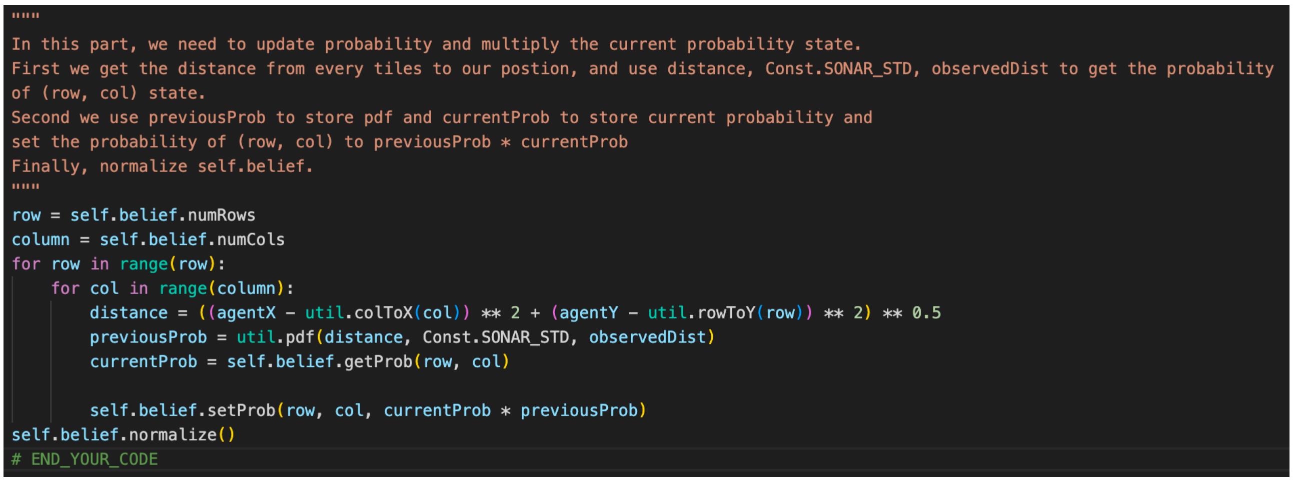
Homework 5: Car Tracking

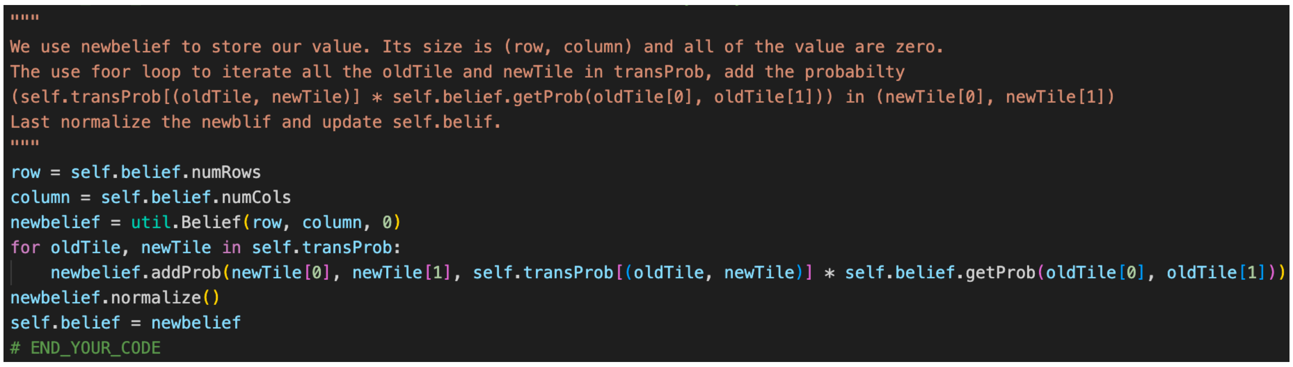
Part I. Implementation (20%):

