Racecar – Development Report

Smit Patel

Objective:

For this Race car Project, We have to develop a Rule Based agent, that can complete all 8 tracks using Lidar sensor, and focus staying on center of the track with completing all track without crashing

Development Strategy:

My Development strategy is that I gave 5 Lidar sensors for Farleft, Left, Center, Right , FarRight to run the car more centralized. If there is a sharp turn, car turns based on the Lidar sensor readings. For steering, it is most probably based on the each side of the road distance. Speeding control is based on the road, if the road is straight, speed is high. And If the road has turns, speed is low.(I have defined velocity as speed).For Last track(Spiral), there is a threshold used to avoid crashing. I tried to stabilize the car on white line with moderate speed to ensure safety.

Code:

```
import random

class Agent:

    def chooseAction(self, observations, possibleActions):

        #for lidar and speed info

        lidar = observations['lidar']

        velocity = observations['velocity']

        #inf is for bigger number

        lidar = [d if d != float('inf') else 100 for d in lidar]

        farLeft, left, center, right, farRight = lidar

        #for driving car in center

        leftSide = farLeft + left

        rightSide = farRight + right

        #for sharp turn
```

```python
        if min(lidar) < 0.05:
            if ('straight', 'brake') in possibleActions:
                return ('straight', 'brake')
            return random.choice(possibleActions)
        #decide left and right turn
        if rightSide + 0.3 < leftSide:
            move = 'left'
        elif leftSide + 0.3 < rightSide:
            move = 'right'
        else:
            move = 'straight'
        #narrow path
        if center < 0.6 or left < 0.5 or right < 0.5:
            if (move, 'brake') in possibleActions:
                return (move, 'brake')
        if velocity > 0.15:
            if (move, 'brake') in possibleActions:
                return (move, 'brake')
            elif (move, 'coast') in possibleActions:
                return (move, 'coast')
        #speedign on straight road
        if velocity <= 0.15 and center > 1.0 and left > 0.8 and right >
0.8:
            if (move, 'accelerate') in possibleActions:
                return (move, 'accelerate')
        #if nothing matches, just coast (means no accelerate or brake)
        if (move, 'coast') in possibleActions:
            return (move, 'coast')
        #last backup
        return random.choice(possibleActions)
```

```
    def load(self, data=None):

        pass
```

Performance:

1st Iteration

This is the main code that I have used and committed at first, I ran this 100 times for every track and get the Average.

Track 1 → Average Reward = 112.461501466129

Track 2 → Average Reward = 113.617434974375

Track 3 → Average Reward = 115.527475325037

Track 4 → Average Reward = 115.505632220539

Track 5 → Average Reward = 117.438472242391

Track 6 → Average Reward = 115.037062301693

Track 7 → Average Reward = 115.97211466462

Track 8 → Average Reward = 110.828431169843

2nd Iteration

In second Commit, I updated some values,

Turn threshold from “0.3” to “0.2”

Brake for speed from “0.15” to “0.19”

Speed(Velocity) from “0.15” to “0.30”

And I Got new scores:

Track 1 → Average Reward = 120.032851468588

Track 2 → Average Reward = 120.064774947006

Track 3 → Average Reward = 117.075920344431

Track 4 → Average Reward = 117.143614075571

Track 5 → Average Reward = 119.635659832118

Track 6 → Average Reward = 120.888290616874

Track 7 → Average Reward = 118.716120351477

Track 8 → Average Reward = 117.884907712841

Compared to Previous results, these new values showed better consistency and speed. New code gives more stable performance as compared to previous one.

3rd Iteration

In third Commit, I updated some values,

Velocity from "0.3" to "0.5"

And I Got new scores:

Track 1 → Average Reward = 117.583457065607

Track 2 → Average Reward = 121.257825128989

Track 3 → Average Reward = 122.081297964599

Track 4 → Average Reward = 120.787945758318

Track 5 → Average Reward = 121.052135781843

Track 6 → Average Reward = 117.31463406831

Track 7 → Average Reward = 119.917302568724

Track 8 → Average Reward = 119.09450318443

Compared to the second iteration, these new scores showed slightly better consistency and speed, Except Spital(8th track).

4th Iteration

In Fourth Commit, I tried the Epsilon Greedy method, IN which, Epsilon greedy value is "0.05". Rest all the values are same as the main code.

And I Got new scores:

Track 1 → Average of 100 = 122.346149210975

Track 2 → Average of 100 = 122.359979939718

Track 3 → Average of 100 = 123.190521094216

Track 4 → Average of 100 = 123.337553348831

Track 5 → Average of 100 = 122.400505141986

Track 6 → Average of 100 = 121.345290904165

Track 7 → Average of 100 = 122.209454701955

Track 8 → Average of 100 = 121.515793274363

Due to Leaderboard Issue, I committed last code and tried Epsilon Greedy Method, And got little bit better scores compared to Previous results.

Conclusion:

This Race Car project helped me understand how to build and tune a rule-based agent using LIDAR sensor inputs. I developed a car agent that successfully passes all 8 tracks by making decisions based on sensor distances and adapting speed and direction according to the track. Through multiple iterations, I improved the car's performance by adjusting turning thresholds, braking logic, and speeding conditions. My final version achieved higher average rewards with consistent and stable behavior across all tracks. This project taught me the importance of building logic based on sensor data and balancing between speed and safety to build an efficient autonomous driving agent.