Comp341 Introduction to Artificial Intelligence Assignment4 Report – Fall 2022

Name – Surname: Barış KAPLAN
KU ID Number: 0069054 (69054)
KU Login Name: bkaplan18

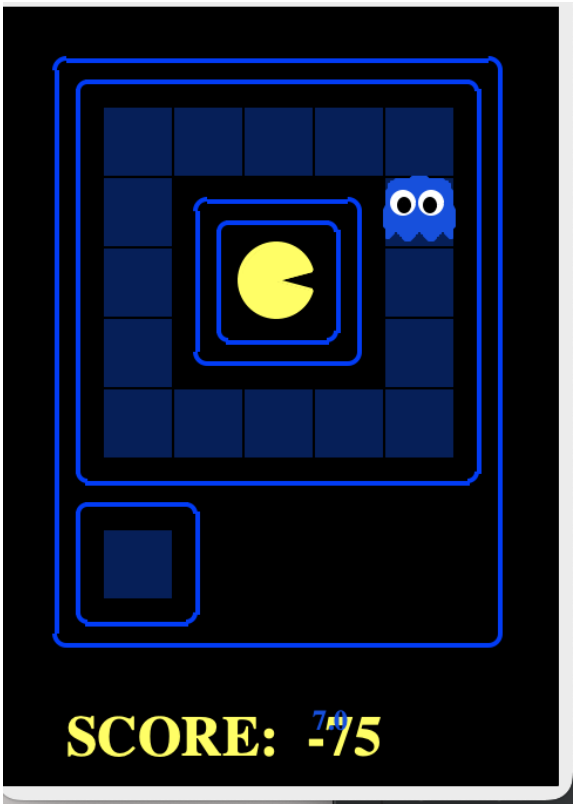
Q1:

```
class: "DoubleInferenceAgentTest"

seed: "188"

layout: """
%%%%%%%
%   G %
% %%% %
% %P% %
% %%% %
%     %
%%%%%%%
% %%%%%
%%%%%%%
"""

observe: "False"

elapse: "True"

checkUniform: "False"
maxMoves: "75"

numGhosts: "1"
ghost: "SeededRandomGhostAgent"
errorMsg: "Exact inference elapseTime test: %d inference errors."
inference: "ExactInference"
L2Tolerance: "0.0001"
```

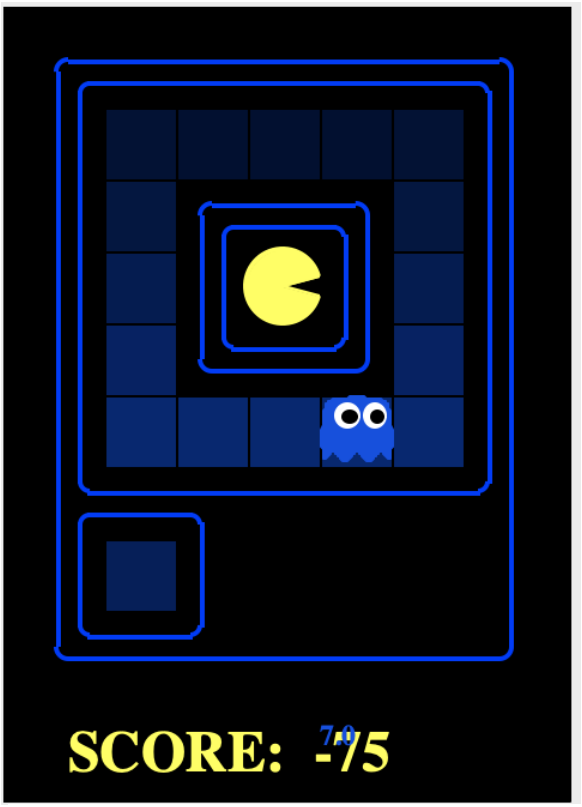
The ".test" file for the test case q3/1-ExactPredict (".test" file SS for q3/1-ExactPredict)
The ghost is "SeededRandomGhostAgent".

```
class: "DoubleInferenceAgentTest"

seed: "188"

layout: """
%%%%%%%
%  G  %
% %%% %
% %P% %
% %%% %
%     %
%%%%%%%
% %%%%%
%%%%%%%
"""

observe: "False"

elapse: "True"

checkUniform: "False"
maxMoves: "75"

numGhosts: "1"
ghost: "GoSouthAgent"
errorMsg: "Exact inference elapseTime test: %d inference errors."
inference: "ExactInference"
L2Tolerance: "0.0001"
```

The ".test" file for the test case q3/2-ExactPredict (".test" file SS for q3/2-ExactPredict).
The ghost is "GoSouthAgent".



