Homework 5: Car Tracking

Part I. Implementation (15%):

- Part1:

  First, we get the size of the map by calling “self.belief.getNumRows()” and “self.belief.getNumCols()”.

  For each position in the map we convert it into x,y location
  And calculate the distance to our agent,
  Have this distance, with observedDist and const.SONAR_STD we can use “util.pdf()” to get the “probability density of a Gaussian distribution”

  Therefore we can use pervious belief $P(H_t|E_{1:t-1})$* $P(H_t|E_t)$ to get new belief $P(H_t|E_{1:t})$. and we can use “self.belief.setProb()” function to update it

  The last we normalized the probability to make it sum equal 1.

```python
    def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
        # BEGIN_YOUR_CODE
        rows = self.belief.getNumRows()
        cols  = self.belief.getNumCols()
        for i in range(rows):
            for j in range(cols):
                y = util.rowToY(i)
                x = util.colToX(j)
                distance = math.sqrt((x-agentX)**2 +(y-agentY)**2)
                density = util.pdf(distance, Const.SONAR_STD, observedDist)
                prob = self.belief.getProb(i,j)
                self.belief.setProb(i, j, density * prob)

        self.belief.normalize()
        return
        # END_YOUR_CODE
```

- Part2:

First, we get the size of the map by calling “self.belief.getNumRows()” and “self.belief.getNumCols()”.

And we use these two information to create a new belief.

* the form of self ((oldx,oldy) , (newx,newy)) : prob
By the transProb we can get the probability of transfer old position to new position.
Furthermore we can use “self.belief.getProb()” to get old probability $P(H_t|E_{1:t})$ and multiply it with transfer probability $P(H_{t+1}|E_t)$ to get newbelief $P(H_{t+1}|E_{1:t})$ and (use.addProb()) function to get add it to new belief.

The last we normalized the probability to make it sum equal 1.

```python
    def elapseTime(self) -> None:
        if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
            return
        # BEGIN_YOUR_CODE
        rows = self.belief.getNumRows()
        cols  = self.belief.getNumCols()
        new_belief = util.Belief(rows,cols,0)
        # print(self.transProb)
        for old, new in self.transProb:
            now_prob = self.belief.getProb(old[0],old[1])
            trans_prob = self.transProb[(old,new)]
            new_belief.addProb(new[0], new[1],  now_prob * trans_prob)

        new_belief.normalize()
        self.belief = new_belief
        return

        # END_YOUR_CODE
```

- Part3-1:

  First：

  For each particle in now self.particles(), we first convert it position into x,y and calculate the distance to our agent.
  Have this distance, with observedDist and const.SONAR_STD we can use "util.pdf()" to get the "probability density of a Gaussian distribution"
  With the probability of old particles, we can calculate the new particle position's probability and record it into the particle_dict.

  Second：

  For each new particle we get before we use "util.weightedRondomchoice()" to get the new position of each particles and record it into the dictionary.
  Last we update self.particles with the new particle we get.

```python
    def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
        # BEGIN_YOUR_CODE
        #Re-weight
        particle_dict = collections.defaultdict(float)
        for row, col in self.particles:
            y = util.rowToY(row)
            x = util.colToX(col)
            distance = math.sqrt((x-agentX)**2 +(y-agentY)**2)
            density = util.pdf(distance, Const.SONAR_STD, observedDist)
            prob = self.particles[(row, col)]
            particle_dict[(row,col)] = prob*density
        #Re-sample
        #print(self.particles)
        new_particle = collections.defaultdict(int)
        for _ in range(self.NUM_PARTICLES):
            new_particle[util.weightedRandomChoice(particle_dict)] += 1
        self.particles = new_particle
        # END_YOUR_CODE

        self.updateBelief()
```

- Part3-2:

  We first initialize an dictionary to record the new particle distribution. And we use the function "util.weightedRondomchoice()" to get the new position of each particles and record it into the dictionary.
  Last we update self.particles with the new particle we get.

```python
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE
    new_particle = collections.defaultdict(int)
    for old_particle in self.particles:
        for _ in range(self.particles[old_particle]):
            new_particle[util.weightedRandomChoice(self.transProbDict[old_particle])]+=1
    self.particles = new_particle
    # END_YOUR_CODE
```

Part II. Question answering (5%):

The problem I encountered is the math in this assignment, when I see the math in the spec I have no idea about it. To solve this problem I read the more information in the spec and search on google to realize the math and why calculate in this way. When I start to write the code I remember the what goal to do in each part, but don't know how to start it. Therefore, I start to read util.py and the functions in it and know the structure of the class and know how to use this function to help me finish this assignment.