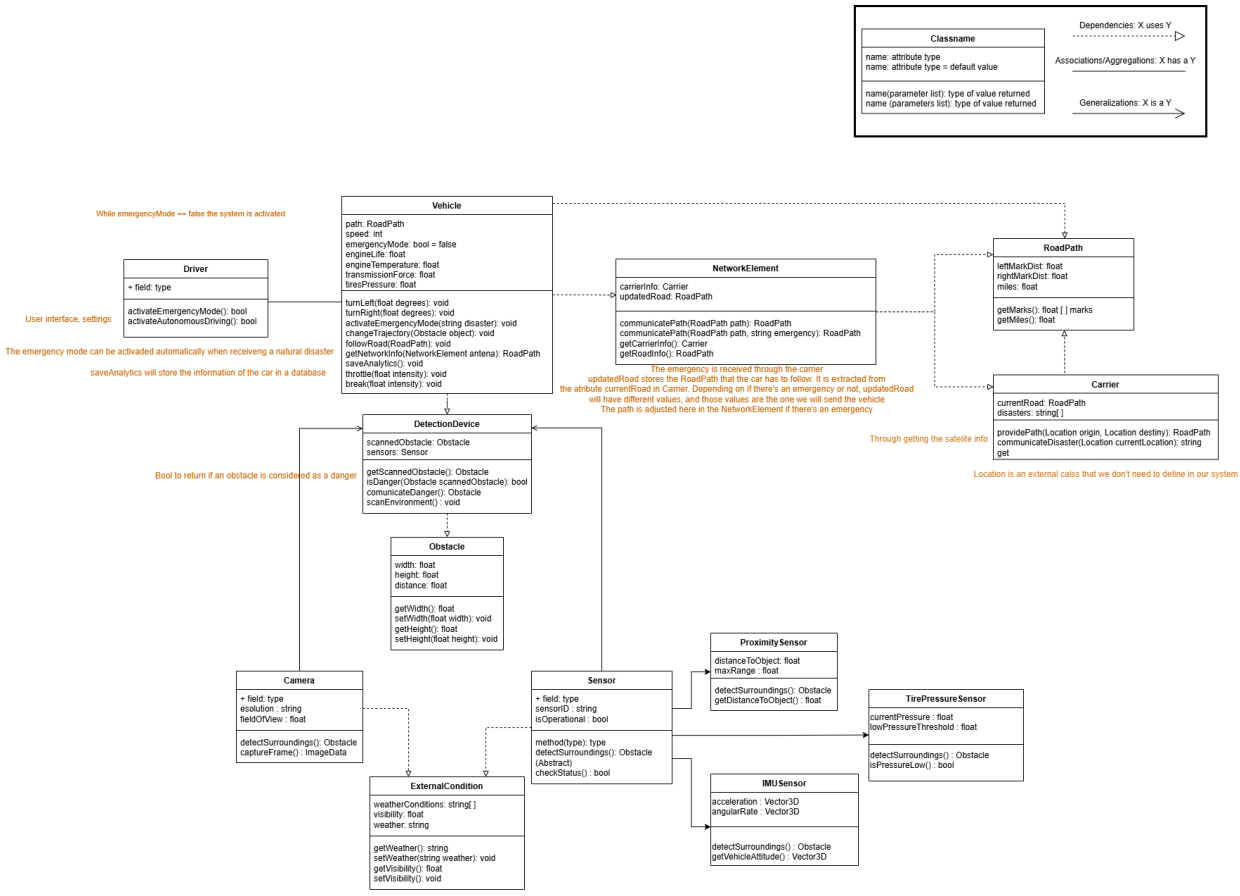
Autonomous Vehicle System

Anwar, John, Daniel, Gabe, Rodolfo, Joseph

CS 250: Introduction to Software Systems

Professor Umut Can Cabuk

Updated UML Class Diagram:



We have made some minor updates to the UML Class Diagram, adding an intensity parameter to the throttle and break function in Vehicle and deleting the relationship existing between Vehicle and ExternalCondition

TEST SET #1:

Targets and tests the DetectionDevice class and its subfunctions in assessing and evaluating the environment around the vehicle.

