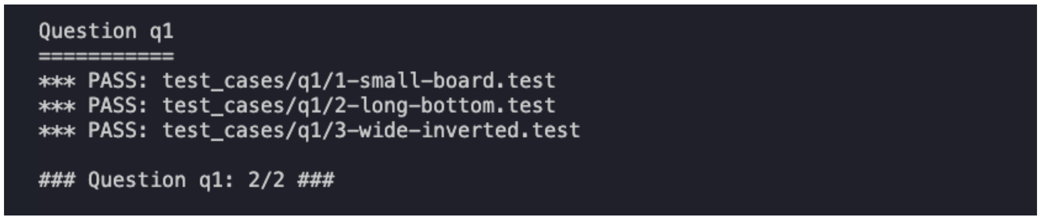
AI HW4

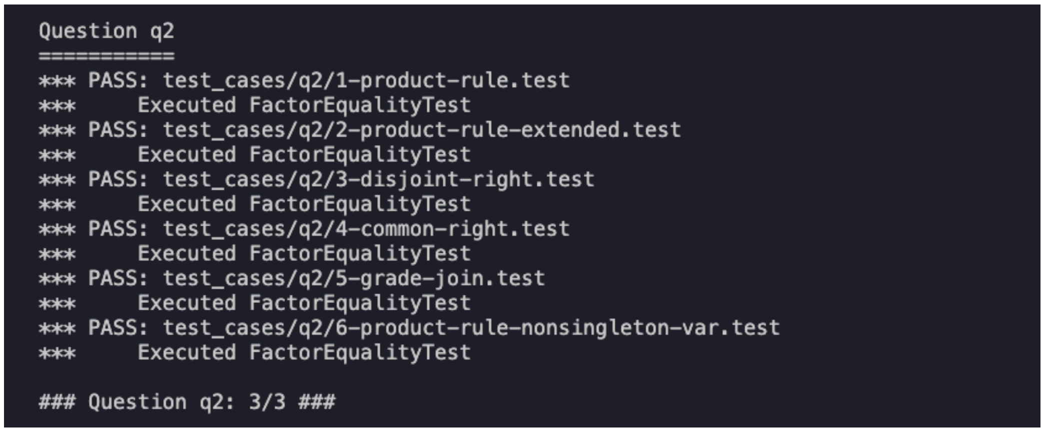
R12922029 陳姵安

1. **Show your autograder results and describe the implementation details.**
    a. **autograder results**
        - **Q1 – Bayes Net Structure**



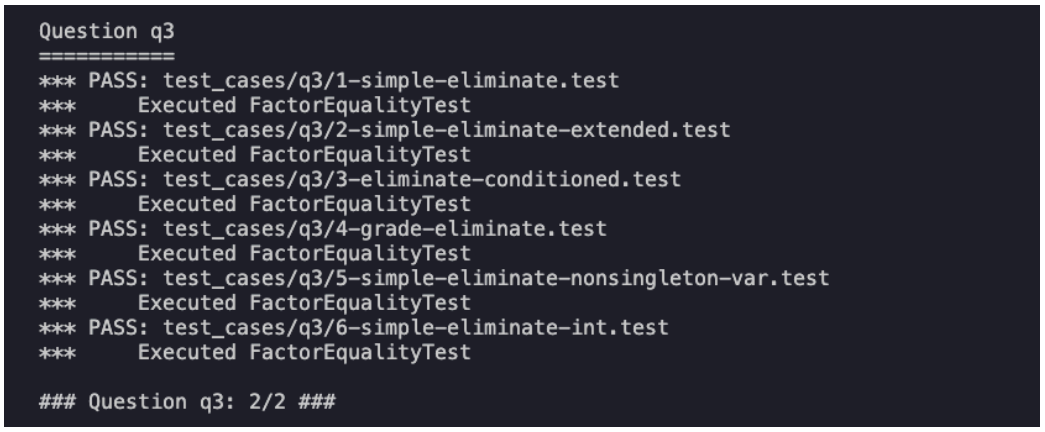
```
Question q1
===========
*** PASS: test_cases/q1/1-small-board.test
*** PASS: test_cases/q1/2-long-bottom.test
*** PASS: test_cases/q1/3-wide-inverted.test

### Question q1: 2/2 ###
```