The output of the test case named as "q3/1-ExactPredict" (AGENT1 SS)



The output of the test case named as "q3/2-ExactPredict" (AGENT2 SS)

Q1: As you can see in the above screenshots named **".test file SS for q3/1-ExactPredict"**, and **".test file SS for q3/2-ExactPredict"**; the observation event is always considered as **False** in both of the ".test" files of the first and second test cases of Q3. Because, in both of these ".test" files, the variable called **observe** is assigned to **False.** In other words, these test cases assume that the observation never happened. Moreover, these test cases do not change the state of observation to **True**. The test cases of Q3 aims to test and update the beliefs based on the movements of ghost and elapsed time of the process of exact inference.

There are two reasons why the values of the probabilities settle although the ghost is moving. **The first reason is the following:** The test cases of Q3 ignore observation and when there exists no observation, **Hidden Markov Model (HMM)** changes to a **regular/normal Markov Model**. Because, the hidden states cannot be inferred without observations. Furthermore, with no observation, the only available and well-defined information are the transition probabilities between the states and the long-term probability values of being in each of those states. Since it well defines the long-term probability values of states, the regular Markov Model will approach to a stationary distribution of the states. **The second reason is the following:** As I showed in the corresponding screenshots, there exists no observation for Q3. Therefore, the discrete distribution of the beliefs held by the Pacman agent does not alter **(or cannot change)** based on the observations of the Pacman. Due to these reasons; even though the ghost is moving, the values of probabilities settle in Q3 tests.

By observing the state distributions of the ghosts, I can tell the two ghosts apart. In the first test case of Q3**,** the random ghost called **"SeededRandomGhostAgent"** is used. Moreover, the stationary state distribution of this random ghost is uniformly random. Because regardless of where the ghost really is, the probability of the ghost being in the subsequent legal position is calculated **(This makes probability of being in each of the next legal positions uniform and random).** In the second test case of Q3, the agent called **"GoSouthAgent" is used.** This agent (**"GoSouthAgent")** is more likely to move to the southern places, which means that the beliefs of the stationary state distribution have higher values in the southern locations of the grid for this agent.

These ghosts can be aparted also by observing the colors of the squares of grids. Lighter blue colors of squares means the probability of being in those squares is higher. As you can see in the **"AGENT2 SS"**, the bottom squares have lighter colors of blue. This means that the second agent tends to move to the southern locations, the beliefs of southern positions are higher for the second agent, and thus the second agent is **"GoSouthAgent".** However, in the **"AGENT1 SS"**, the brightness of colors are almost same in each square. This means that stationary state distribution of the first agent is uniformly random, and thus the first agent is "**SeededRandomGhostAgent".**

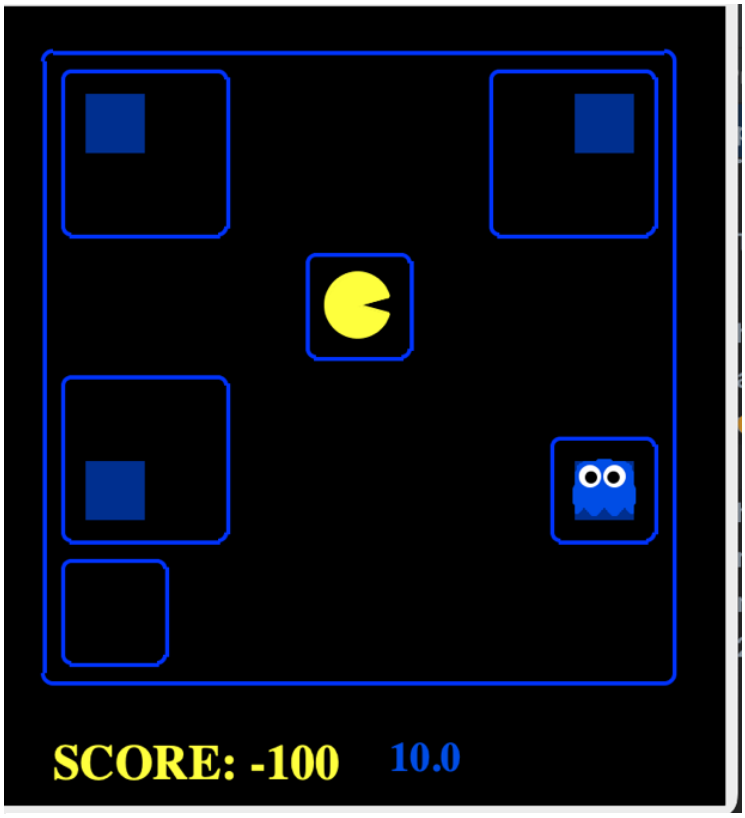
```
observe: "False"

elapse: "True"

checkUniform: "False"
maxMoves: "200"

numGhosts: "1"
ghost: "GoSouthAgent"
errorMsg: "Exact inference elapseTime test: %d inference errors."
inference: "ExactInference"
L2Tolerance: "0.0001"
```