• Part 1



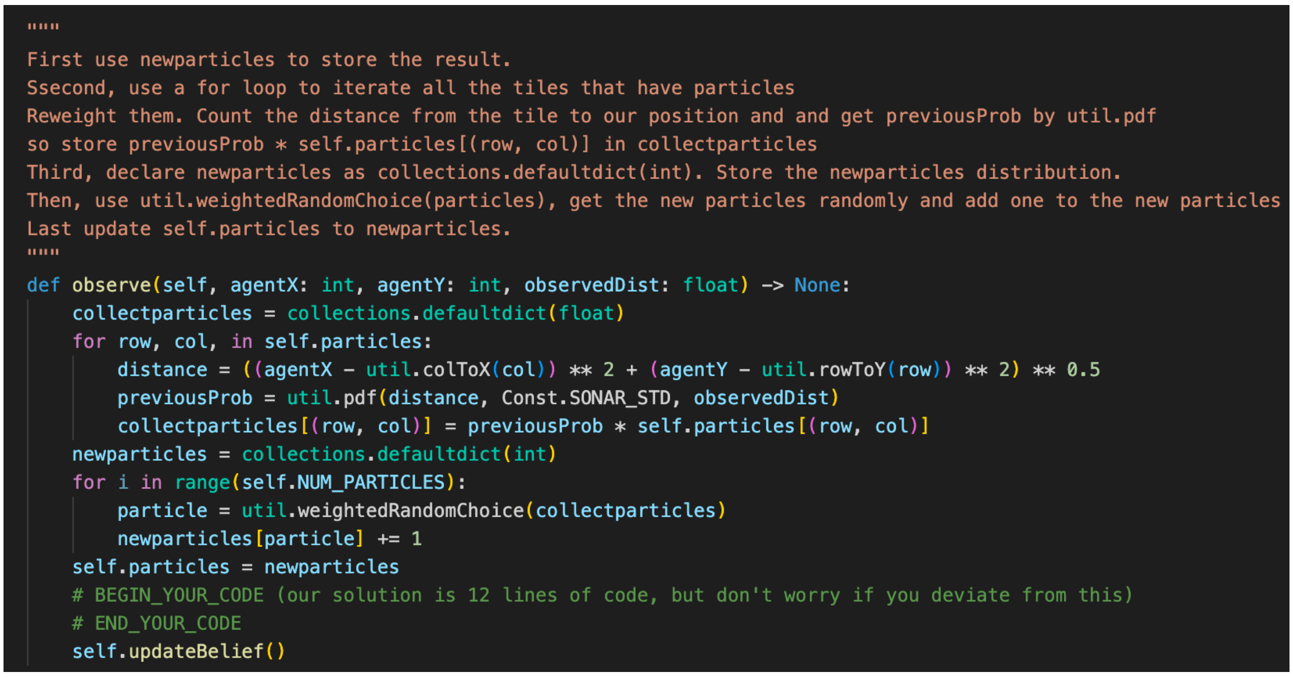
```python
"""
In this part, we need to update probability and multiply the current probability state.
First we get the distance from every tiles to our postion, and use distance, Const.SONAR_STD, observedDist to get the probability
of (row, col) state.
Second we use previousProb to store pdf and currentProb to store current probability and
set the probability of (row, col) to previousProb * currentProb
Finally, normalize self.belief.
"""
row = self.belief.numRows
column = self.belief.numCols
for row in range(row):
    for col in range(column):
        distance = ((agentX - util.colToX(col)) ** 2 + (agentY - util.rowToY(row)) ** 2) ** 0.5
        previousProb = util.pdf(distance, Const.SONAR_STD, observedDist)
        currentProb = self.belief.getProb(row, col)

        self.belief.setProb(row, col, currentProb * previousProb)
self.belief.normalize()
# END_YOUR_CODE
```