        - **Q2 – Join Factors**



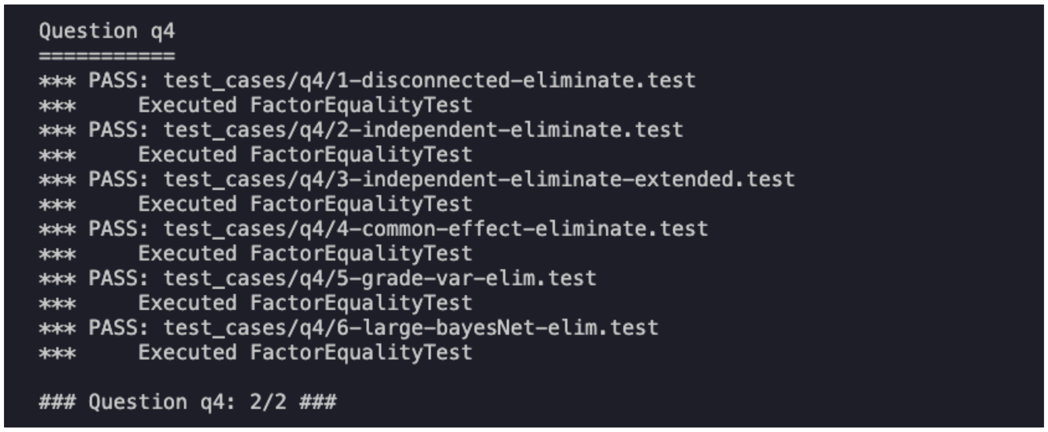
```
Question q2
===========
*** PASS: test_cases/q2/1-product-rule.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q2/2-product-rule-extended.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q2/3-disjoint-right.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q2/4-common-right.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q2/5-grade-join.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q2/6-product-rule-nonsingleton-var.test
***     Executed FactorEqualityTest

### Question q2: 3/3 ###
```

        - **Q3 – Eliminate (not ghosts yet) 講義 p22-25**



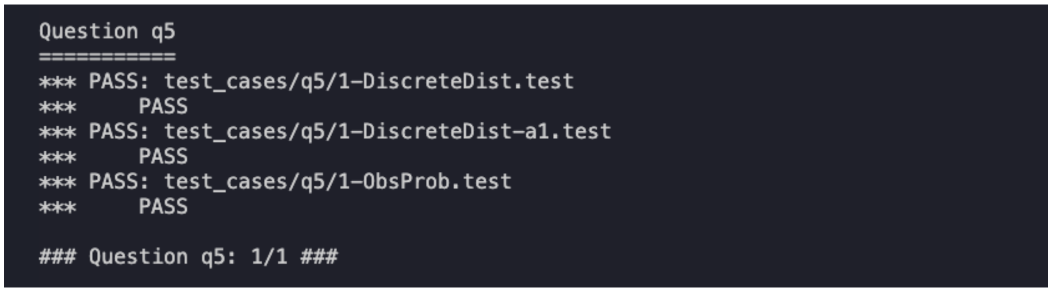
```
Question q3
===========
*** PASS: test_cases/q3/1-simple-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q3/2-simple-eliminate-extended.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q3/3-eliminate-conditioned.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q3/4-grade-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q3/5-simple-eliminate-nonsingleton-var.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q3/6-simple-eliminate-int.test
***     Executed FactorEqualityTest

### Question q3: 2/2 ###
```

