Department of
Computer Engineering

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

به نام خالق زیبایی ها

پروژه عملی دوم

درس هوش محاسباتی

امیرفاضل کوزه گر کالجی

9931099

فاز اول "مسیر یابی"

مرحله اول)

در این مرحله با توجه به دستور کار، می‌بایستی که شیب و معادله خط های نمودار ها را با توجه به بازه شان، می یافتیم.

برای محاسبه معادله خط در حالات مختلف، برای هر حالت یک تابع تعریف کردم.

در تابع های اولیه، ابتدا مقادیر معادله خط را دستی حساب می کردم و سپس در کد وارد می‌کردم:

```
    def eq_far_L(self, x):...

    def eq_moderate_L(self, x):...

    # Right distance equations
    def eq_close_R(self, x):...

    def eq_far_R(self, x):...

    def eq_moderate_R(self, x):...

    # Rotate amount equations

    def eq_high_R(self, x):...

    def eq_low_R(self, x):
        if -20 < x < -10:
            point1 = Point(-20, 0)
```

در ادامه تصمیم گرفتم برای تابع های بعدی، کارم را راحت کرده و معادله خط را نیز توسط کد محاسبه کنم

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y


class FuzzyController:
    """
    #todo
    write all the fuzzify,inference,defuzzify method in this class
    """

    def __init__(self):
        pass

    def equation(self, point1, point2, x):
        gd = float(point2.y - point1.y) / float(point2.x - point1.x)
        return gd * (x - point1.x) + point1.y
```

با توجه به عکس بالا، یک کلاس نقطه و یک متد برای محاسبه معادله خط، به کد اضافه نمودم.

```python
def eq_low_R(self, x):...

def eq_nothing(self, x):...

def eq_low_L(self, x):...

def eq_high_L(self, x):
    if 5 < x < 20:
        point1 = Point(5, 0)
        point2 = Point(20, 1)
        return self.equation(point1, point2, x)
    elif 20 <= x < 50:
        point1 = Point(20, 1)
        point2 = Point(50, 0)
        return self.equation(point1, point2, x)
    return 0
```

با توجه به عکس بالا، از راهکار جدیدم استفاده کردم و با دادن دو نقطه از خط، محاسبه معادله خط را به کد واگذار کردم :))

مرحله دوم)

در این مرحله باید از قوانین استفاده می‌کردیم تا مقادیر تعلق را به دست آوریم:

```python
low_right_membership = min(self.eq_close_L(left_dist), self.eq_moderate_R(right_dist))
high_right_membership = min(self.eq_close_L(left_dist), self.eq_far_R(right_dist))
nothing_membership = min(self.eq_moderate_R(right_dist), self.eq_moderate_L(left_dist))
low_left_membership = min(self.eq_close_R(right_dist), self.eq_moderate_L(left_dist))
high_left_membership = min(self.eq_close_R(right_dist), self.eq_far_L(left_dist))
```

سوال امتیازی

وقتی چندین قانون با مجموعه نهایی یکسان فعال شوند، می توان برای محاسبه مقدار تعلق نهایی این مجموعه از روش های مختلفی استفاده کرد. یکی از این روشها، روش استفاده از تابع بیشینه یا maximum است.

در این روش برای محاسبه مقدار تعلق نهایی، از بین تمامی قانون هایی که با مجموعه نهایی یکسان فعال شده اند، تابع عضویت بیشینه را انتخاب می کنیم و مقدار تعلق نهایی را برابر با مقدار این تابع در نقطه مورد نظر خود محاسبه می‌کنیم.

مرحله سوم)

با توجه به نکات گفته شده در دستور کار، این امکان وجود دارد در هنگام طی جریان بازی، مقادیر متفاوتی برای حالت های چرخیدن مان به وجود بیاید؛ به همین دلیل بزرگترین مقدار را در نظر می‌گیریم.

تابعی که در هر مقدار **x**، مقدار بیشینه را به ما دهد، به صورت زیر تعریف شده است:

```
def max_rotate_val(x):
    LR = min(low_right_membership, self.eq_low_R(x))
    HR = min(high_right_membership, self.eq_high_R(x))
    NTH = min(nothing_membership, self.eq_nothing(x))
    LL = min(low_left_membership, self.eq_low_L(x))
    HL = min(high_left_membership, self.eq_high_L(x))

    return max(LR, HR, NTH, LL, HL)
```

در ادامه باید میزان چرخش را حساب کنیم تا بتوانیم تابع تصمیم را به هدف نهایی خود برسانیم. با کمک گرفتن از نمونه کد آورده شده درون خود دستورکار، روند محاسبه مقدار مرکز جرم را به شکل زیر انجام می‌دهیم:

```
numerator = 0.0
denominator = 0.0
lineSpace = self.linespace(-50, 50, 1000)
delta = lineSpace[1]
for i in lineSpace[0]:
    U = max_rotate_val(i)
    numerator += U * i * delta
    denominator += U * delta
if denominator != 0:
    return 1.0 * float(numerator) / float(denominator)
return 0
```

فاز دوم "تعیین سرعت"

در این فاز نیز تمامی کار های انجام شده همانند فاز قبلی می‌باشد با این تفاوت که این بار فازی سازی برای محاسبه میزان سرعت در حالات گوناگون انجام می‌شود.

در مرحله اول توابعی تعریف می کنیم تا معادله خط را برای حالات متفاوت به ما بدهند. اینجا نیز از کلاس Point و تابعی برای محاسبه معادله خط بهره برده ایم.

```
def eq_close_dist(self, x):...

def eq_moderate_dist(self, x):...

def eq_far_dist(self, x):...

def eq_low_gas(self, x):...

def eq_medium_gas(self, x):...

def eq_high_gas(self, x):...
```

در ادامه میزان تعلقات را حساب می کنیم:

```
low_membership = self.eq_close_dist(center_dist)
high_membership = self.eq_far_dist(center_dist)
medium_membership = self.eq_moderate_dist(center_dist)
```

سپس تابعی برای محاسبه مقدار بیشینه، و در نهایت مرکز جرم را حساب می‌کنیم و بر می‌گردانیم:

```
def max_gas_val(x):
    L = min(low_membership, self.eq_low_gas(x))
    H = min(high_membership, self.eq_high_gas(x))
    M = min(medium_membership, self.eq_medium_gas(x))
    return max(L, H, M)

numerator = 0.0
denominator = 0.0
lineSpace = self.linespace(0, 90, 900)
delta = lineSpace[1]
for i in lineSpace[0]:
    U = max_gas_val(i)
    numerator += U * i * delta
    denominator += U * delta
if denominator != 0:
    return 1.0 * float(numerator) / float(denominator)
return 0
```

نکته قابل توجه، این است که پس از نوشتن کد های این قسمت و اجرای مجدد بازی، ماشین در جای خود ایستاده و تکان نمی‌خورد و مشکل آن این است که در x های بزگتر از 200، چون مقداری تعریف نکرده ایم 0 برگردانده می‌شود و ماشین در جای خود ثابت می‌ماند. برای رفع این مشکل، برای x های بزرگتر از 200 مقدار 1 را برمی‌گردانم و کد را به شکل زیر باز سازی می‌کنم:

```
def eq_far_dist(self, x):
    if 90 <= x < 200:
        point1 = Point(90, 0)
        point2 = Point(200, 1)
        return self.equation(point1, point2, x)
    if x >= 200:
        return 1
    return 0
```

نکته اضافه:

تابع linespace از متد های نامپای بوده ولی تصمیم گرفتم آن را به صورت ساده و دستی پیاده سازی، کنم که به شکل زیر است:

```
def linespace(self, start, stop, count):
    delta = (stop - start) / count
    evenly_spaced = [start + i * delta for i in range(count + 1)]
    return evenly_spaced, delta
```