Homework 5: Car Tracking

110550085 房天越

Part I: Implementation(15%):

Part1:

```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE
    '''
    In this part, for all the coordinates in the grid:
        First, I calculate the distance of my car with Pythagoras' theorem.
        Then, I calculate the PDF with the function given in spec.
        At last, I update the probability with the current probability multiplied by the PDF.
    Finally, I normalize self.belief with normalize function.
    '''
    for row in range(self.belief.numRows):
        for col in range(self.belief.numCols):
            d_ma=math.sqrt((util.colToX(col)-agentX)*(util.colToX(col)-agentX)+(util.rowToY(row)-agentY)*(util.rowToY(row)-agentY))
            pdf_cal=util.pdf(d_ma, Const.SONAR_STD, observedDist)
            prob=self.belief.getProb(row, col)
            self.belief.setProb(row, col, prob*pdf_cal)
    self.belief.normalize()
    # END_YOUR_CODE
```

Part2:

```python
def elapseTime(self) -> None:
    if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
        return
    # BEGIN_YOUR_CODE
    '''
    In this part, I first set the new belief's all values to zero.
    Then update probability for the new belief with the new location and delta.
    Finally, normalize the new belief and set self.belief to it.
    '''
    newbelief=util.Belief(self.belief.numRows, self.belief.numCols, value=0)
    for oldtile, newtile in self.transProb:
        # [0] is row, [1] is column.
        newbelief.addProb(newtile[0], newtile[1], self.belief.getProb(oldtile[0], oldtile[1])*self.transProb[(oldtile, newtile)])
    newbelief.normalize()
    self.belief=newbelief
    return
    # END_YOUR_CODE
```

Part3-1:

```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE
    '''
    In this part, first I create a dictionary to store the current particle that is being calculated.
    Then, do the things like in part1.
    Finally, I resample all new particles with weightRandomChoice() function, and set it to self.particles.
    '''
    tmpparticle=collections.defaultdict(float)
    for row, col in self.particles:
        d_ma=math.sqrt((util.colToX(col)-agentX)*(util.colToX(col)-agentX)+(util.rowToY(row)-agentY)*(util.rowToY(row)-agentY))
        pdf_cal=util.pdf(d_ma, Const.SONAR_STD, observedDist)
        tmpparticle[(row, col)]=self.particles[(row, col)]*pdf_cal
    newparticle=collections.defaultdict(int)

    for i in range(self.NUM_PARTICLES):
        particle=util.weightedRandomChoice(tmpparticle)
        newparticle[particle]+=1
    self.particles=newparticle
    # END_YOUR_CODE

    self.updateBelief()
```

Part3-2:

```python
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE
    '''
    In this part, I set a new dictionary newparticle, for every
    particle in self.particles, I call weightRandomChoice for every
    particle at the location. Finally, I update self.particles with
    newparticle.
    '''
    newparticle=collections.defaultdict(int)
    for particle in self.particles:
        pnum=self.particles[particle]
        for i in range(pnum):
            newtile=self.transProbDict[particle]
            tmpparticle=util.weightedRandomChoice(newtile)
            newparticle[tmpparticle]+=1
    self.particles=newparticle
    # END_YOUR_CODE
```

Part II. Question answering (5%):

- Please describe problems you met and how you solved them

In this homework, I find that the most difficult part is understanding the concept, especially it requires the basis of probability and I am really bad at this. After reading the spec and doing some research about it. I can finally realize the concept and write the code.

Also, the spec tells us to try to play the game manually first with the code:

python drive.py -l lombard -i none

However, in the given code, the part2 and part3 had to be modified to do that. After some modification, I was able to play that.