- **Q4 – Variable Elimination**



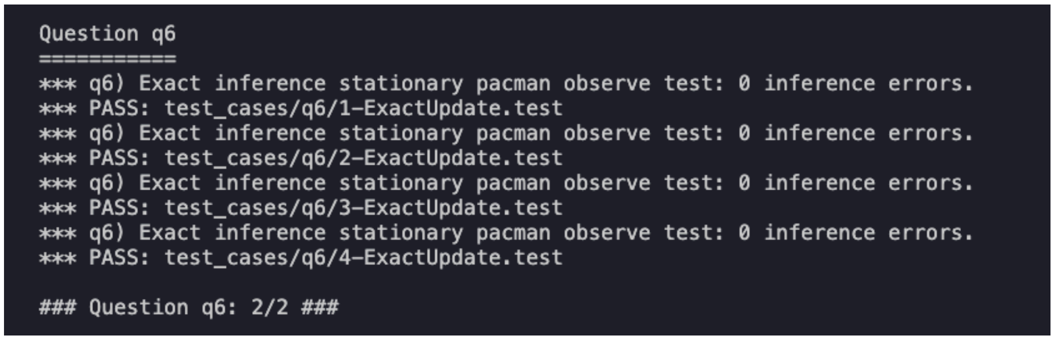
```
Question q4
===========
*** PASS: test_cases/q4/1-disconnected-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q4/2-independent-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q4/3-independent-eliminate-extended.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q4/4-common-effect-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q4/5-grade-var-elim.test
***     Executed FactorEqualityTest
*** PASS: test_cases/q4/6-large-bayesNet-elim.test
***     Executed FactorEqualityTest

### Question q4: 2/2 ###
```

- **Q5a – DiscreteDistribution Class & Q5b – Observation Probability**



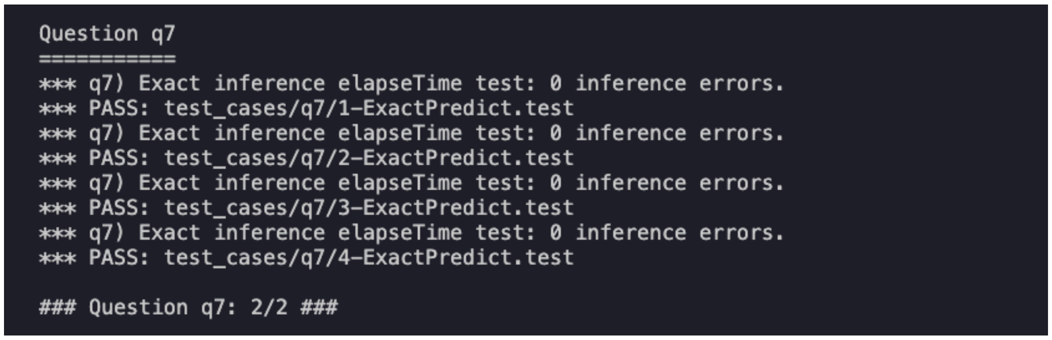
```
Question q5
===========
*** PASS: test_cases/q5/1-DiscreteDist.test
***     PASS
*** PASS: test_cases/q5/1-DiscreteDist-a1.test
***     PASS
*** PASS: test_cases/q5/1-ObsProb.test
***     PASS

### Question q5: 1/1 ###
```

- **Q6 – Exact Inference Observation**



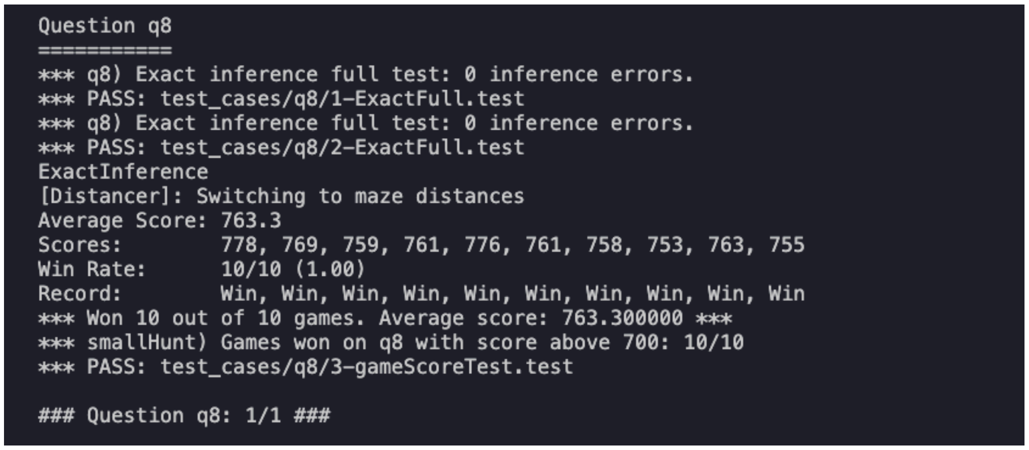
```
Question q6
===========
*** q6) Exact inference stationary pacman observe test: 0 inference errors.
*** PASS: test_cases/q6/1-ExactUpdate.test
*** q6) Exact inference stationary pacman observe test: 0 inference errors.
*** PASS: test_cases/q6/2-ExactUpdate.test
*** q6) Exact inference stationary pacman observe test: 0 inference errors.
*** PASS: test_cases/q6/3-ExactUpdate.test
*** q6) Exact inference stationary pacman observe test: 0 inference errors.
*** PASS: test_cases/q6/4-ExactUpdate.test

### Question q6: 2/2 ###
```

