Homework 5: Car Tracking

Part I. Implementation (15%):

Part 1:

```
def observe(self, agentX: int, agentY: int, observedDist: float) → None:
    # BEGIN_YOUR_CODE
    """
    For each place in grid:
        get the distance of the grid to my car and get its pdf with mean: dist, std: SONAR_STD
        set the probability of belief to current probability.
    Last, normalize the belief.
    """
    for row in range(self.belief.numRows):
        for col in range(self.belief.numCols):
            dist = math.sqrt((util.colToX(col) - agentX) ** 2 + (util.rowToY(row) - agentY) ** 2)
            prob_distr = util.pdf(dist, Const.SONAR_STD, observedDist)
            self.belief.setProb(row, col, self.belief.getProb(row, col) * prob_distr)
    self.belief.normalize()
    # END_YOUR_CODE

############################################################################
```

Part 2:

```
def elapseTime(self) → None:
    if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
        return
    # BEGIN_YOUR_CODE
    """
    nb for newBelief for set the default all 0.
    For every transition (old → new) in self.transProb:
        cur_prob = old.probability
        trans = transProb(old→new)
        delta = cur_prob * trans
        addProb(new_row, new_col, delta)
    Last, normalize the newBelief and set it to belief.
    """
    nb = util.Belief(self.belief.numRows, self.belief.numCols, value=0)
    for otile, ntile in self.transProb:
        cur_prob = self.belief.getProb(otile[0], otile[1])
        trans = self.transProb[(otile, ntile)]
        nb.addProb(ntile[0], ntile[1], cur_prob * trans)
    nb.normalize()
    self.belief = nb
    # END_YOUR_CODE

# Function: Get Belief
```

Part 3-1:

```python
##################################################################################
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE
    """
    Create new dict (proposed) to store current particles.
    For each particles:
        get the distance of the grid to my car and get its pdf with mean: dist, std: SONAR_STD
        set current particle * pdf into proposed[cur_particle]
    Create new dict (np) to store newParticles.
    Do NUM_PARTICLES times:
        choose a particle randomly in proposed
        add 1 of value of np which index = particle
    Last, set the np to particles.
    """
    proposed = collections.defaultdict(float)
    for row, col in self.particles:
        dist = math.sqrt((util.colToX(col) - agentX) ** 2 + (util.rowToY(row) - agentY) ** 2)
        prob_distr = util.pdf(dist, Const.SONAR_STD, observedDist)
        proposed[(row, col)] = self.particles[(row, col)] * prob_distr
    np = collections.defaultdict(int)
    for i in range(self.NUM_PARTICLES):
        particle = util.weightedRandomChoice(proposed)
        np[particle] += 1
    self.particles = np
    # END_YOUR_CODE

    self.updateBelief()

##################################################################################
```

Part 3-2:

```python
##################################################################################
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE
    """
    Create new dict (np) to store newParticles.
    For every particles:
        confirm that tile is in transProbDict
        call weightRandomChoice for every particle at the location
        new_weight = transition from old to new and do weightRandom Choice based on new weightDict
    Last, set the np to particles and updateBelief.
    """
    np = collections.defaultdict(int)
    for tile, value in self.particles.items():
        if tile in self.transProbDict:
            for v in range(value):
                new_weight = self.transProbDict[tile]
                p = util.weightedRandomChoice(new_weight)
                np[p] += 1
    self.particles = np
    self.updateBelief()
    # END_YOUR_CODE
```

Part III. Question Answering (5%):

I have learned terribly in probability last semester, so I got much strenuous in this homework. However, “observe” and “elapseTime” are doing similar things so that I got better in the second half.