Homework5-report

- Part 1. Implementation (descriptions are in the images)

1. 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)
    """update P(H_t|E) according to P(E_t|H_t)"""
    for row in range(self.belief.getNumRows()):
        for col in range(self.belief.getNumCols()):

            """calculate the distance between the target tile and current agent tile"""
            dist = math.sqrt((util.colToX(col) - agentX) ** 2 + (util.rowToY(row) - agentY) ** 2)

            """calculate P(E_t|H_t)"""
            prob = util.pdf(dist, Const.SONAR_STD, observedDist)

            """update the P(H_t|E) according to current observation"""
            post = self.belief.getProb(row, col)
            self.belief.setProb(row, col, prob * post)

    """don't forget to normalize"""
    self.belief.normalize()
    # END_YOUR_CODE
```

2. 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"""
    tmp = util.Belief(self.belief.getNumRows(), self.belief.getNumCols(), 0)
    for trans in self.transProb.items():
        """tiles before and after transition"""
        old, new = trans[0]

        """transition probability"""
        prob = trans[1]

        """update P(H_t+1|E) according to P(H_t|E)"""
        post = self.belief.getProb(old[0], old[1])
        tmp.addProb(new[0], new[1], post * prob)

    """update and normalize the belief"""
    self.belief = tmp
    self.belief.normalize()
    # END_YOUR_CODE
```

3. 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)
    """calculate P(H_t|E_t) according to E_t and P(E_t|H_t)"""
    """updated P(H_t|E_t) dict"""
    particlesProb = dict()
    for tile, count in self.particles.items():
        """if there is no particle"""
        if self.particles[tile] == 0:
            continue

        """calculate the distance and probability"""
        dist = math.sqrt((agentX - util.colToX(tile[1])) ** 2 + (agentY - util.rowToY(tile[0])) ** 2)
        prob = util.pdf(dist, Const.SONAR_STD, observedDist)

        """P(particle on the tile) = P(particle is on the tile|E_t) * #(sampled particles on the tile)"""
        particlesProb[tile] = prob * count

    newParticles = collections.defaultdict(int)
    for _ in range(self.NUM_PARTICLES):
        """resample the particles"""
        newParticles[util.weightedRandomChoice(particlesProb)] += 1
    self.particles = newParticles
    # END_YOUR_CODE
    self.updateBelief()
```

4. 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)

    """for the particles on each tile"""
    for oldTile, count in self.particles.items():
        if oldTile not in self.transProbDict:
            continue

        """get the transition probability of each tile at time t+1 according to time t"""
        probs = self.transProbDict[oldTile]

        for _ in range(count):
            """randomly choose new tile for each particle on the tile at time t"""
            newParticles[util.weightedRandomChoice(probs)] += 1
    self.particles = newParticles
    # END_YOUR_CODE
```