گزارش پروژه چهارم

اشکان شکیبا (9931030)

بخش اول

```
if ghostPosition == jailPosition:
    if noisyDistance is None:
        return 1.0
    else:
        return 0.0
if noisyDistance is None:
    return 0.0
return busters.getObservationProbability(noisyDistance,
manhattanDistance(pacmanPosition, ghostPosition))
```

در این تابع، ابتدا بررسی می‌شود که اگر روح در زندان است، در صورت None بودن noisy distance مقدار یک و در غیر این صورت مقدار صفر بازگردانی شود. اگر روح در زندان نباشد و noisy distance برابر None باشد صفر بازگردانی شده و در غیر این صورت حاصل تابع احتمال مشاهده شکارچی‌ها بر اساس فاصله منهتنی پک‌من و روح (معادل مقدار P(noisyDistance | pacmanPosition, ghostPosition) در داکیومنت پروژه) بازگردانی می‌شود.

بخش دوم

```
def observeUpdate(self, observation, gameState):
    """
    Update beliefs based on the distance observation and Pacman's
position.

    The observation is the noisy Manhattan distance to the ghost you are
    tracking.

    self.allPositions is a list of the possible ghost positions, including
    the jail position. You should only consider positions that are in
    self.allPositions.

    The update model is not entirely stationary: it may depend on Pacman's
    current position. However, this is not a problem, as Pacman's current
    position is known.
    """

    for ghost_position in self.allPositions:
        self.beliefs[ghost_position] *= self.getObservationProb(
            observation, gameState.getPacmanPosition(), ghost_position,
self.getJailPosition())

    self.beliefs.normalize()
```

در این تابع با پیمایش بر روی موقعیت همه روح‌ها، احتمال با تابع احتمال مشاهده و بر اساس موقعیت پک‌من، روح مورد بررسی و زندان محاسبه شده و در فیلد گفته شده در داکیومنت پروژه، ضرب و ذخیره می‌شود.

بخش سوم

```
def elapseTime(self, gameState):
    """
    Predict beliefs in response to a time step passing from the current
    state.

    The transition model is not entirely stationary: it may depend on
    Pacman's current position. However, this is not a problem, as Pacman's
    current position is known.
    """

    discrete_distribution = DiscreteDistribution()

    for position in self.allPositions:
        new_position = self.getPositionDistribution(gameState, position)
        for key in new_position:
            discrete_distribution[key] += self.beliefs[position] *
new_position[key]

    self.beliefs = discrete_distribution
```

در این تابع ابتدا یک شی از کلاس DiscreteDistribution ساخته شده و سپس با پیمایش بر روی موقعیت‌های قبلی روح و بررسی آنها با تابع getPositionDistribution، موقعیت جدید به شکل یک دیکشنری ذخیره می‌شود که با دادن هر موقعیت به آن، احتمال حضور روح در آن موقعیت در زمان t+1 بازگردانی می‌شود.

با پیمایش بر روی این دیکشنری، مقادیر شی ساخته شده بر اساس حاصل‌ضرب مقادیر پیش‌تر ذخیره شده در مقادیر جدید، تکمیل می‌شوند و در نهایت این شی جایگزین شی مربوط به مقادیر قبلی کلاس می‌شود.

بخش چهارم

```
def chooseAction(self, gameState):
    """
    First computes the most likely position of each ghost that has
    not yet been captured, then chooses an action that brings
    Pacman closest to the closest ghost (according to mazeDistance!).
    """
    pacmanPosition = gameState.getPacmanPosition()
    legal = [a for a in gameState.getLegalPacmanActions()]
    livingGhosts = gameState.getLivingGhosts()
    livingGhostPositionDistributions = [
        beliefs
        for i, beliefs in enumerate(self.ghostBeliefs)
        if livingGhosts[i + 1]
    ]

    ghost_positions = []
    for position in livingGhostPositionDistributions:
        ghost_positions.append(max(position, key=position.get))

    best_action = None
    for position in ghost_positions:
        for action in legal:
            if self.distancer.getDistance(pacmanPosition, position) > \

self.distancer.getDistance(Actions.getSuccessor(pacmanPosition, action),
position):
                best_action = action
        return best_action
```

در این تابع ابتدا با پیمایش بر روی موقعیت روح‌ها، لیستی از موقعیت روح‌ها با بالاترین احتمال ساخته شده و سپس با بررسی موقعیت همه روح‌ها، در صورتی که فاصله تا آن روح بیشتر از فاصله عمل گرفته شده از تابع successor باشد، آن عمل را به عنوان بهترین گزینه تا آن لحظه ذخیره می‌کنیم و جستجو را ادامه می‌دهیم و در نهایت بهترین عمل ذخیره شده را بازگردانی می‌کنیم.

