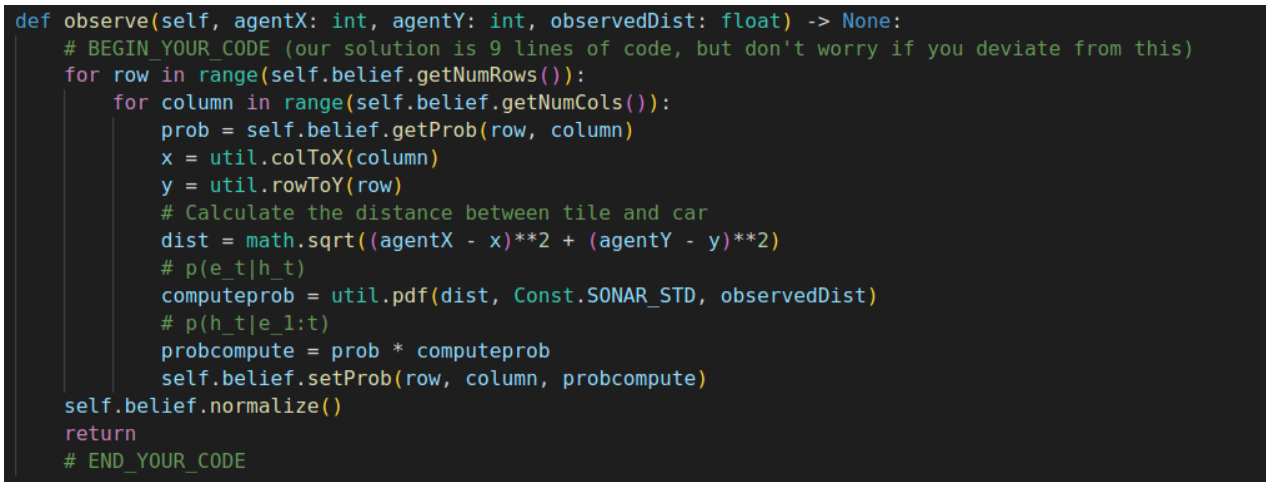
109550073 陳宥安

Homework 5: Car Tracking

Please keep the title of each section and delete examples.

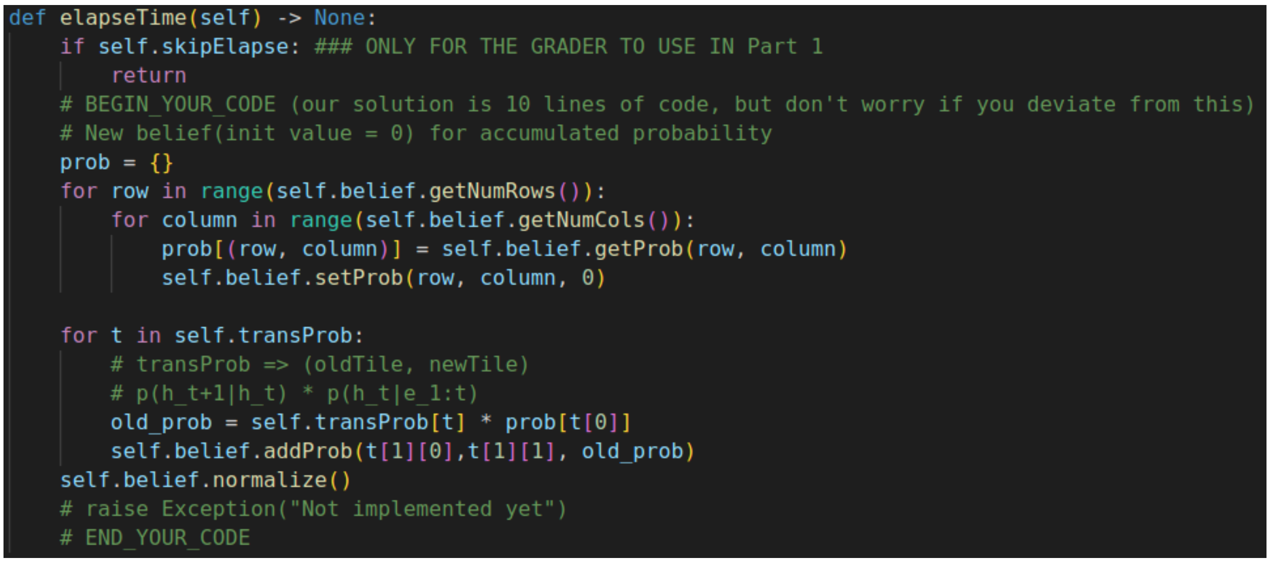
Part I. Implementation (20%):

