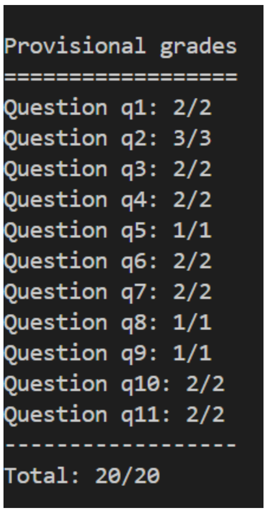
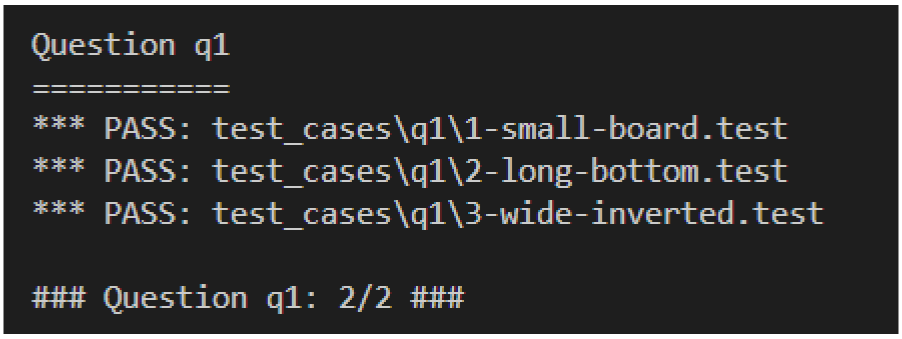

```
Provisional grades
==================
Question q1: 2/2
Question q2: 3/3
Question q3: 2/2
Question q4: 2/2
Question q5: 1/1
Question q6: 2/2
Question q7: 2/2
Question q8: 1/1
Question q9: 1/1
Question q10: 2/2
Question q11: 2/2
------------------
Total: 20/20
```

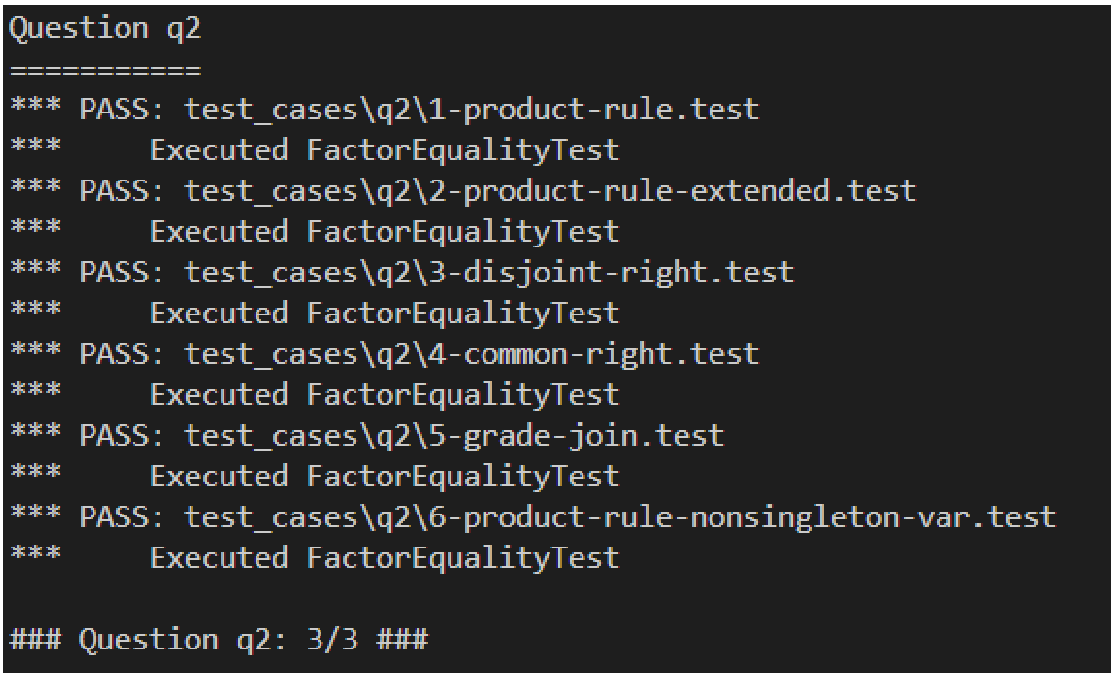
Q1


```
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 ###
```

- Identified the “variables” in the Bayesian network as Pacman, two ghosts, and their corresponding observations using predefined identifiers.
- Establish edges in the network to represent conditional dependencies, where each observation node is connected to both Pacman and one ghost. This structure indicates that the observations are influenced by the positions of both Pacman and the respective ghost.
- Computed all possible positions within the game grid to cover every potential location of Pacman and the ghosts. Assigned these positions as domain values for the respective variables, ensuring comprehensive coverage of all navigable space.
- Determined the range of potential Manhattan distances, adjusted for maximum noise, to set the domain values for the observation variables.
- Set up the initial structure of the Bayesian network with the defined variables and established edges, preparing for the addition of conditional probability tables.