- **Q7 – Exact Inference with Time Elapse**



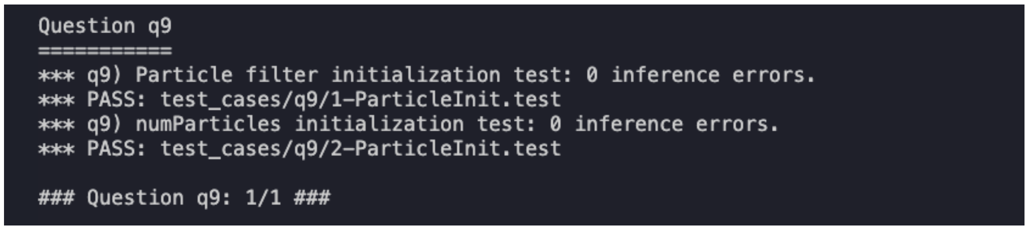
```
Question q7
===========
*** q7) Exact inference elapseTime test: 0 inference errors.
*** PASS: test_cases/q7/1-ExactPredict.test
*** q7) Exact inference elapseTime test: 0 inference errors.
*** PASS: test_cases/q7/2-ExactPredict.test
*** q7) Exact inference elapseTime test: 0 inference errors.
*** PASS: test_cases/q7/3-ExactPredict.test
*** q7) Exact inference elapseTime test: 0 inference errors.
*** PASS: test_cases/q7/4-ExactPredict.test

### Question q7: 2/2 ###
```

- **Q8 – Exact Inference Full Test**



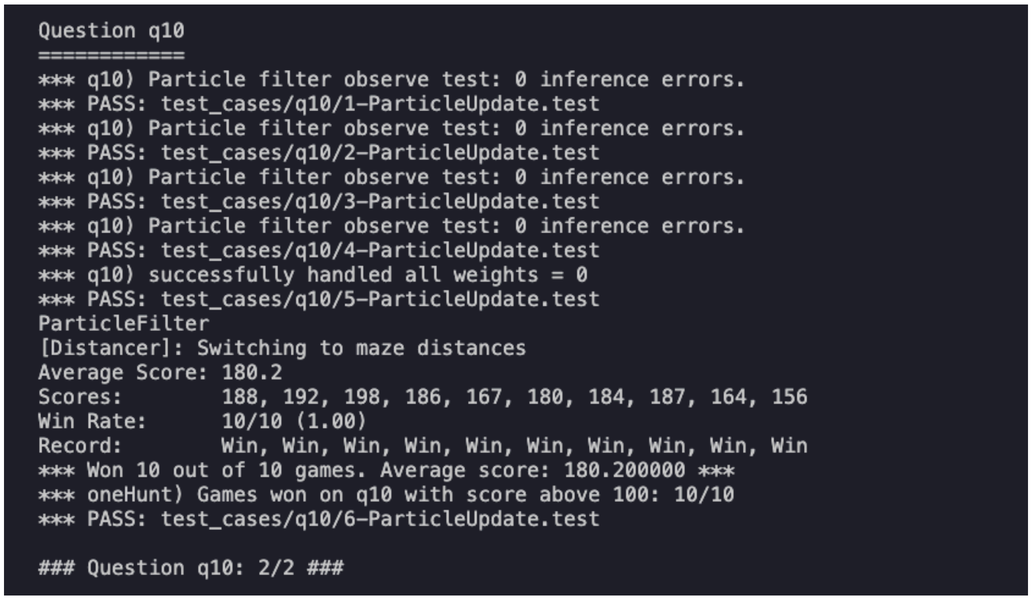
```
Question q8
===========
*** q8) Exact inference full test: 0 inference errors.
*** PASS: test_cases/q8/1-ExactFull.test
*** q8) Exact inference full test: 0 inference errors.
*** PASS: test_cases/q8/2-ExactFull.test
ExactInference
[Distancer]: Switching to maze distances
Average Score: 763.3
Scores:        778, 769, 759, 761, 776, 761, 758, 753, 763, 755
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** Won 10 out of 10 games. Average score: 763.300000 ***
*** smallHunt) Games won on q8 with score above 700: 10/10
*** PASS: test_cases/q8/3-gameScoreTest.test

### Question q8: 1/1 ###
```