- 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 (our solution is 9 lines of code, but don't worry if you deviate from this)
    for row in range(self.belief.getNumRows()):
        for column in range(self.belief.getNumCols()):
            prob = self.belief.getProb(row, column)
            x = util.colToX(column)
            y = util.rowToY(row)
            # Calculate the distance between tile and car
            dist = math.sqrt((agentX - x)**2 + (agentY - y)**2)
            # p(e_t|h_t)
            computeprob = util.pdf(dist, Const.SONAR_STD, observedDist)
            # p(h_t|e_1:t)
            probcompute = prob * computeprob
            self.belief.setProb(row, column, probcompute)
    self.belief.normalize()
    return
    # END_YOUR_CODE
```

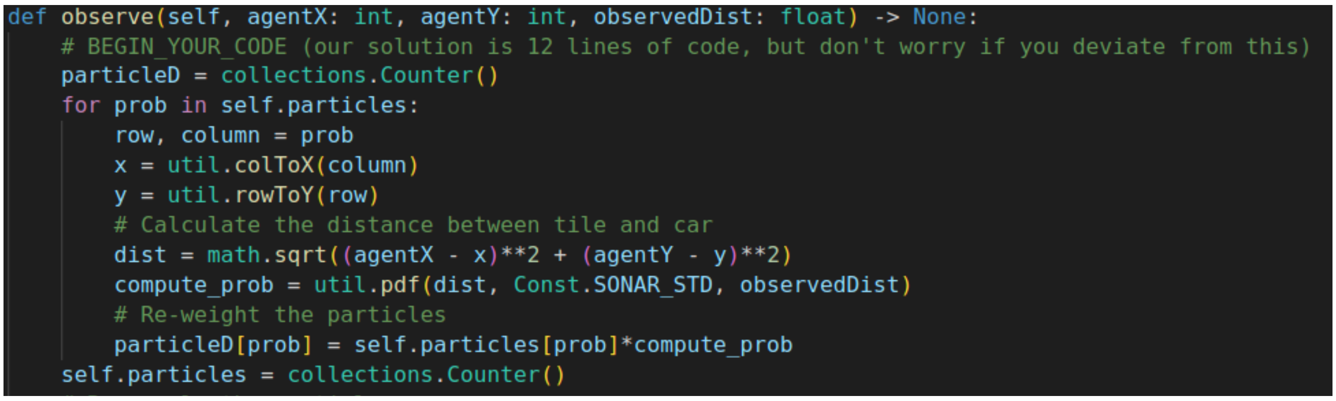
- Part 2



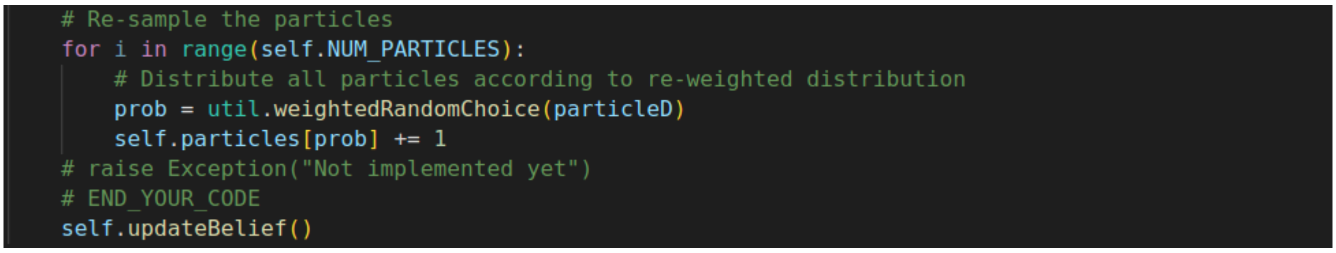
```python
def elapseTime(self) -> None:
    if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
        return
    # BEGIN_YOUR_CODE (our solution is 10 lines of code, but don't worry if you deviate from this)
    # New belief(init value = 0) for accumulated probability
    prob = {}
    for row in range(self.belief.getNumRows()):
        for column in range(self.belief.getNumCols()):
            prob[(row, column)] = self.belief.getProb(row, column)
            self.belief.setProb(row, column, 0)

    for t in self.transProb:
        # transProb => (oldTile, newTile)
        # p(h_t+1|h_t) * p(h_t|e_1:t)
        old_prob = self.transProb[t] * prob[t[0]]
        self.belief.addProb(t[1][0],t[1][1], old_prob)
    self.belief.normalize()
    # raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

