交大資工 朱致伶
110550062

Homework 5: Car Tracking

Part I. Implementation (15%):

(Please screenshot your code snippets of Part 1 ~ Part 3, and explain your implementation.)

- Part 1:

```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE
    # Loop through each row and column in the belief grid
    for row in range(self.belief.numRows):
        for col in range(self.belief.numCols):
            # compute the distance between the agent car and the tile
            dist = math.dist((agentX, agentY), (util.colToX(col), util.rowToY(row)))
            # update the probability of the tile in belief
            # by multiplying the old probability with the pdf of the distance
            pdf = util.pdf(dist, Const.SONAR_STD, observedDist)
            old_prob = self.belief.getProb(row, col)
            self.belief.setProb(row, col, old_prob * pdf)
    # Normalize the belief grid after updating all the probabilities.
    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
    # initialize a new belief grid which is numRows by numCols with value=0
    newBeilf = util.Belief(self.belief.getNumRows(), self.belief.getNumCols(), 0)
    # for each tile in the new belief grid, add the probability of all possible old tiles
    for (oldTile, newTile), transProb in self.transProb.items():
        old_prob = self.belief.getProb(oldTile[0], oldTile[1])
        newBeilf.addProb(newTile[0], newTile[1], old_prob * transProb)
    # update the belief grid
    self.belief = newBeilf
    # normalize the new belief grid
    self.belief.normalize()
    # END_YOUR_CODE
```

- Part 3-1:

```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE
    # Re-weight the particles with null dictionary
    reweight = collections.defaultdict()
    # check if the particles are in the grid
    for (row, col), parts_num in self.particles.items():
        # calculate the distance between the agent and the particle
        dist = math.dist((util.colToX(col), util.rowToY(row)), (agentX, agentY))
        # update the weight by multiplying the numbers of particles at the grid square
        # with the pdf of the distance
        reweight[(row, col)] = self.particles[(row, col)] * util.pdf(dist, Const.SONAR_STD, observedDist)

    # Re-sample the particles with null dictionary
    resample = collections.defaultdict()
    # sample the particles from the re-weighted distribution
    for _ in range(self.NUM_PARTICLES):
        particle = util.weightedRandomChoice(reweight)
        if particle in resample:
            resample[particle] += 1
        else:
            resample[particle] = 1
    # update the particles
    self.particles = resample
    # END_YOUR_CODE
    self.updateBelief()
```

- Part 3-2:

```python
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE
    # create a null dictionary to store the proposal at time t+1
    proposal = collections.defaultdict(int)
    # resample the particles from the transition model
    for particle, num in self.particles.items():
        for i in range(num):
            new_particle = util.weightedRandomChoice(self.transProbDict[particle])
            if new_particle in proposal:
                proposal[new_particle] += 1
            else:
                proposal[new_particle] = 1
    # update the particles with the new proposal
    self.particles = proposal
    # END_YOUR_CODE
```

- Result:

```
C:\Users\Lynn\Documents\Sophomore2\Introduction to AI\HW5>python grader.py
========== START GRADING
----- START PART part1-1: part1-1 test for emission probabilities
----- END PART part1-1 [took 0:00:00.004749 (max allowed 5 seconds), 10/10 points]

----- START PART part1-2: part1-2 test ordering of pdf
----- END PART part1-2 [took 0:00:00.006172 (max allowed 5 seconds), 10/10 points]

----- START PART part2: part2 test correctness of elapseTime()
----- END PART part2 [took 0:00:00.003869 (max allowed 5 seconds), 20/20 points]

----- START PART part3-1: part3-1 test for PF observe
----- END PART part3-1 [took 0:00:00.018173 (max allowed 5 seconds), 10/10 points]

----- START PART part3-2: part3-2 test for PF elapseTime
----- END PART part3-2 [took 0:00:00.007521 (max allowed 5 seconds), 10/10 points]

----- START PART part3-3: part3-3 test for PF observe AND elapseTime
----- END PART part3-3 [took 0:00:00.015639 (max allowed 5 seconds), 20/20 points]

========== END GRADING [80/80 points + 0/0 extra credit]
```

Part II. Question answering (5%):

● Please describe problems you met and how you solved the

The most difficult part in the assignment for me is understanding each function in util.py and each classes in submission.py, it took me a long time to realize how to use these functions and classes properly with no logic, syntax, attributes errors. Below are some of the errors I get stuck in when writing the assignment:

- Problem 1:
  For ParticleFilter class, when reweighting the particles based on the observation, resampling the particles and updating proposal based on the particle distribution at current time t, all of the 3 actions must declare a new dictionary type parameters to store the new value before updating, just as the hint mentioned given by Tas.

  ```
  - In order to work with the grader, you must create a new dictionary when you are
    re-sampling the particles, then set self.particles equal to the new dictionary at the end.
  ```

  However, I didn't notice at first, assigning the new value to the same register as the old ones, which took me a while to realize the problem and solutions.

- Problem 2:

  ```
  AttributeError: 'collections.defaultdict' object has no attribute
  'item'. Did you mean: 'items'?
  ```

  Since I'm not familiar with the collections.defaultdict datatype, I didn't know how to loop through all the items, which make me utilize the items function wrong.