3rd test case file of Q3

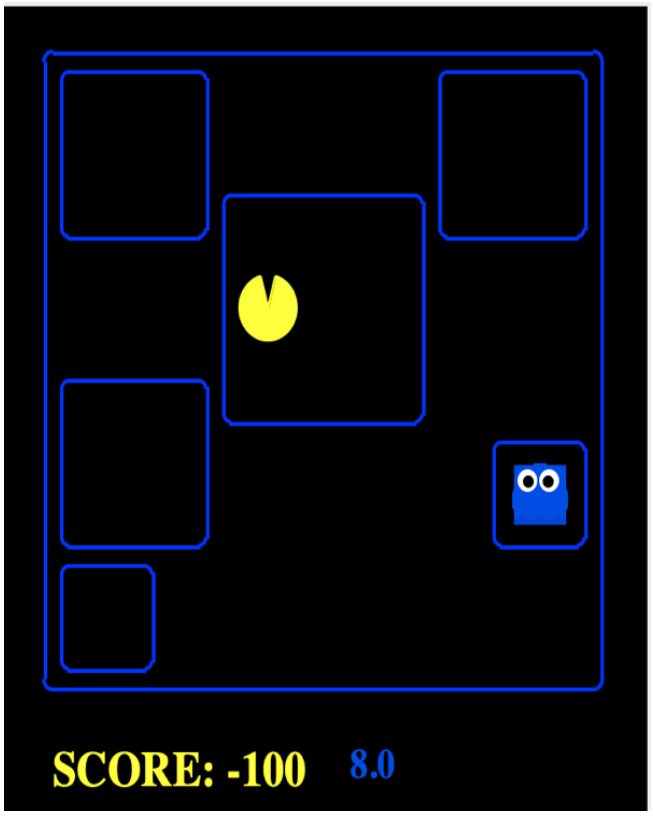
```
observe: "False"

elapse: "True"

checkUniform: "False"
maxMoves: "500"

numGhosts: "1"
ghost: "SeededRandomGhostAgent"
errorMsg: "Exact inference elapseTime test: %d inference errors."
inference: "ExactInference"
L2Tolerance: "0.0001"
```

4th test case file of Q3

Q2:



The output of the second test case of the second question (The output of python autograder.py -t test_cases/q2/2-ExactUpdate)



The output of the third test case of the second question (The output of python autograder.py -t test_cases/q2/3-ExactUpdate)

Q2 : As you can see in the corresponding above screenshot, the Pacman agent does not **(or cannot)** move in the second test case of the second question. Due to this reason, the position of the Pacman agent is always stationary and the observation which the agent has/obtains is always identical and constant in the 2nd test case of the 2nd question. Thus, in this scenario, the distance between the Pacman agent and each grid location that can be visited by the agent always stays constant/identical. This situation makes the probability of Pacman agent moving to any of the grid positions which are equidistant from its current stationary position identical. In this question, Pacman's belief is updated by multiplying the value of the current belief with the value of the probability of observation. Therefore, from this situation, we can infer that the only way to update the beliefs in this question is obtaining different and changing observations. However, since the observation and the position of Pacman are always constant in this test scenario, the value of the belief about possible ghost locations cannot be updated. Furthermore, the Pacman agent cannot obtain new observations about different corner locations. Due to these reasons, the ghost cannot be found in the second scenario.

On the other hand, in the third test case of the second question, the agent can move to the different grid locations. Therefore, it can reach different observations about the possible corner locations of ghosts and update its belief accordingly. By obtaining new observations and updating its belief, Pacman can better guess the location of ghost and better eliminate/ignore the locations where the ghost does not exist. Therefore, consequently, the discrete distribution of beliefs approaches to the actual position of the ghost. Thus, the ghost can be found.