- **Q9 – Approximate Inference Initialization and Beliefs**



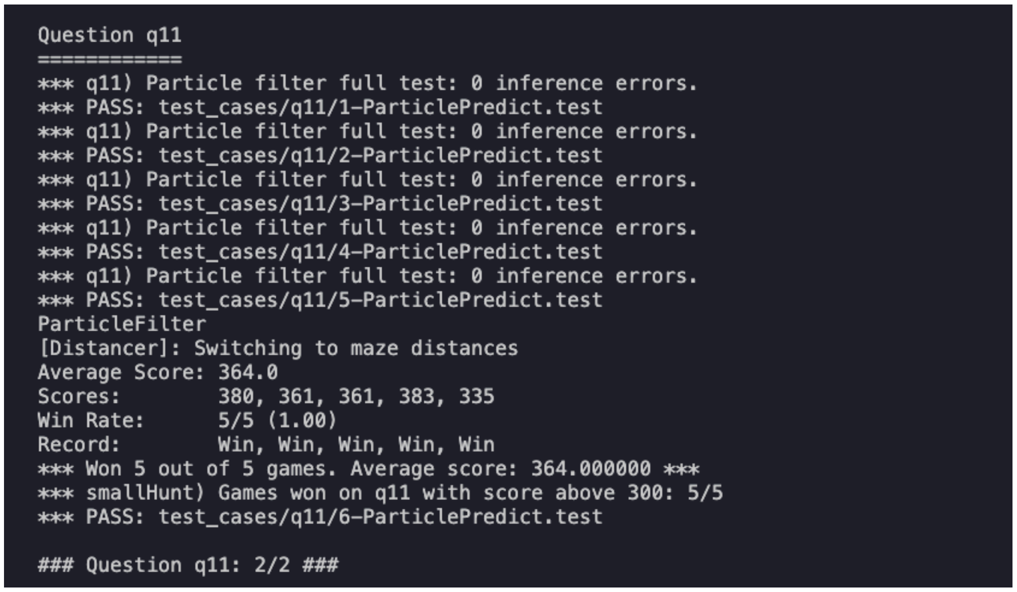
```
Question q9
===========
*** q9) Particle filter initialization test: 0 inference errors.
*** PASS: test_cases/q9/1-ParticleInit.test
*** q9) numParticles initialization test: 0 inference errors.
*** PASS: test_cases/q9/2-ParticleInit.test

### Question q9: 1/1 ###
```

