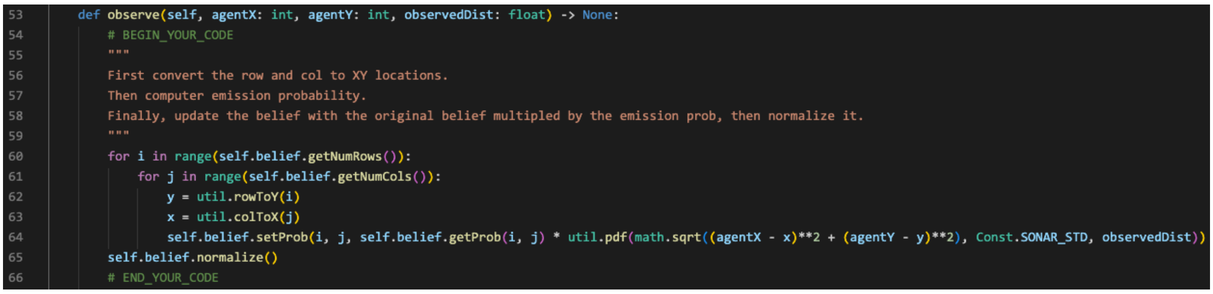
Homework 5: Car Tracking

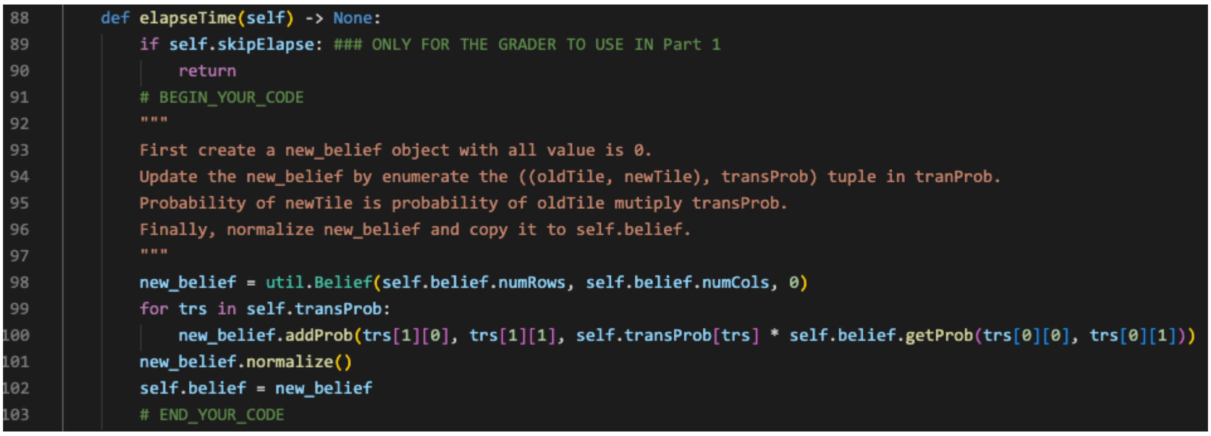
Part I. Implementation (15%):

- Part 1



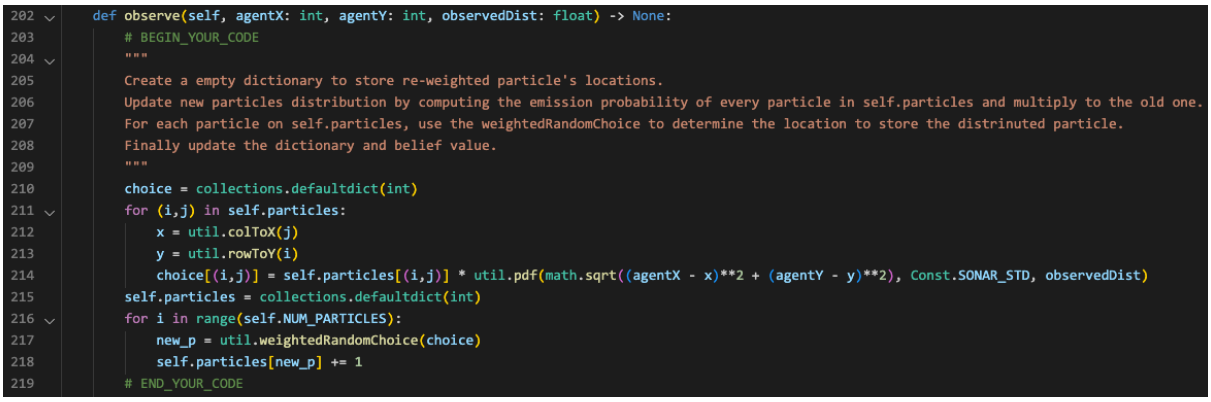
```python
    def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
        # BEGIN_YOUR_CODE
        """
        First convert the row and col to XY locations.
        Then computer emission probability.
        Finally, update the belief with the original belief multipled by the emission prob, then normalize it.
        """
        for i in range(self.belief.getNumRows()):
            for j in range(self.belief.getNumCols()):
                y = util.rowToY(i)
                x = util.colToX(j)
                self.belief.setProb(i, j, self.belief.getProb(i, j) * util.pdf(math.sqrt((agentX - x)**2 + (agentY - y)**2), Const.SONAR_STD, observedDist))
        self.belief.normalize()
        # END_YOUR_CODE
```