Q3:

Q3: The particles are reinitialized when the weight of each of the particles is equal to 0 (when all the particles get weights equal to 0) depending on the current evidence we have. Here, what I meant by weight is the probability of Pacman agent observing the particle based on the distance from the agent location to the particle location. **The reason why the reinitialization of the particles happen is as follows:** As the time elapses, the particle observations of the Pacman agent are updated, and the particles gain weights. In the 5th and 6th test cases of the question 6, the time elapse is not utilized (**because it is set to False**). However, the **observe** metric is used in these test cases (**because it is set to True).** Therefore, in these test cases, the Pacman agent can have inferences only by using its observations, and it cannot use the **time elapse** as the metric of this inference process. As a result, the Pacman agent ignores the movements of the ghosts within the elapsed time of the 5th and 6th test cases of Q6. At some point of these test cases, the particles cannot accurately/exactly represent the discrete distribution the actual locations of the ghosts. Therefore, based on the particles, the Pacman agent controls/visits all feasible locations of the ghost. This means that the Pacman agent could not infer the ghost location, and the probability of Pacman agent observing the ghost based on the particles becomes 0. Hence, the particles are reinitialized, and the weights of these particles become 0.

Moreover, the involvement of the **stochasticity and randomness** in the **process of particle resampling** is another reason why the particles are reinitialized at some point. The randomness in the particle resampling process can lead to focusing on a small region of the whole state space and discarding high-weighted particles which represent the actual location of the ghost and actual state of the game better. Moreover, due to this randomness, the particle resampling may select and resample the low-weighted particles more. More focus on low-weighted particles and less focus on high weighted particles can bring a less accurate representation of the actual state of the game and actual location of ghost. As a result, the agent could not infer the ghost location and it checks all possible ghost locations. Therefore, the particle weights become 0 and the particles are reinitialized.

Increasing the number of particles is not a guaranteed solution for the particle reinitialization. However, it may increase the total number of high-weighted particles, thus provide a more accurate representation of the actual ghost location and actual game state. This can prevent the **"particle depletion"**, and thus the reinitialization of the particles. However, the time elapse metric is not used in this question. Therefore, the particles are not changing over time in these test cases. Consequently, even though the number of particles increases, particle reinitialization is less likely to occur in this context. Therefore, increasing the number of particles may not be a solution of the particle reinitialization issue. In the fifth test case, as the position of agent is stationary, I did not observe any effect of increasing the number of particles in solving the particle reinitialization issues of Q6.

Q4:

Q4: The consequences of the exact inference are more accurate than the consequences of the approximate inference. Moreover, the probabilities computed in the approximate inference method involves higher level of uncertainty than the exact inference. Because, in the case of exact inference, the actual data is used. The probabilities are exactly calculated in the exact inference case. On the other hand, in the case of approximate inference, random samples are generated, and the probabilities are calculated based on those samples. Moreover, the approximate inference case is faster than the exact inference case. Because it needs less calculation steps. Moreover, its time complexity (or computational complexity) is less. In Q2, Q3 vs Q5, Q6 (exact vs approximate), I observed that the probabilities are evolved similarly. The outcomes of approximate inference cases approached/converged the consequences of the exact inference cases. I also observed that the results of approximate inference cases involve low uncertainties. Overall, there were no significant differences between the results of the approximate inference cases and the exact inference cases. If the time elapse metric is involved, then 5000 particles make sense for the approximate inference problems. Because, without "time elapse", the agent may become more focused on certain areas of the state space, eliminate the high-weighted particles, and focus more on low-weighted ones in those specific areas. This will bring lower accuracy. However, with "time elapse", as the number of high-weighted particles highly likely to increase, more accuracy and better results can be obtained. The high-weighted particles represent the true location of ghost and actual game state more accurately compared to the low-weighted ones. Moreover, as I mentioned, the approximate inference has less time complexity than the exact inference. Therefore, by increasing the particle number to 5000, we can get more accurate results and less time complexity simultaneously for the approximate inference. The time complexity is higher in the exact inference problems compared to approximate inference problems. Having 5000 particles (which is too high) will significantly increase the time complexity for the exact inference problems. This is also supported by the situation that the number of states (almost 12) of the exact inference problems is much less than 5000. Therefore, having 5000 particles does not make sense for the exact inference problems. You can see information supporting this below.

