Homework 5: Car Tracking

Please keep the title of each section and delete examples.

Part I. Implementation (20%):

- Please screenshot your code snippets of Part 1 ~ Part 3, and explain your implementation.

Part 1:

```
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE (our solution is 9 lines of code, but don't worry if you deviate from this)
    for row in range(self.belief.numRows):
        for col in range(self.belief.numCols):
            x = util.colToX(col)
            y = util.rowToY(row)
            dist = math.sqrt((x-agentX)**2+(y-agentY)**2) #calculate distance from car location to (x,y)
            prob_dist = util.pdf(dist, Const.SONAR_STD, observedDist) #calculate probability density
            p = prob_dist*self.belief.getProb(row, col) #calculate current posterior probability
            self.belief.setProb(row, col, p) #update
    self.belief.normalize()
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE

    # For each column and each row, convert (row, column) to (x, y) location
    # Use Pythagoras to calculate the distance from (agentX, agentY) car location to (x, y) location
    # Calculate probability density of distance and calculate current posterior probability value
    # Update value to self.belief
```

Part 2:

```
def elapseTime(self) -> None:
    if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
        return
    # BEGIN_YOUR_CODE (our solution is 10 lines of code, but don't worry if you deviate from this)
    new_belief = util.Belief(self.belief.numRows, self.belief.numCols, 0) #init
    for oldTile, newTile in self.transProb:
        delta = self.belief.getProb(oldTile[0], oldTile[1])*self.transProb[(oldTile, newTile)]
        new_belief.addProb(newTile[0], newTile[1], delta) #calculate posterior probability
    new_belief.normalize()
    self.belief = new_belief
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE

    # First, initialize temp dictionary have zero probability
    # For each ((oldTile, newTile), transProb) pair, to calculate multiple
    # posterior probability current time with transition probability
    # Add value to temp dictionary (newTile[0], newTile[1])
    # Update temp dictionary to self.belief
```

Part 3:

```
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE (our solution is 12 lines of code, but don't worry if you deviate from this)
    #Re-weight
    weight_dict = collections.defaultdict(float)
    for row, col in self.particles:
        x = util.colToX(col)
        y = util.rowToY(row)
        dist = math.sqrt((x-agentX)**2+(y-agentY)**2)
        prob_dist = util.pdf(dist, Const.SONAR_STD, observedDist)
        weight_dict[(row, col)] = prob_dist*self.particles[(row, col)]
    #Re-sample
    new_particles = collections.defaultdict(int)
    for i in range(self.NUM_PARTICLES):
        particle = util.weightedRandomChoice(weight_dict)
        new_particles[particle]+=1
    self.particles = new_particles
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE
    self.updateBelief()

    # For each column and each row, convert (row, column) to (x, y) location
    # Use Pythagoras to calculate the distance from (agentX, agentY) car location to (x, y) location
    # Calculate probability density of distance and calculate current posterior probability value
    # Initialize temp dictionary have zero probability
    # Random choice new location in posterior probability dictionary and count the number of times
    # location is chosen
    #Update new value tp self.particles
```

```
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
    new_particles = collections.defaultdict(int)
    for particle in self.particles:
        for _ in range(self.particles[particle]):
            p = util.weightedRandomChoice(self.transProbDict[particle])
            new_particles[p]+=1
    self.particles = new_particles
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE

    # Initialize temp dictionary have zero probability
    # For each particle, random choice particle location in transition probabilities
    # and count the number of times location is chosen
    # Update value to self.particles
```