Assignment 3
Bayesian and Decision Networks

Mehmet Enes Erciyes, 68906
Computer Engineering

Question 1

In Q3 tests, the elapse time step of the exact inference is tested. By looking at the .test files, we can see that there is no observation update in these tests:

```
16
17    observe: "False"
18
19    elapse: "True"
20
```

In elapse time, the beliefs(probability that the ghost is in a position) are updated according to the **transition model of the HMM.** In our case, this transition model is the model of the ghost's movement. By implementing this step, we can incorporate our knowledge of how the ghost moves to our inferences.

The reason that the probabilities settle even though the ghost is moving is that (1) Pacman does not update its beliefs according to its observation and (2) without the observation, HMM turns to a normal Markov process and Markov processes converge to a steady state distribution.

The two ghosts can be told apart by looking at the steady state distributions. In the first case, a random ghost is used, and the steady state distribution is **uniform.**

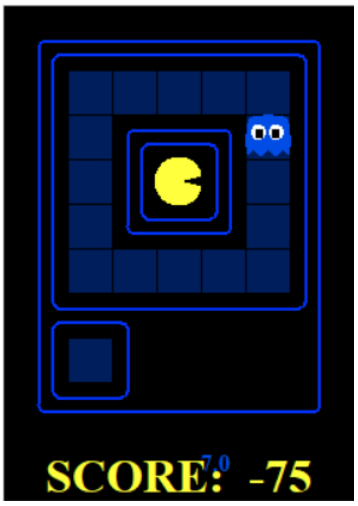

Figure 1 - Uniform distribution at the end of test-1

In the second case, GoSouthAgent is used, therefore, **the steady state belief is higher in the southern regions** of the grid.

Figure 2- Higher belief in southern regions

Question 2

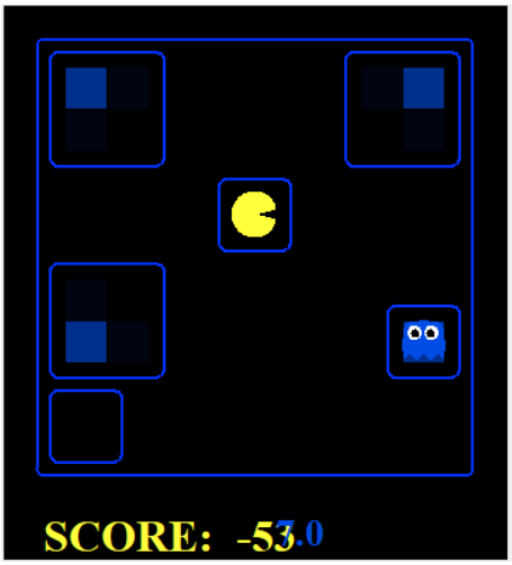

Figure 3 - Test case 2 - unable to find the ghost

In test case 2, Pacman does not move. Since it does not move, the observation it gets is constant. Therefore, the locations at the same distance from the Pacman's stationary position becomes equally likely.

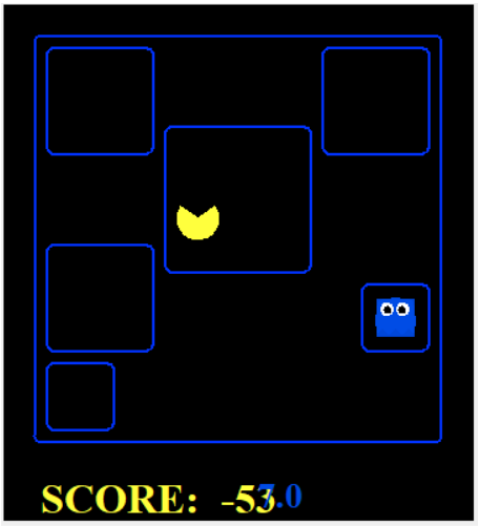

Figure 4 - Test case 3 - ghost is found

However, in test case 3, Pacman can move and can make different observation updates for different positions. Since the probability of observation given that the ghost is where it is gets larger when the Pacman gets closer, the distribution congregates in the true location.

Question 3

The particles are reinitialized when the probability of the observation given the particle's distance, or shortly its weight, is 0 for all particles.

The reason this happens is that the all particles eventually **congregates** to a state as a result of randomness in the resampling procedure. The particles fail to represent the exact distribution of ghost's location and at some point, the probability of an observation given a particle become zero for all of the particles as there are no particles left in the vicinity of the true state, so called **particle deprivation.**

Particle deprivation can be solved by increasing the number of particles as it will become harder for all particles to congregate to a state. However, my experiments in Pacman world showed that in 6th example where Pacman moves, more particles help with particle deprivation. However, in the 5th example where the Pacman is stationary, arbitrarily high particles count does not seem to have any effect on this problem.

Question 4

Particle filter is a very accurate sampling method for inference on HMM's. Therefore, in both exact inference and particle filter update and predict steps, the probabilities evolve almost in the same way. In some test cases like the second case, the exact inference seems to converge to final values in fewer steps. However, there are no substantial differences.

Using 5000 particle for this problem does **not** make sense for these problems. There are about 10-15 states in these problems. Belief updates for these few states are not that costly. However, using 5000 particle requires sampling, weight calculation, resampling for 5000 times. Therefore, it becomes more costly to do particle filtering with that many particles in a problem with few states.

Question 5

In Question 5, I used a **tuple of ghost positions** as particles.

In observation update part of the Dynamic Bayes' Net inference, the likelihood weight of a particle P that holds ghosts, G_{1^a} and G_{1^b} given the observations of these ghosts' locations, E_{1^a} and E_{1^b} is:

$$P(E_{1^a} \mid G_{1^a}) * P(E_{1^b} \mid G_{1^b})$$

In the code,

```
weightedParticles = []
for particle in self.particles:
    weight = 1
    for i in range(self.numGhosts):
        weight *= self.getObservationProb(observation[i], gameState.getPacmanPosition(), particle[i],
                                          self.getJailPosition(i))
    weightedParticles.append((weight, particle))
```

I loop over the ghosts and weight the entire particle with the weight given above. Then I do resampling as usual.

In the elapse time of the joint particle filter,

I, like the normal particle filter, add a new particle from sampling from the next position distribution found using the transition model $P(X_{t+1}|X)$. This time the only difference is that I sample the next position of each ghost in particle tuple separately and then gather them together to form the next particle tuple.

The code is follows:

```
newParticles = []
for oldParticle in self.particles:
    newParticle = list(oldParticle)  # A list of ghost positions

    # now loop through and update each entry in newParticle...
    "*** YOUR CODE HERE ***"
    for i in range(len(newParticle)):
        newParticle[i] = self.getPositionDistribution(gameState,newParticle,i,self.ghostAgents[i]).sample()
    """*** END YOUR CODE HERE ***"""
    newParticles.append(tuple(newParticle))
self.particles = newParticles
```