- **Q10 – Approximate Inference Observation**



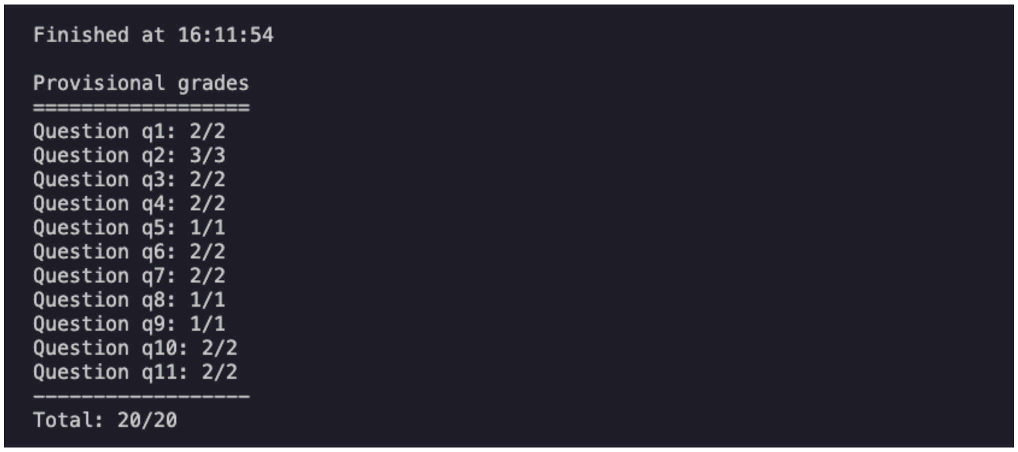
```
Question q10
============
*** q10) Particle filter observe test: 0 inference errors.
*** PASS: test_cases/q10/1-ParticleUpdate.test
*** q10) Particle filter observe test: 0 inference errors.
*** PASS: test_cases/q10/2-ParticleUpdate.test
*** q10) Particle filter observe test: 0 inference errors.
*** PASS: test_cases/q10/3-ParticleUpdate.test
*** q10) Particle filter observe test: 0 inference errors.
*** PASS: test_cases/q10/4-ParticleUpdate.test
*** q10) successfully handled all weights = 0
*** PASS: test_cases/q10/5-ParticleUpdate.test
ParticleFilter
[Distancer]: Switching to maze distances
Average Score: 180.2
Scores:        188, 192, 198, 186, 167, 180, 184, 187, 164, 156
Win Rate:      10/10 (1.00)
Record:        Win, Win, Win, Win, Win, Win, Win, Win, Win, Win
*** Won 10 out of 10 games. Average score: 180.200000 ***
*** oneHunt) Games won on q10 with score above 100: 10/10
*** PASS: test_cases/q10/6-ParticleUpdate.test

### Question q10: 2/2 ###
```

- **Q11 – Approximate Inference with Time Elapse**



```
Question q11
============
*** q11) Particle filter full test: 0 inference errors.
*** PASS: test_cases/q11/1-ParticlePredict.test
*** q11) Particle filter full test: 0 inference errors.
*** PASS: test_cases/q11/2-ParticlePredict.test
*** q11) Particle filter full test: 0 inference errors.
*** PASS: test_cases/q11/3-ParticlePredict.test
*** q11) Particle filter full test: 0 inference errors.
*** PASS: test_cases/q11/4-ParticlePredict.test
*** q11) Particle filter full test: 0 inference errors.
*** PASS: test_cases/q11/5-ParticlePredict.test
ParticleFilter
[Distancer]: Switching to maze distances
Average Score: 364.0
Scores:        380, 361, 361, 383, 335
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
*** Won 5 out of 5 games. Average score: 364.000000 ***
*** smallHunt) Games won on q11 with score above 300: 5/5
*** PASS: test_cases/q11/6-ParticlePredict.test

### Question q11: 2/2 ###
```

- **Total**



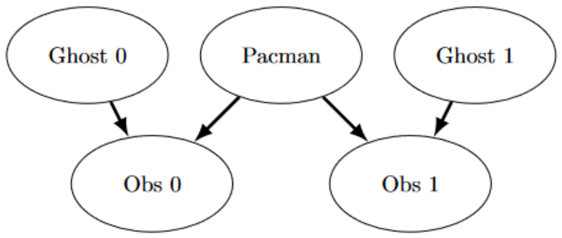
```
Finished at 16:11:54

Provisional grades
==================
Question q1: 2/2
Question q2: 3/3
Question q3: 2/2
Question q4: 2/2
Question q5: 1/1
Question q6: 2/2
Question q7: 2/2
Question q8: 1/1
Question q9: 1/1
Question q10: 2/2
Question q11: 2/2
------------------
Total: 20/20
```

2. Describe the implementation details

a. Q1 – Bayes Net Structure

The syntax of a Bayes' Net consists of a directed acyclic graph (topology) with local probability information attached to each node. The semantics define how the syntax corresponds to a joint distribution over the variables of the network. This function is designed to help us practice constructing an empty Bayes Net structure as depicted in Figure 1, which includes the components listed below.

