Homework 5: Car Tracking

Report 109550159 李驊恩

Part1:

```
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)
    NumofRows = self.belief.getNumRows()     # Get the numbers of rows and columns
    NumofColumns = self.belief.getNumCols()
    for row in range(NumofRows):             # Discretize the world into tiles first
        for column in range(NumofColumns): # represented by (row, column) pairs
            # For each tile store a probability representing our
            # belief that there's a car on that tile
            P = self.belief.getProb(row,column)
            # Then convert from a tile to a location.
            x_coordinate = util.colToX(column)
            y_coordinate = util.rowToY(row)
            # Compute the distance between the agent and other cars.
            distance = math.sqrt((agentX - x_coordinate)**2 + (agentY - y_coordinate)**2)
            # Gaussian random variable with mean equal to the true
            # distance betweenyour car and the other car.
            Et = util.pdf(distance, Const.SONAR_STD, observedDist)
            PP = P * Et
            self.belief.setProb(row,column,PP)
    # Normalize the posterior probability.
    self.belief.normalize()

    # END_YOUR_CODE
```

Part2:

```
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
    _Belief_ = util.Belief(self.belief.numRows, self.belief.numCols, value = 0)
    # Get the transition probabilities
    for old_tile, new_tile in self.transProb:
        # Returns the belief for tile row, column
        OLD = self.belief.getProb(*old_tile)
        P = OLD * self.transProb[(old_tile, new_tile)]
        #  Use the addProb to modify and access the probabilities
        _Belief_.addProb(new_tile[0], new_tile[1], P)
    # Reduce the belief and normalize the posterior probability.
    self.belief = _Belief_
    self.belief.normalize()

    # END_YOUR_CODE
```

Part3:

```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)
    # Re-weight the particles based on the observation.
    A = collections.Counter()
    for row,column in self.particles:
        x_coordinate = util.colToX(column)
        y_coordinate = util.rowToY(row)
        distance = math.sqrt((x_coordinate - agentX)**2 + (y_coordinate - agentY)**2)
        # Compute Gaussian random variable with mean equal to the true
        # distance betweenyour car and the other car.
        PD = util.pdf(distance,Const.SONAR_STD, observedDist)
        A[(row,column)] = PD * self.particles[(row,column)]
    # New particles
    # (Re-sample the particles)
    _particles_ = collections.Counter()
    for i in range(self.NUM_PARTICLES):
        # Give the weight to elements in A by "weightedRandomChoice" function
        P = util.weightedRandomChoice(A)
        _particles_[P] += 1
    self.particles = _particles_
    # END_YOUR_CODE
    self.updateBelief()
```

```python
def elapseTime(self) -> None:
    # BEGIN_YOUR_CODE (our solution is 6 lines of code, but don't worry if you deviate from this)
    # New particles
    new_particles = collections.Counter()
    for i in self.particles:
        for j in range(self.particles[i]):
            # particle distribution at current time t
            particle = util.weightedRandomChoice(self.transProbDict[i])
            # Propose the particle distribution at time t+1
            if particle in new_particles:
                new_particles[particle] = new_particles[particle] + 1
            else:
                new_particles[particle] = 1
    # sampling
    self.particles = new_particles
    # END_YOUR_CODE
```