بخش پنجم

```
class ParticleFilter(InferenceModule):
    """
    A particle filter for approximately tracking a single ghost.
    """

    def __init__(self, ghostAgent, numParticles=300):
        InferenceModule.__init__(self, ghostAgent)
        self.setNumParticles(numParticles)

    def setNumParticles(self, numParticles):
        self.numParticles = numParticles

    def initializeUniformly(self, gameState):
        """
        Initialize a list of particles. Use self.numParticles for the
number of
        particles. Use self.legalPositions for the legal board positions
where
        a particle could be located. Particles should be evenly (not
randomly)
        distributed across positions in order to ensure a uniform prior.
Use
        self.particles for the list of particles.
        """
        self.particles = []

        for i in range(self.numParticles):
            self.particles.append(self.legalPositions[i %
len(self.legalPositions)])

    def observeUpdate(self, observation, gameState):
        """
        Update beliefs based on the distance observation and Pacman's
position.

        The observation is the noisy Manhattan distance to the ghost you
are
        tracking.

        There is one special case that a correct implementation must
handle.
        When all particles receive zero weight, the list of particles
should
        be reinitialized by calling initializeUniformly. The total method
of
        the DiscreteDistribution may be useful.
        """
        "*** YOUR CODE HERE ***"
        raiseNotDefined()
```

```
    def elapseTime(self, gameState):
        """
        Sample each particle's next state based on its current state and
the
        gameState.
        """
        "*** YOUR CODE HERE ***"
        raiseNotDefined()

    def getBeliefDistribution(self):
        """
        Return the agent's current belief state, a distribution over ghost
        locations conditioned on all evidence and time passage. This
method
        essentially converts a list of particles into a belief
distribution.

        This function should return a normalized distribution.
        """

        distribution = Counter()
        for particle in self.particles:
            distribution[particle] += 1
        distribution.normalize()
        return distribution
```

در تابع initializeUniformly هر بار ذره‌ای با انتخاب موقعیتی از لیست موقعیت‌های legal ساخته می‌شود که برای محاسبه ایندکس آن باقیمانده شمارشگر حلقه را بر تعداد موقعیت‌ها به دست می‌آوریم و ذره ساخته شده را در لیست ذرات ذخیره می‌کنیم.

در تابع getBeliefDistribution با بررسی موقعیت همه روح‌ها، به ازای هر ذره distribution بروزرسانی شده و در نهایت نرمالایز و بازگردانی می‌شود.

بخش ششم

```
def observeUpdate(self, observation, gameState):
    """
    Update beliefs based on the distance observation and Pacman's
position.

    The observation is the noisy Manhattan distance to the ghost you are
    tracking.

    There is one special case that a correct implementation must handle.
    When all particles receive zero weight, the list of particles should
    be reinitialized by calling initializeUniformly. The total method of
    the DiscreteDistribution may be useful.
    """

    discrete_distribution = DiscreteDistribution()

    for particle in self.particles:
        discrete_distribution[particle] += self.getObservationProb(
            observation, gameState.getPacmanPosition(), particle,
self.getJailPosition())

    if discrete_distribution.total() > 0:
        discrete_distribution.normalize()
        self.beliefs = discrete_distribution

        for i in range(self.numParticles):
            self.particles[i] = discrete_distribution.sample()
    else:
        self.initializeUniformly(gameState)
```

در این تابع ابتدا یک شی از کلاس DiscreteDistribution ساخته شده و سپس با پیمایش بر روی ذرات، مقدار آن ذره با حاصل تابع احتمال مشاهده بر اساس موقعیت پک‌من و زندان جمع می‌شود.

پس از آن اگر مجموع آنها بیشتر از صفر باشد وزن‌ها نرمالایز شده و به هر ذره یک sample از distribution مربوطه اختصاص داده می‌شود؛ و اگر مجموع آنها صفر باشد (به این معنا که ذره‌ای وجود ندارد) تابع initializeUniformly را فراخوانی می‌شود.