- Part 2



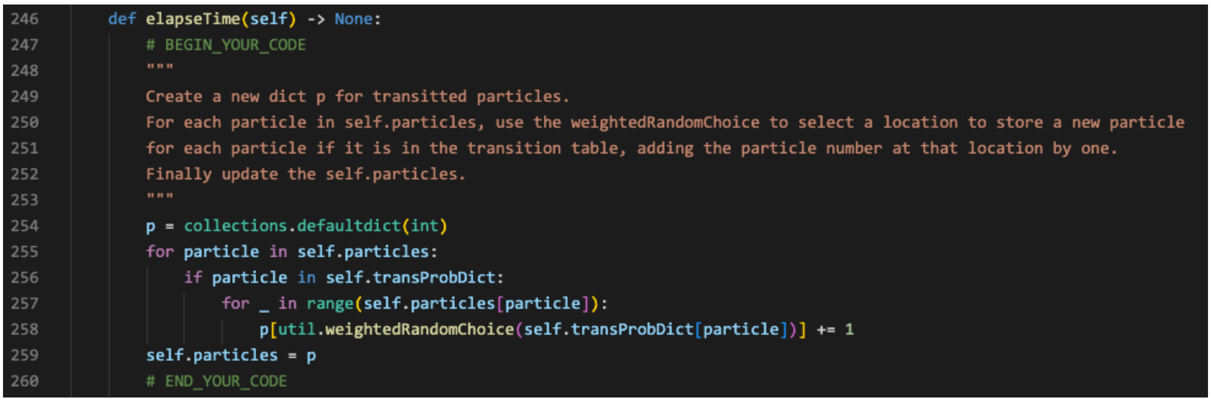
```python
    def elapseTime(self) -> None:
        if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
            return
        # BEGIN_YOUR_CODE
        """
        First create a new_belief object with all value is 0.
        Update the new_belief by enumerate the ((oldTile, newTile), transProb) tuple in tranProb.
        Probability of newTile is probability of oldTile mutiply transProb.
        Finally, normalize new_belief and copy it to self.belief.
        """
        new_belief = util.Belief(self.belief.numRows, self.belief.numCols, 0)
        for trs in self.transProb:
            new_belief.addProb(trs[1][0], trs[1][1], self.transProb[trs] * self.belief.getProb(trs[0][0], trs[0][1]))
        new_belief.normalize()
        self.belief = new_belief
        # END_YOUR_CODE
```

- Part 3-1



```python
    def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
        # BEGIN_YOUR_CODE
        """
        Create a empty dictionary to store re-weighted particle's locations.
        Update new particles distribution by computing the emission probability of every particle in self.particles and multiply to the old one.
        For each particle on self.particles, use the weightedRandomChoice to determine the location to store the distrinuted particle.
        Finally update the dictionary and belief value.
        """
        choice = collections.defaultdict(int)
        for (i,j) in self.particles:
            x = util.colToX(j)
            y = util.rowToY(i)
            choice[(i,j)] = self.particles[(i,j)] * util.pdf(math.sqrt((agentX - x)**2 + (agentY - y)**2), Const.SONAR_STD, observedDist)
        self.particles = collections.defaultdict(int)
        for i in range(self.NUM_PARTICLES):
            new_p = util.weightedRandomChoice(choice)
            self.particles[new_p] += 1
        # END_YOUR_CODE
```

- **Part 3-2**



```
    def elapseTime(self) -> None:
        # BEGIN_YOUR_CODE
        """
        Create a new dict p for transitted particles.
        For each particle in self.particles, use the weightedRandomChoice to select a location to store a new particle
        for each particle if it is in the transition table, adding the particle number at that location by one.
        Finally update the self.particles.
        """
        p = collections.defaultdict(int)
        for particle in self.particles:
            if particle in self.transProbDict:
                for _ in range(self.particles[particle]):
                    p[util.weightedRandomChoice(self.transProbDict[particle])] += 1
        self.particles = p
        # END_YOUR_CODE
```

Part II. Question answering (5%):

What problem I have encountered is that the topic of this homework is the one I have ever met before. I have heard of the Bayesian network before, but this time I have the chance to implement it. I have searched on the topic for a while which cost me some time, but I have learned a lot.