Assignment 4 – Part 2

Mertcan Aşgün 63948

1. Even though the ghost is moving, we have access to the given getPositionDistribution method, which gives the probability regarding the ghost’s next position. So, our predictions take this into account. In the first test case, ghost uses “SeededRandomGhostAgent”, so its movement is random. Therefore, the probabilities of each possible position that ghost can be located are the same. In the second test case however, ghost uses “GoSouthAgent”. So, it is easy to realize that ghost is probably located in southern parts of the given map. Same logic can be used in opposite direction as well, for example, if probabilities tend to gather in an area, probably that ghost uses an agent that tends to go in that direction.
2. The given observations are for the positions around the pacman, in a certain radius. By the implementation, we update our beliefs with probability values we get for each position. My thought is that, since the pacman does not move in the first one, it cannot update the weights, and so the probabilities above a certain radius. That is why most of the squares turn into black as evidences come, however the ones in the corners do not change, simply because they are out of the observation reach for the pacman. In the second code, we see that pacman is moving. By doing so, it can reach those corners that initially was out of the reach. Their calculated weights were low and by multiplying it continuously with initial belief value, we continuously decrease them and eventually reach very low belief values, which is represented by black squares.
3. In both test cases, initially a high number of squares start as blue since we initialized them uniformly distributed. As time passes, as evidences are gathered, some of those squares turns into black. When ghost change its position to a different cluster than before, or when the pacman itself checks that square itself and sees that there is no ghost, the previous beliefs turn to be wrong and they turn into black as well. Eventually the total of the beliefs become 0 and then it initializes uniformly again, and number of blue squares increases instantly. Increasing the number of particles could help up to a point but after that it slows the algorithm a lot. And also, due to high number of particles, at each step when we normalize the probabilities, we may still lose precision due to representation problems of float numbers.
4. As we might expected, exact inference results are more accurate, since we are calculating all possibilities, instead of relying on samples. On the other hand, approximate inference is faster to measure, from the same reason. If we were to increase the number of samples, we would benefit from it up to a point, but then we would lose the advantage of approximate inference, which is the speed of calculation. So instead of increasing sample sizes to 5000, it would be better to switch into exact inference, if we are able to.
5. I tried to answer those parts but looks like I had a problem and the autograder gives 0. My idea was nearly the same with the particle filtering methods I wrote for the approximate inference. Only this time, I used an additional for loop to include all ghosts, which was just 2 in our case. I got the jail positions for each ghost and put them in a list. For each ghost, and for each particle, I used the getObservationProb method to measure the weight and multiplied this weight value with initial belief value. Then I checked the total of those weighted beliefs to handle the special case. After normalizing the beliefs values, I sampled from it for each particle.

I did not have the chance to look for last part, Q10 in code since it is also connected to Q9. I just wanted to inform that.