Homework 5: Car Tracking
109550164 徐聖哲

Part I. Implementation (20%)

PART 1

In part1, we have to calculate the probability by emission probability and belief. First change row and col to x and yand calculate probability density function by util.pdf which use the distance between agent and the station car to generate mean. Then, get the belief from self.belief.getprob() and multiply both which is the final probability. Finally, add it to belief and normalize belief .

```
53      def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
54          # BEGIN_YOUR_CODE (our solution is 9 lines of code, but don't worry if you deviate from this)
55          for row in range(self.belief.getNumRows()):
56              for col in range(self.belief.getNumCols()):
57                  x = util.colToX(col);
58                  y = util.rowToY(row);
59                  p = util.pdf(math.sqrt((agentX - x)**2 + (agentY - y)**2), Const.SONAR_STD, observedDist);
60                  pre = self.belief.getProb(row, col);
61                  post = pre*p;
62                  self.belief.setProb(row, col, post);
63          self.belief.normalize();
64          # raise Exception("Not implemented yet")
65          # END_YOUR_CODE
```

PART 2

In part2, we calculate the probability using transition probabilities, because the other car will move, so we have alter the probability in belief to apply the situation.
First, let new be the temp belief in order to change the value later. Then, get the transition probability from self.transProb , let p be the value. Finally, multiply pre and p to calculate the probability and add to new.

```
87      def elapseTime(self) -> None:
88          if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
89              return
90          # BEGIN_YOUR_CODE (our solution is 10 lines of code, but don't worry if you deviate from this)
91          new = util.Belief(self.belief.getNumRows(), self.belief.getNumCols(), 0)
92          for (oldTile, newTile) in self.transProb:
93              pre = self.belief.getProb(oldTile[0], oldTile[1])
94              p = self.transProb[(oldTile, newTile)]
95              post = pre * p
96              new.addProb(newTile[0], newTile[1], post)
97          new.normalize()
98          self.belief = new
99          # raise Exception("Not implemented yet")
100         # END_YOUR_CODE
```

PART 3-1

First, create a dictionary from collections.counter() which is used to store particle probability. Then, use part1 to calculate probability and store them in self.particles[tile]. Finally, random choose the particles, if a particle is chosen, add the value in the dictionary by one, and refresh self.particles by new.

```
    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)
        new = collections.Counter()
        for tile in self.particles.keys():
            x = util.colToX(tile[1]);
            y = util.rowToY(tile[0]);
            pre = self.particles[tile];
            p = util.pdf(math.sqrt((agentX - x) ** 2 + (agentY - y) ** 2), Const.SONAR_STD, observedDist);
            post = pre * p;
            self.particles[tile] = post;

        for i in range(self.NUM_PARTICLES):
            sample = util.weightedRandomChoice(self.particles);
            new[sample] += 1;
        self.particles = new;
        # raise Exception("Not implemented yet")
        # END_YOUR_CODE
        self.updateBelief()
```

PART 3-2

First, create a dictionary from collections.counter() which is used to store particle probability. Then, get the transition probability by self.transProbdict(). Because there are multiple particles at a particular location, so I need th use self.weightRandomChoice to get one. Finally, add the select one by 1 in new and set self.particles to new.

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