109550182 莊婕妤

AI Homework5 Report

Part I. Implementation (20%)

Part 1

```
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          """ The other car is stationary: H_t = H_t-1 for all t """
56          for row in range(self.belief.numRows):
57              for col in range(self.belief.numCols):
58                  # Calculate the distance between the tile and my car
59                  distance = math.sqrt((util.colToX(col) - agentX) ** 2 + (util.rowToY(row) - agentY) ** 2)
60                  # Calculate p(e_t|h_t)
61                  pdf = util.pdf(distance, Const.SONAR_STD, observedDist)
62                  # Update the probability with respect to the observation
63                  # P(H_t|e_1:t) = p(e_t|h_t) * P(H_t|e_1:t-1)
64                  updated_probability = pdf * self.belief.getProb(row, col)
65                  self.belief.setProb(row, col, updated_probability)
66          # Normalize the belief
67          self.belief.normalize();
68          # END_YOUR_CODE
```

Part 2

```
90      def elapseTime(self) -> None:
91          if self.skipElapse: ### ONLY FOR THE GRADER TO USE IN Part 1
92              return
93          # BEGIN_YOUR_CODE (our solution is 10 lines of code, but don't worry if you deviate from this)
94          # Declare a new belief for counting accumulated probability
95          new_belief = util.Belief(self.belief.numRows, self.belief.numCols, value=0)
96          # Accumulating the probability
97          for (oldTile, newTile) in self.transProb:
98              # P(h_t|e_1:t) * p(h_t+1|h_t)
99              reweighted = self.belief.getProb(*oldTile) * self.transProb[(oldTile, newTile)]
100             new_belief.addProb(newTile[0], newTile[1], reweighted)
101         # Normalize the belief
102         new_belief.normalize()
103         # Update the belief
104         self.belief = new_belief
105         # END_YOUR_CODE
```

109550182 莊婕妤

Part 3-1

```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)
        # Store reweighted particle distribution to a dictionary
        reweighted = collections.defaultdict(float)
        # Reweight the particle distribution with emission probability p(e_t|h_t)
        for (row, col) in self.particles:
            distance = math.sqrt((util.colToX(col) - agentX) ** 2 + (util.rowToY(row) - agentY) ** 2)
            pdf = util.pdf(distance, Const.SONAR_STD, observedDist)
            # Update the probability
            reweighted[(row, col)] = self.particles[(row, col)] * pdf
        # Store resampled particles to a dictionary
        new_particles = collections.defaultdict(int)
        # Resample particles
        for _ in range(self.NUM_PARTICLES):
            # Distribute to new particles according to the reweighted distribution
            particle = util.weightedRandomChoice(reweighted)
            new_particles[particle] += 1
        self.particles = new_particles

        # END_YOUR_CODE
        self.updateBelief()
```

Part 3-2

```python
    def elapseTime(self) -> None:
        # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
        # Store new particles to a dictionary
        new_particles = collections.defaultdict(int)
        # Update the particles with transition probability in each tile
        for particle in self.particles:
            for _ in range(self.particles[particle]):
                # Get the next locations for each particle at t+1
                new_particle = util.weightedRandomChoice(self.transProbDict[particle])
                new_particles[new_particle] += 1
        self.particles = new_particles
        # END_YOUR_CODE
```