- **Variables :** pacman, ghost0, ghost1, observation0, observation1
- **Edges :** [(pacman, observation0), (ghost0, observation0), (pacman, observation1), (GHOST1, observation1)]
- **Variable Domains :** all the possible positions of each variable



◂ Figure 1 (source : https://inst.eecs.berkeley.edu/~cs188/sp22/project4/)

b. Q2 – Join Factors

The ***joinFactors*** function in Python is designed to combine multiple probabilistic factors into a single new factor. This process is crucial in Bayesian networks, particularly when performing probabilistic inferences that involve calculating joint distributions from multiple conditional probabilities. The resulting factor represents the product of the input factors, effectively aggregating the conditional dependencies captured by the input factors into a unified framework.

First, identify the unconditioned and conditioned variables, and obtain a variableDomainsDict that maps variables to their domains for the join of the factors. Use the ***difference_update*** method to modify the set of conditioned variables by removing all elements found in ***unconditioned_vars***. Then, create a new factor that includes these variables and whose probability entries are the products of the corresponding rows of the input factors.

c. Q3 – Eliminate (not ghosts yet)

The primary purpose of the eliminate function is to marginalize over a specified variable in a given probabilistic factor, effectively reducing the factor's complexity by summing out the specified variable. This operation allows for the simplification of the factor by eliminating unnecessary dimensions, which can be essential for efficient computation in larger graphical models.

First, identify the unconditioned and conditioned variables, and obtain a ***variableDomainsDict*** from the factor. Next, create a new unconditional variable set that does not contain the elimination variable. Generate a new factor table with the elimination variable removed. Then, enumerate all the assignments in the new factor and sum the probabilities of the assignments that can be combined.

d. Q4 – Variable Elimination

This function is designed to answer probabilistic queries using a Bayesian Network. It uses variable elimination to compute the conditional probability ***P(queryVariables|evidenceDict)***. Different from the aforementioned ***eliminate*** function, it operates on a higher level, coordinating multiple steps in the inference process, including the application of both joining and eliminating factors according to a specified order.
Its main goal is to simplify the network to focus only on the query variables and evidence provided, removing irrelevant variables systematically.

First, use the bayesNet.getAllCPTsWithEvidence(evidenceDict) function to get all factors that contain the evidence variables. Next, iterate over the eliminationOrder and join all factors that contain. It's important to eliminate the variable from the factor if the factor has more than one unconditioned variable because the result of the elimination would be 1 (after marginalizing all of the unconditioned variables), but it is not a valid factor. Last, join all factors together and normalize the result.

e. Q5a – DiscreteDistribution Class

The main purpose of this class is to provide a structured way to manage and manipulate probability distributions over discrete sets of elements, which represent potential states or observations in probabilistic models.

The ***normalize(self)*** function calculates the total sum of the values in the distribution. If the total is not zero, each value is divided by this total to normalize the distribution. If the distribution is empty or the total is zero, no action is taken to avoid division by zero errors and maintain the distribution unchanged.
The ***sample(self)*** function generates a random number between 0 and the total sum of the distribution's values. It then iterates through the elements, accumulating their weights until the accumulated sum exceeds the random number. The element at which this occurs is returned as the sampled element.

f. Q5b – Observation Probability

The primary aim of this method is to calculate the probability ***P(noisyDistance|pacmanPosition, ghostPosition)***, which represents the likelihood of Pacman's sensor reading (noisy distance) given the actual positions of Pacman and the ghost. This probability helps in determining how likely a particular observation of distance is, given the true circumstances in the game.

Firstly, check if the ghost is in jail, if so, return 1.0 if the ***noisyDistance*** is None, else return 0.0. Also, if the ***noisyDistance*** is None and the ghost is not in jail, return 0.0 because in this case, the ghost cannot be at the ***noisyDistance*** from Pacman. Calculate the true distance between Pacman's location and the ghost's location using the ***manhattanDistance*** function. Finally, using the ***busters.getObservationProbability*** method, return the likelihood of the noisy distance given the true distance.

g. Q6 – Exact Inference Observation

The primary purpose of the ***observeUpdate*** method is to refine the agent's beliefs about where ghosts might be located based on observed distances. It implements an online belief update mechanism that adjusts the probabilities associated with each potential ghost position whenever new evidence (an observation of distance) is processed.

