Homework 5: Car Tracking

109550134梁詠晴

Part I. Implementation (20%):

Part 1:

```
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)
    #raise Exception("Not implemented yet")
    for r in range (self.belief.numRows):
        for c in range (self.belief.numCols):#for each place in grid
            temp = pow((util.colToX(c)-agentX),2)+pow((util.rowToY(r)-agentY),2)
            mycar_distance = math.sqrt(temp) #dist of the grid to my car
            pdf_calculated = util.pdf(mycar_distance,Const.SONAR_STD,observedDist) # mean : mycar_distance, sonar_std : std,
            cur_prob = self.belief.getProb(r,c)                                    # value : observedDist
            self.belief.setProb(r,c,cur_prob* pdf_calculated) #update probability
    self.belief.normalize()#normalize self.belief

    # END_YOUR_CODE
```

Part 2:

```
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)
    newBelief = util.Belief(self.belief.numRows, self.belief.numCols, value=0) # new belief for update with default all 0
    for old,new in self.transProb:
        old_r,old_c = old # old col & row
        new_r,new_c = new # new col & row
        cur_prob = self.belief.getProb(old_r,old_c) # get current probability of current(old) row & col
        trans_prob = self.transProb[(old, new)] # get transprob with (old,new) pair
        newBelief.addProb(new_r, new_c,  cur_prob * trans_prob) # update probability with new location and delta(cur_prob*trans_prob)
    newBelief.normalize()
    self.belief = newBelief # update normalized belief
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE
```

Part 3-1:

```
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)
    temp_particles = collections.defaultdict(float) # create new dictionary to store current particles calculated with
    for r, c in self.particles:
        temp = pow((util.colToX(c)-agentX),2)+pow((util.rowToY(r)-agentY),2)
        mycar_distance = math.sqrt(temp) #dist of the grid to my car
        pdf_calculated = util.pdf(mycar_distance,Const.SONAR_STD,observedDist) # mean: mycar_distance, sonar_std: std, value: observedDist
        temp_particles[(r, c)] = self.particles[(r, c)] * pdf_calculated # updete new dictionary with current particle*pdf
    newParticles = collections.defaultdict(int) # create new dictionary for new particles

    for i in range(self.NUM_PARTICLES):
        particle = util.weightedRandomChoice(temp_particles) #new NUM_PARTICLES sampled from the new re-weighted distribution
        newParticles[particle] += 1 # dict : add 1 of val which index = particle
    self.particles = newParticles # update new particles
    #raise Exception("Not implemented yet")
    # 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)
    newParticles = collections.defaultdict(int) # create new dictionary for new particles
    for particle in self.particles:# loop over particles
        val = self.particles[particle] # corresponding particles at the location
        for i in range(val):  #call weightedRandomChoice for every particles at the location
            new_t = self.transProbDict[particle] #self.transProbDict[oldTile][newTile], particle = oldtile; new_t = newtile(new weight dict)
            temp_particle = util.weightedRandomChoice(new_t)# weightedRandomChoice based on new weight dict
            newParticles[temp_particle] += 1  # dict : add 1 of val which index = temp_particle
    self.particles = newParticles #update particles
    #raise Exception("Not implemented yet")
    # END_YOUR_CODE
```