Homework 5: Car Tracking

Part I. Implementation (20%):

Part1

$D_t \sim N(\|a_t - C_t\|_2, \sigma^2)$

a_t：our car's position

C_t：other car's position

What we have to do in part 1 is $P(C_t | D_1 = d_1, \ldots, D_{t-1} = d_{t-1})\ p(d_t | c_t)$

$p(d_t | c_t)$ means that knowing other car's position, the probability of that dt equals to distance of our car and other car.

We have to update the whole two dimension array in belief.

Util.pdf can help us calculate $D_t \sim N(\|a_t - C_t\|_2, \sigma^2)$

At last, normalize it.

```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)
    #raise Exception("Not implemented yet")
    for row in range(self.belief.numRows):
        for col in range(self.belief.numCols):
            distance = math.sqrt((util.colToX(col) - agentX) ** 2 + (util.rowToY(row) - agentY) ** 2)
            probability = util.pdf(distance, Const.SONAR_STD, observedDist)
            self.belief.setProb(row, col, self.belief.getProb(row, col) * probability) # get the old value and update it with new value
    self.belief.normalize()
    # END_YOUR_CODE
```

Part2

$\sum_{c_t} P(C_t = c_t | D_1 = d_1, \ldots, D_t = d_t) p(c_{t+1} | c_t)$

$p(c_{t+1} | c_t)$ is the probability of the other car being in newTile at time step t+1 given that it was in oldTile at time step t.

In the beginning, we initialize a new belief, newBelief.

After observation, we have the probability of the car's position when time=t.

All we have to do is to add all the action's transition probability which can let the car go to newTile to newTile's probability.

As a result, we can predict where the other car will most likely go to.

```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)
    newBelief = util.Belief(self.belief.numRows, self.belief.numCols, value=0)
    for oldTile, newTile in self.transProb:
        newBelief.addProb(newTile[0], newTile[1], self.belief.getProb(*oldTile) * self.transProb[(oldTile, newTile)])
    newBelief.normalize()
    self.belief = newBelief
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Part 3-1

We transfer the probability to another format -> particles.

When we have to update particles, we let the old number of particles times the probability we have calculated.

Next, we add the particles in weighted random way. It means that the higher

the probability it is, the more chances it will be chosen to add particles.

```
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)
    #raise Exception("Not implemented yet")
    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
    newParticles = collections.defaultdict(int)
    for i in range(self.NUM_PARTICLES):
        particle = util.weightedRandomChoice(proposed)
        newParticles[particle] += 1
    self.particles = newParticles
    # END_YOUR_CODE
    self.updateBelief()
```

Part 3-2

trasProDict is a dictionary that records the probability that every oldTile goes to a newTile.

newWeightDict is the dictionary that records the probability that the given oldTile goes to a newTile. Like part 3-1, using weightedRandomChoice, we can convert the probability to particles.

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