Unit:

```
// First we declare the necessary constants
const dangerDistance = 10.0
const safeWidth = 0.5
const safeHeight = 0.5

// Obstacles that are at the exact danger distance and different measures
Obstacle obs1 = Obstacle(dangerDistance, safeWidth, safeHeight)
Obstacle obs2 = Obstacle(dangerDistance, safeWidth, safeHeight + 0.1)
Obstacle obs3 = Obstacle(dangerDistance, safeWidth + 0.1, safeHeight)
Obstacle obs4 = Obstacle(dangerDistance, safeWidth + 0.1, safeHeight + 0.1)
Obstacle obs5 = Obstacle(dangerDistance, safeWidth, safeHeight - 0.1)
Obstacle obs6 = Obstacle(dangerDistance, safeWidth - 0.1, safeHeight)
Obstacle obs7 = Obstacle(dangerDistance, safeWidth - 0.1, safeHeight - 0.1)
Obstacle obs8 = Obstacle(dangerDistance, safeWidth + 0.1, safeHeight - 0.1)
Obstacle obs9 = Obstacle(dangerDistance, safeWidth - 0.1, safeHeight + 0.1)

// Obstacles that are closer than the danger distance and different measures
Obstacle obs10 = Obstacle(dangerDistance - 0.1, safeWidth, safeHeight)
Obstacle obs11 = Obstacle(dangerDistance - 0.1, safeWidth, safeHeight + 0.1)
Obstacle obs12 = Obstacle(dangerDistance - 0.1, safeWidth + 0.1, safeHeight)
Obstacle obs13 = Obstacle(dangerDistance - 0.1, safeWidth + 0.1, safeHeight + 0.1)
Obstacle obs14 = Obstacle(dangerDistance - 0.1, safeWidth, safeHeight - 0.1)
Obstacle obs15 = Obstacle(dangerDistance - 0.1, safeWidth - 0.1, safeHeight)
Obstacle obs16 = Obstacle(dangerDistance - 0.1, safeWidth - 0.1, safeHeight - 0.1)
Obstacle obs17 = Obstacle(dangerDistance - 0.1, safeWidth + 0.1, safeHeight - 0.1)
Obstacle obs18 = Obstacle(dangerDistance - 0.1, safeWidth - 0.1, safeHeight + 0.1)

// Obstacles that are further than the danger distance and different measures
Obstacle obs19 = Obstacle(dangerDistance + 0.1, safeWidth, safeHeight)
Obstacle obs20 = Obstacle(dangerDistance + 0.1, safeWidth, safeHeight + 0.1)
Obstacle obs21 = Obstacle(dangerDistance + 0.1, safeWidth + 0.1, safeHeight)
Obstacle obs22 = Obstacle(dangerDistance + 0.1, safeWidth + 0.1, safeHeight + 0.1)
Obstacle obs23 = Obstacle(dangerDistance + 0.1, safeWidth, safeHeight - 0.1)
Obstacle obs24 = Obstacle(dangerDistance + 0.1, safeWidth - 0.1, safeHeight)
Obstacle obs25 = Obstacle(dangerDistance + 0.1, safeWidth - 0.1, safeHeight - 0.1)
Obstacle obs26 = Obstacle(dangerDistance + 0.1, safeWidth + 0.1, safeHeight - 0.1)
Obstacle obs27 = Obstacle(dangerDistance + 0.1, safeWidth - 0.1, safeHeight + 0.1)


testsVector = [obs*]    // Add every obstacle to testsVector
resultsVector = [isDanger(test) for test in testsVector]
```

```
/*Expected output for obstacles that are at the exact danger distance and different measures*/
expectedResults1 = [0, 1, 1, 1, 0, 0, 0, 1, 1]

/*Expected output for obstacles that are closer than the danger distance and different measures*/
expectedResults2 = [0, 1, 1, 1, 0, 0, 0, 1, 1]

/*Expected output for obstacles that are further than the danger distance and different measures*/
expectedResults3 = [0, 0, 0, 0, 0, 0, 0, 0, 0]

// Now we combine all the expected results in a single list
expectedResultsVector = expectedResults1 + expectedResults2 + expectedResults3

If resultsVector == expectedResultsVector:
        return PASS    // The test has passed
else:
        // Print the vectors to see which tests failed
        print(resultsVector)
        print(expectedResultsVector)
        return FAIL      // The test has failed
```

Explanation:
We test the isDanger function considering if a specified obstacle received as a parameter is considered a **danger** or not. For that, we try with different types of obstacles from the **Obstacle** class (width, height, distance). If the obstacle's width or height is less or equal than 0.5 meters we do not consider that the obstacle is a danger to the vehicle. If the obstacle is strictly more than 10.0 meters away no matter its width or height, it will not be considered as a danger either.

In other words, a **danger** is an obstacle that:
distance <= 10 && (weight > 0.5 || height > 0.5)

First, we build all the possible critical combinations of the obstacles. Then, we add them all to a vector called 'testsVector'. After that, we build our 'expectedResultsVector' vector and we apply the function we want to test (inDanger(Obstacle)) to every obstacle in the tests vector. Lastly, if the vectors are the same we conclude that the test is successful, otherwise, we print both vectors to see which results are different and we conclude that the test is unsuccessful.

Integration:

```
ProximitySensor()
maxRange = 10
obstacle f
obstacle t
distanceToObject(t) = 11
distanceToObject(f) = 7

If detectSurroundings() == f, t
        return PASS
else
        return FAIL
DetectionDevice()
If isDanger(f) = false && isDanger(t) = true
        return PASS
else
        return FAIL
```

Explanation:
The integration test makes sure that the whole ProximitySensor class works fine when detecting and assessing the environment around the vehicle. Here in this example we make sure that the detectSurroundings() method correctly returns obstacles surrounding a vehicle when they're detected that are closer than the maxRange distance. Not passing this test is a serious risk for the driver since it jeopardizes their safety and more tests should be conducted on the whole class.

System:

Explanation:

TEST SET #2:
Targets and tests ____ feature/function

Unit:

Explanation:

Integration:

Explanation:

System:

Explanation:

TO ADD:

- [ ] Image of if needed updated SWA Diagram + Description