- Part3-1

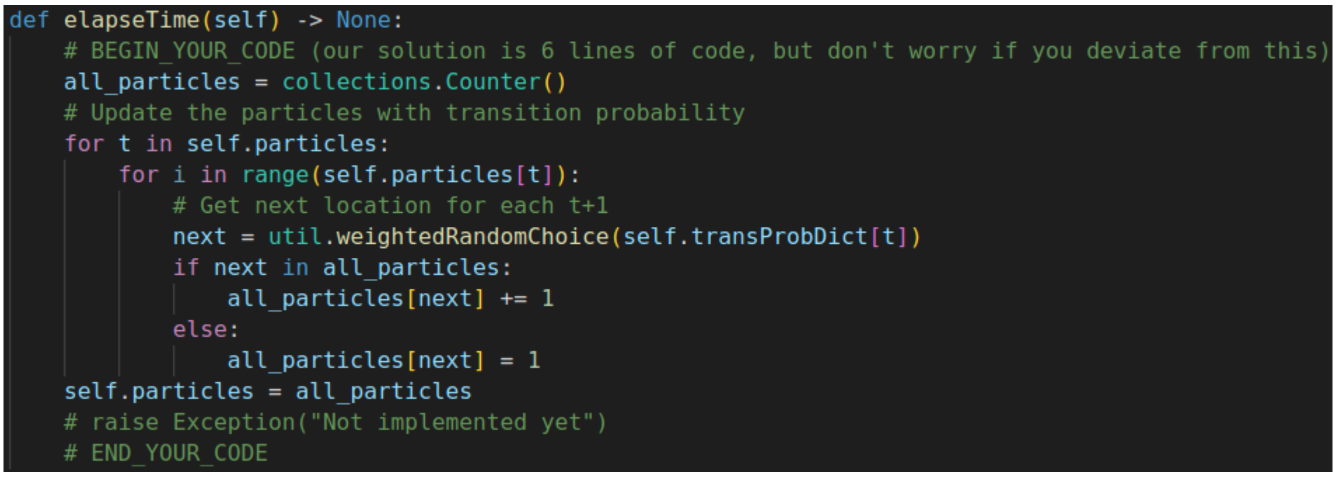


```python
def observe(self, agentX: int, agentY: int, observedDist: float) -> None:
    # BEGIN_YOUR_CODE (our solution is 12 lines of code, but don't worry if you deviate from this)
    particleD = collections.Counter()
    for prob in self.particles:
        row, column = prob
        x = util.colToX(column)
        y = util.rowToY(row)
        # Calculate the distance between tile and car
        dist = math.sqrt((agentX - x)**2 + (agentY - y)**2)
        compute_prob = util.pdf(dist, Const.SONAR_STD, observedDist)
        # Re-weight the particles
        particleD[prob] = self.particles[prob]*compute_prob
    self.particles = collections.Counter()
```



```
    # Re-sample the particles
    for i in range(self.NUM_PARTICLES):
        # Distribute all particles according to re-weighted distribution
        prob = util.weightedRandomChoice(particleD)
        self.particles[prob] += 1
    # raise Exception("Not implemented yet")
    # END_YOUR_CODE
    self.updateBelief()
```

- **Part3-2**



```
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
    all_particles = collections.Counter()
    # Update the particles with transition probability
    for t in self.particles:
        for i in range(self.particles[t]):
            # Get next location for each t+1
            next = util.weightedRandomChoice(self.transProbDict[t])
            if next in all_particles:
                all_particles[next] += 1
            else:
                all_particles[next] = 1
    self.particles = all_particles
    # raise Exception("Not implemented yet")
    # END_YOUR_CODE
```