First get the initial belief of the pacman, as well as the pacman position and jail position. Enumerate all the possible ghost positions and update the beliefs based on the observation by multiplying the belief of the ghost position by the probability of the observation given the pacman position and the ghost position.

h. Q7 – Exact Inference with Time Elapse

The primary aim of the ***elapseTime*** method is to simulate the passage of time and its effect on the estimated positions of ghosts based on their potential movements. It adjusts the belief distribution to reflect the probable new positions of ghosts, considering their previous positions and movement constraints.

For implementation, initialize ***newPosDist*** as a ***DiscreteDistribution*** object first. Enumerate over all old positions. Then, obtain the distribution over new positions given the old position and update the new beliefs based on the distribution from the old position. Finally, update ***self.beliefs*** to the normalized ***newBeliefs***.

i. Q8 – Exact Inference Full Test

The primary goal of the ***chooseAction*** method is to guide Pacman to move strategically towards the nearest ghost based on the aforementioned updated belief distributions. This approach utilizes both Pacman's observations and knowledge of ghost movements to determine the optimal path for capturing ghosts, thereby increasing the game's efficiency and success rate.

As for the implementation, enumerate through all the beliefs of the ghost positions to find the most likely position of each ghost (the position with the highest probability). Among all the most likely ghost positions, find the one that is closest to Pacman. Then, Find the action that brings Pacman closest to the closest ghost.

j. Q9 – Approximate Inference Initialization and Beliefs

The primary purpose of the ***ParticleFilter*** is to efficiently track the location of a ghost using a set of particles, where each particle represents a possible ghost location. This class provides methods to initialize these particles uniformly across all legal positions and to convert the distribution of particles into a belief distribution that reflects the probability of the ghost being at each location.

The ***nitializeUniformly*** function aims to initialize an evenly distributed list of particles. Therefore, in this function, it first gets the legal positions if there are any. If not, do nothing and return. Then, initialize the particles by evenly distributing them across the legal positions.
The ***getBeliefDistribution*** function aims to get the distribution of the list of particles. Firstly, initialize the belief distribution as a ***DiscreteDistribution*** object and further count the number of particles at each position. Then, Normalize the belief distribution to convert the counts to probabilities.

k. Q10 – Approximate Inference Observation

The ***observeUpdate*** method here is similar to the observeUpdate method at ***Q6.*** Both versions of the observeUpdate method aim to update the belief distribution about the ghost's position based on new observations provided by Pacman's sensors. The primary function is to adjust the internal state of the system to reflect new information, typically a noisy measurement of the distance from Pacman to the ghost.

Unlike using probabilities associated with each potential ghost position, the ***observeUpdate*** method of the approximate inference here adjusts the belief distribution based on the latest observation by weighting each particle—each representing a possible ghost position—according to the likelihood of the observed distance from Pacman's position. Firstly, calculate weights for each particle based on the observation and check if all weights are zero. If so, use the ***initializeUniformly*** function to reinitialize the particles. If not, resample from this weighted distribution to create a new set of particles, reflecting the updated belief about the ghost's location.

l. Q11 – Approximate Inference with Time Elapse

The primary aim of the ***elapseTime*** method of the ***ParticleFilter*** class is to update each particle, which represents a hypothesis about the ghost's location, according to probable ghost movements determined by the game state. This adjustment simulates the passage of time and the ghost's movement, maintaining the particle filter's effectiveness in tracking the ghost's trajectory.

Like the ***elapseTime*** method of the ***ExactInference*** class, it uses ***newPosDist = self.getPositionDistribution(gameState, oldPos)*** function to obtain the distribution over new positions given the old position. Then, sample a new position by randomly selecting from the distribution of new positions and update the self.particles.

Although it is not as precise as the elapseTime method of the ExactInference class, this method saves a lot of computational resources. Also, by implementing this particle filtering method, it is expected to be able to track ghosts nearly as effectively as with exact inference.