بخش هفتم

```
def elapseTime(self, gameState):
    """
    Sample each particle's next state based on its current state and the
    gameState.
    """

    particles = {}
    for i in range(self.numParticles):
        oldPos = self.particles[i]
        if oldPos in particles:
            self.particles[i] = particles[oldPos].sample()
        else:
            newPosDist = self.getPositionDistribution(gameState, oldPos)
# same as document
            self.particles[i] = newPosDist.sample()
            particles[oldPos] = newPosDist
```

در این تابع ابتدا یک دیکشنری ساخته می‌شود که حالت بعدی ذرات را به همراه sample آنها ذخیره کند. سپس با بررسی همه ذرات ذخیره شده، اگر ذره در دیکشنری جدید وجود داشته باشد مقدار آن با sample مربوط به حالت بعدی بروزرسانی می‌شود، و اگر ذره در دیکشنری جدید وجود نداشته باشد یک sample بر اساس یک distribution مربوط به موقعیت قبلی ساخته شده و هم در دیکشنری جدید و هم در مقادیر ذخیره شده قبلی افزوده می‌شود.

بخش هشتم

```
def initializeUniformly(self, gameState):
    """
    Initialize particles to be consistent with a uniform prior. Particles
    should be evenly distributed across positions in order to ensure a
    uniform prior.
    """
    self.particles = []

    permutations = list(itertools.product(self.legalPositions,
repeat=self.numGhosts))
    random.shuffle(permutations)

    i = 0
    while i < self.numParticles:
        for particle in permutations:
            self.particles.append(particle)
            i += 1
```

در این تابع ابتدا ضرب کارتزین همه موقعیت‌ها و تعداد روح‌ها انجام شده و جایگشت حاصل به شکل تصادفی shuffle می‌شود. سپس در یک حلقه، ذرات هر جایگشت به لیست ذرات افزوده می‌شوند.

بخش نهم

```
def observeUpdate(self, observation, gameState):
    """
    Update beliefs based on the distance observation and Pacman's
position.

    The observation is the noisy Manhattan distances to all ghosts you
    are tracking.

    There is one special case that a correct implementation must handle.
    When all particles receive zero weight, the list of particles should
    be reinitialized by calling initializeUniformly. The total method of
    the DiscreteDistribution may be useful.
    """

    discrete_distribution = DiscreteDistribution()

    for particle in self.particles:
        probability = 1
        for i in range(self.numGhosts):
            probability *= self.getObservationProb(
                observation[i], gameState.getPacmanPosition(),
particle[i], self.getJailPosition(i))
        discrete_distribution[particle] += probability

    particles_count = len(self.particles)
    self.particles = []

    if discrete_distribution.total() > 0:
        while particles_count > 0:
            particle = discrete_distribution.sample()
            self.particles.append(particle)
            particles_count -= 1
    else:
        self.initializeUniformly(gameState)
```

در این تابع ابتدا یک شی از کلاس DiscreteDistribution ساخته شده و سپس با پیمایش بر روی ذرات مربوط به موقعیت روح‌ها، احتمال مشاهده بر اساس موقعیت روح (ذره)، موقعیت پک‌من و زندان و فاصله منهتنی مربوط به روح که در observation ذخیره شده محاسبه می‌شود و distribution بر اساس احتمال محاسبه شده بروزرسانی می‌گردد.

در ادامه تعداد ذرات ذخیره شده و لیست ذرات دوباره ساخته می‌شود. اگر مجموع discrete distribution از صفر بیشتر باشد در حلقه‌ای sample ذرات

ساخته شده و به لیست افزوده می‌شود؛ و اگر مجموع صفر باشد (به این معنا که ذره‌ای وجود ندارد) تابع initializeUniformly را فراخوانی می‌شود.

بخش دهم

```
def elapseTime(self, gameState):
    """
    Sample each particle's next state based on its current state and the
    gameState.
    """
    newParticles = []
    for oldParticle in self.particles:
        newParticle = list(oldParticle)  # A list of ghost positions

        # now loop through and update each entry in newParticle...

        for i in range(self.numGhosts):
            # same as document
            prevGhostPositions = oldParticle
            newPosDist = self.getPositionDistribution(gameState,
prevGhostPositions, i, self.ghostAgents[i])
            newParticle[i] = newPosDist.sample()

        newParticles.append(tuple(newParticle))
    self.particles = newParticles
```

در این تابع با پیمایش بر روی روح‌ها، موقعیت جدید بر اساس موقعیت همه روح‌ها در زمان قبلی محاسبه شده و بر اساس آن sample مربوطه ساخته می‌شود و در لیست newParticles ذخیره می‌گردد.

سوال)

در تست ۱، محاسبه predict تنها در لحظه انجام می‌شود؛ در حالی که در تست ۳ در لحظات متفاوت و متوالی انجام می‌گردد.