Trade-offs in Exact Inference and Approximate Inference in General:

- Exact Inference: More accuracy and More Time Complexity
- Approximate Inference: Less accuracy and Less Time Complexity

Q5:

In the new particle creation with the elapsed time (for the Joint Particle Filter), I have firstly defined the length of the new particle list which stores the ghost positions as a list. For this purpose, I used len() function. Then, I have iterated through the entries of the list called "newParticle". Then, I have obtained each ghost by using self.ghostAgents. Here, to self.ghostAgents, I passed the index of the particle as parameter. Then, I called getPositionDistributions() function to obtain the next ghost position distribution and then update each particle within the list of ghost positions. After that, I have obtained samples from the updated particles. For that purpose, I have used sample() function. Finally, I have assigned the sample obtained from the updated particle to the old ghost particle. While obtaining samples from the subsequent position distribution, I have followed a transition model whose mathematical representation is as follows:

Transition Model: P(Y(t+1)|P(Y(t))). Here, **t+1** represents next state, and **t** represents current state.

In the weight calculation, I have firstly created a **DiscreteDistributon** data structure by using the DiscreteDistribution() call. Then, I have iterated over all particles by using a for loop. Here, the particles (namely self.particles) contain all **the lists of the ghost positions**. Then, I have initialized the value of observation probability to 1. After that, I have iterated over the ghosts via using a for loop. In this for loop, by using the **getObservationProb()** method, I have obtained the ghost observation probability for each ghost. Next, I have updated the current weight by multiplying it with the newly found observation probability. Subsequently, I have added the updated weight to each particle. Then, if the summation of all values within the DiscreteDistribution() is 0, then I initialized the game state uniformly. Otherwise, I created a list of samples from all particles and assigned the current list of particles to this newly created list (i.e., **resampling process**).

Conditional probability based mathematical formula I used:

Let W1(i) and W1(j) represent the current particle weights. Let Ex1(i) and Ex1(j) represent the new observations of the positions of ghosts.

Then, the formula is as follows:
P(Ex1(i) | W1(i)) * P(Ex1(j) | W1(j))

> **IMPORTANT NOTE:** You can find my corresponding implementations in the below screenshots.

```
all_possible_particles = self.particles  # obtaining all particles
total_number_of_particles = self.numParticles  # obtaining the total number of the particles
total_number_of_ghosts = self.numGhosts  # obtaining the total number of the ghosts
agent_pacman_location = gameState.getPacmanPosition()  # obtaining the pacman position

custom_range_of_ghost_number = range(0, total_number_of_ghosts)

instance_of_discrete_distribution = DiscreteDistribution()
for particle_index in all_possible_particles:
    first_val_of_evidence_probability = 1
    for ghost_ind in custom_range_of_ghost_number:
        single_instance_of_ghost_observation = observation[ghost_ind]
        single_instance_of_particle = particle_index[ghost_ind]
        location_of_jail = self.getJailPosition(ghost_ind)
        probability_of_observation = self.getObservationProb(
            single_instance_of_ghost_observation,
            agent_pacman_location,
            single_instance_of_particle,
            location_of_jail
        )  # obtaining observation probability
        first_val_of_evidence_probability = first_val_of_evidence_probability * probability_of_observation
    instance_of_discrete_distribution[particle_index] = instance_of_discrete_distribution[
                                                        particle_index] + first_val_of_evidence_probability
```

WEIGHT CALCULATION CODE SS1

```
keys_of_discrete_distribution = instance_of_discrete_distribution.values()
summation_of_distribution = sum(keys_of_discrete_distribution)
summation_of_distribution = float(summation_of_distribution)
check_summation_non_zero = (summation_of_distribution > 0 or summation_of_distribution < 0)
check_summation_zero = (summation_of_distribution == 0)
if check_summation_non_zero:
    lst_of_all_smp = list()
    my_particle_number_range = range(0, total_number_of_particles)
    for particle_index_val in my_particle_number_range:
        lst_of_all_smp.append(instance_of_discrete_distribution.sample())
    self.particles = list(lst_of_all_smp)
elif check_summation_zero:
    self.initializeUniformly(gameState)
```

WEIGHT CALCULATION CODE SS2

```
length_of_updated_particle = len(newParticle)
my_custom_particle_range = range(0, length_of_updated_particle)
for particle_index in my_custom_particle_range:
    agent_ghost_adversary = self.ghostAgents[
        particle_index
    ]
    position_distribution_instance = self.getPositionDistribution(
        gameState,
        newParticle,
        particle_index,
        agent_ghost_adversary
    )
    sampled_pos_from_position_distribution = position_distribution_instance.sample()
    newParticle[particle_index] = sampled_pos_from_position_distribution
```

CODE OF THE PARTICLE CREATION WITH THE ELAPSED TIME (SS1)