Q2



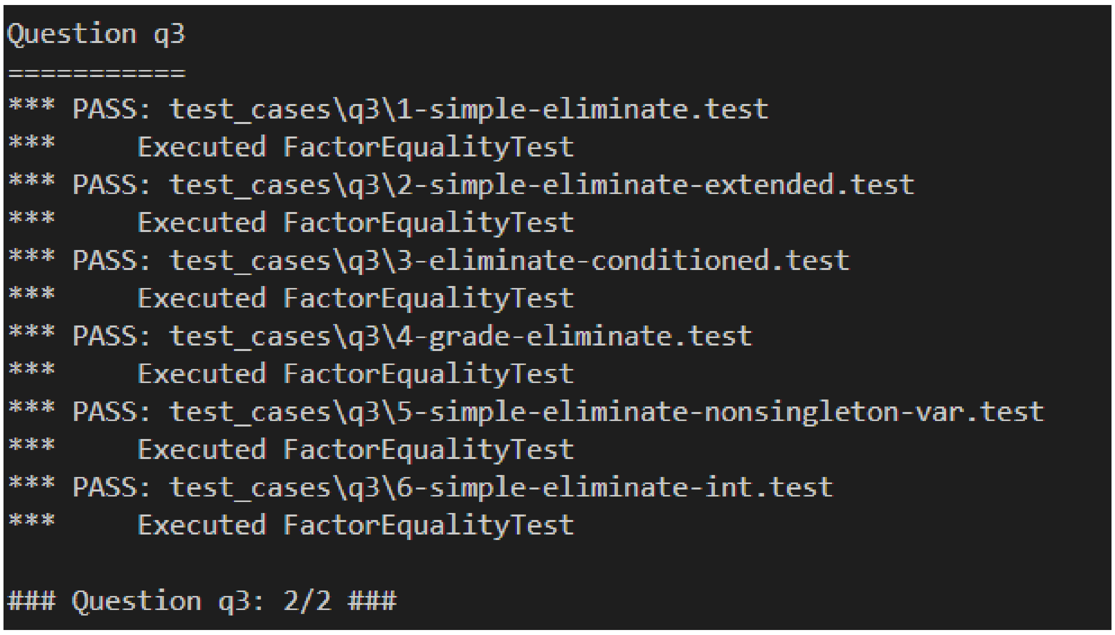
```
Question q2
===========
*** PASS: test_cases\q2\1-product-rule.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q2\2-product-rule-extended.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q2\3-disjoint-right.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q2\4-common-right.test
***     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 ###
```

- Recursively combined pairs of factors using the merge_two_factors function. This process continued until all input factors were merged into a single comprehensive factor.
- The set of **unconditioned variables** for each new factor was determined by taking the union of the unconditioned variables from the two merging factors. This union ensures that the new factor accounts for all possible outcomes represented by the original factors.
- **Conditioned variables** for the new factor were calculated by first taking the union of the conditioned variables from both factors, then subtracting any variables that also appeared as unconditioned. This step prevents duplication and maintains correct variable dependencies.
- Each new factor inherited the variable domain dictionary from one of the original factors, assuming consistency across all factors. Verification of domain compatibility was conducted when factors originated from different model segments.
- For each possible assignment within the combined variables, the probability associated with the new factor was computed by multiplying the corresponding probabilities from the original factors. This multiplication is for preserving the probabilistic relationships defined by the original models, ensuring that the joint probabilities reflect combined evidence and dependencies accurately.
- After all factors were merged, the resulting single factor represented the joint probability distribution of all included variables. This final factor provides a comprehensive model of the combined probabilistic knowledge from the input

factors.