• Part 2



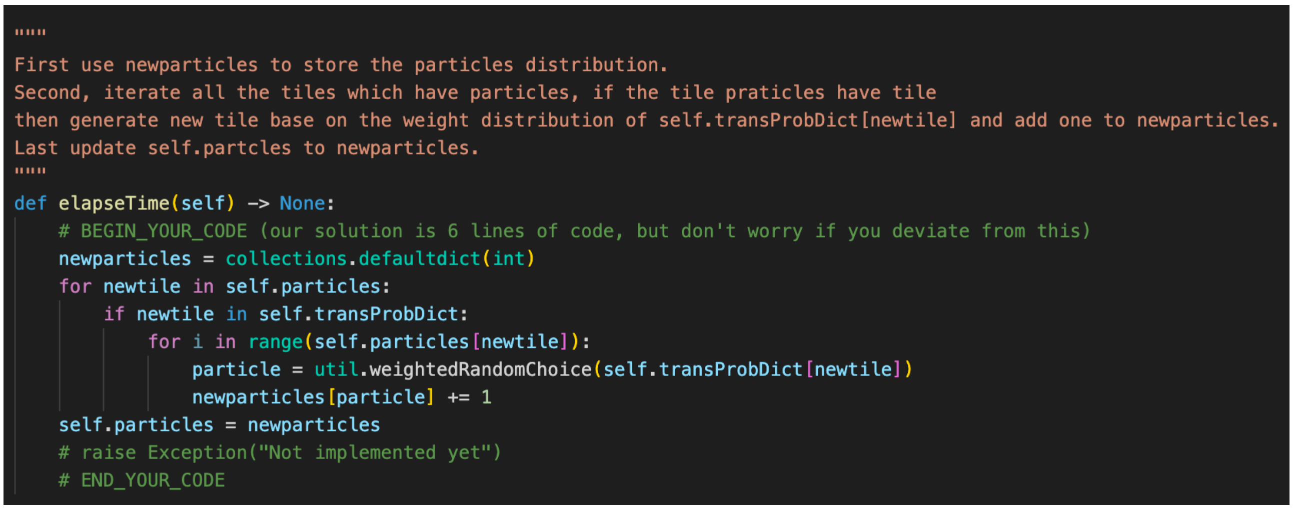
```python
"""
We use newbelief to store our value. Its size is (row, column) and all of the value are zero.
The use foor loop to iterate all the oldTile and newTile in transProb, add the probabilty
(self.transProb[(oldTile, newTile)] * self.belief.getProb(oldTile[0], oldTile[1])) in (newTile[0], newTile[1])
Last normalize the newblif and update self.belif.
"""
row = self.belief.numRows
column = self.belief.numCols
newbelief = util.Belief(row, column, 0)
for oldTile, newTile in self.transProb:
    newbelief.addProb(newTile[0], newTile[1], self.transProb[(oldTile, newTile)] * self.belief.getProb(oldTile[0], oldTile[1]))
newbelief.normalize()
self.belief = newbelief
# END_YOUR_CODE
```

• Part 3-1



```python
"""
First use newparticles to store the result.
Ssecond, use a for loop to iterate all the tiles that have particles
Reweight them. Count the distance from the tile to our position and and get previousProb by util.pdf
so store previousProb * self.particles[(row, col)] in collectparticles
Third, declare newparticles as collections.defaultdict(int). Store the newparticles distribution.
Then, use util.weightedRandomChoice(particles), get the new particles randomly and add one to the new particles
Last update self.particles to newparticles.
"""
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    collectparticles = collections.defaultdict(float)
    for row, col, in self.particles:
        distance = ((agentX - util.colToX(col)) ** 2 + (agentY - util.rowToY(row)) ** 2) ** 0.5
        previousProb = util.pdf(distance, Const.SONAR_STD, observedDist)
        collectparticles[(row, col)] = previousProb * self.particles[(row, col)]
    newparticles = collections.defaultdict(int)
    for i in range(self.NUM_PARTICLES):
        particle = util.weightedRandomChoice(collectparticles)
        newparticles[particle] += 1
    self.particles = newparticles
    # BEGIN_YOUR_CODE (our solution is 12 lines of code, but don't worry if you deviate from this)
    # END_YOUR_CODE
    self.updateBelief()
```

- Part 3-2



```
"""
First use newparticles to store the particles distribution.
Second, iterate all the tiles which have particles, if the tile praticles have tile
then generate new tile base on the weight distribution of self.transProbDict[newtile] and add one to newparticles.
Last update self.partcles to newparticles.
"""
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
    newparticles = collections.defaultdict(int)
    for newtile in self.particles:
        if newtile in self.transProbDict:
            for i in range(self.particles[newtile]):
                particle = util.weightedRandomChoice(self.transProbDict[newtile])
                newparticles[particle] += 1
    self.particles = newparticles
    # raise Exception("Not implemented yet")
    # END_YOUR_CODE
```