Homework 5: Car Tracking Report

Part I. Implementation (20%):

Part 1:

```python
    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.getNumRows()):
            for col in range(self.belief.getNumCols()):
                u = math.sqrt((agentY - rowToY(row)) ** 2 + (agentX - colToX(col)) ** 2)
                p = util.pdf(u, Const.SONAR_STD, observedDist)
                self.belief.setProb(row, col, p * self.belief.getProb(row, col))
        self.belief.normalize()
        return
        raise Exception("Not implemented yet")
        # END_YOUR_CODE
```

Explanation of part 1:

For every tile, we first compute the true distance from agent to the tile as our mean of Gaussian distribution. Then, with the mean(u), Const.SONAR_STD, and observed distance, we can get the probability by calling util.pdf(). Thus, we are able to update $P(H_t|E_{1:t})$ with previous belief and product of $P(E_t|H_t)$ and $P(H_t|E_{1:t-1})$ by calling belief.setProb(). Finally, normalize the probability we just updated for every tile because sum of them may exceed 1.

Part 2:

```python
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.getNumRows(), self.belief.getNumCols(), value=0)
    for oldTile, newTile in self.transProb:
        new_belief.addProb(newTile[0], newTile[1], self.belief.getProb(*oldTile) * self.transProb[(oldTile, newTile)])
    new_belief.normalize()
    self.belief = new_belief
    return
    raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Explanation of part 2:

First, create a new belief which will be assigned to self.belief after finish computing the $P(H_{t+1}|E_{1:t})$ to new belief by transition probability of every (old_tile, new_tile) ($P(H_{t+1}|H_t)$ and $P(H_t|E_{1:t})$). And then, like part 1, we should do normalization of probability to ensure sum of probability would not exceed 1.

Part 3-1:

```python
    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)
        observe_distribution = collections.defaultdict(float)
        for row, col in self.particles:
            u = math.sqrt((agentY - rowToY(row)) ** 2 + (agentX - colToX(col)) ** 2)
            p = util.pdf(u, Const.SONAR_STD, observedDist)
            observe_distribution[(row, col)] = self.particles[(row, col)] * p
        new_particles = collections.defaultdict(int)
        for _ in range(self.NUM_PARTICLES):
            particle = util.weightedRandomChoice(observe_distribution)
            new_particles[particle] += 1
        self.particles = new_particles
        # raise Exception("Not implemented yet")
        # END_YOUR_CODE
        self.updateBelief()
```

Explanation of part 3-1:

In the first half of part 3-1, we do same thing as part 1, the only thing different is that we replace self.belief with self.particles[tile] (descrbes the number of particles on particular tile) and observe_distribution (record the new distribution of particles).

Then, create new_particle which will be assigned to self.particles after we resample the particles in next half. In the second half of part 3-1, we call util.weightedRandomChoice() with observe_distribution to get the tile where the number of particles will increase.

Part 3-2:

```python
    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_location in self.particles:
            if particle_location in self.transProbDict:
                for _ in range(self.particles[particle_location]):
                    tile = util.weightedRandomChoice(self.transProbDict[particle_location])
                    new_particles[tile] += 1
        self.particles = new_particles
        return
        raise Exception("Not implemented yet")
        # END_YOUR_CODE
```

Explanation of part 3-2:

This part is also similar to part 2 roughly, and we call function util.weightedRandomChoice() with transition probability for every particle to get new particles distribution. Finally, assign new particles distribution to self.particles and finish part 3-2.