Q3

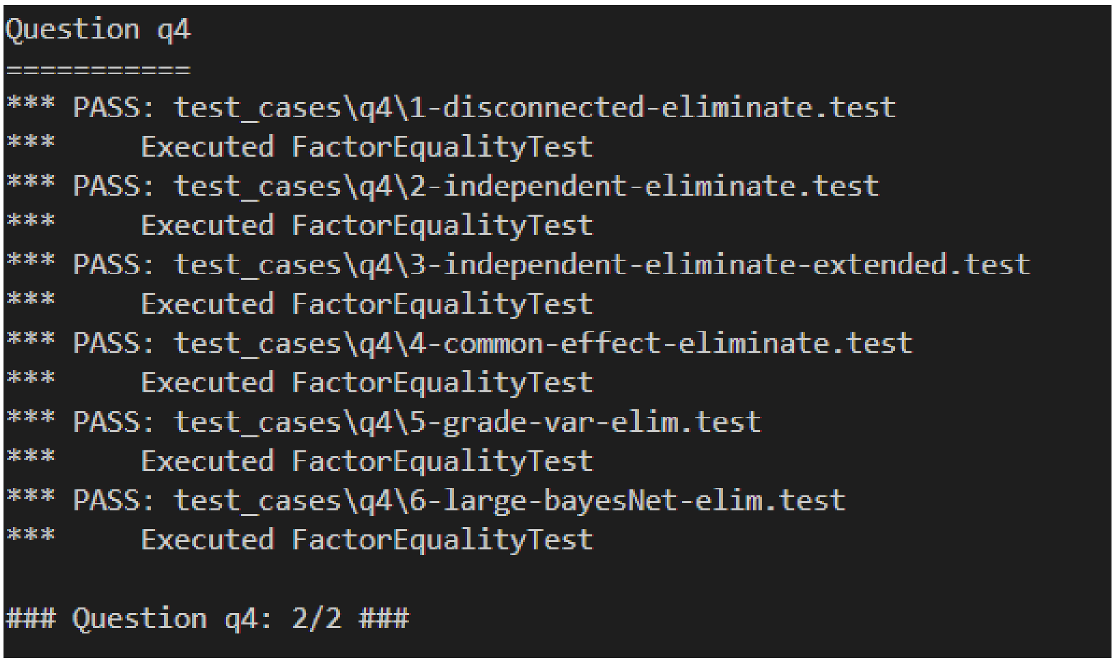


```
Question q3
===========
*** PASS: test_cases\q3\1-simple-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q3\2-simple-eliminate-extended.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q3\3-eliminate-conditioned.test
***     Executed FactorEqualityTest
*** PASS: 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 ###
```

- The set of unconditioned variables is updated by removing the elimination variable. This adjustment refines the factor's scope, focusing on the remaining variables that directly influence the probability distribution without being conditioned by the removed variable. Importantly, the conditioned variables and their domains are left unchanged to maintain existing dependencies and ensure the integrity of the model.
- The probabilities within the new factor are recalculated by aggregating the probabilities from the original factor. Specifically, we iterate over all possible assignments of the original factor, excluding the elimination variable from these assignments. For each combination of remaining variable values, we sum the probabilities corresponding to all potential values of the elimination variable. This process effectively integrates out the eliminated variable, ensuring that its probabilistic effects are encapsulated within the new configuration of the factor.
- After updating the probabilities, the function returns the newly configured factor. This new factor now represents the probability distribution over the remaining variables, having adjusted for the removal of the elimination variable. The distribution thus reflects an updated understanding of variable interactions and dependencies, minus the direct influence of the variable that has been eliminated.

Q4

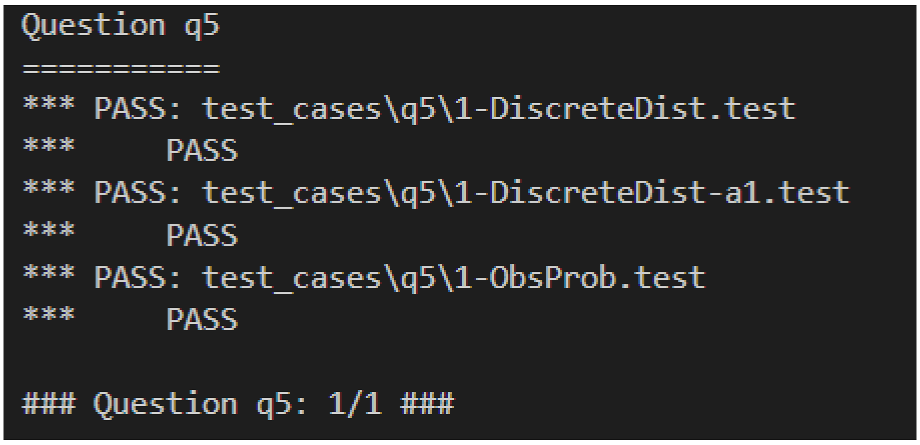


```
Question q4
===========
*** PASS: test_cases\q4\1-disconnected-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q4\2-independent-eliminate.test
***     Executed FactorEqualityTest
*** PASS: test_cases\q4\3-independent-eliminate-extended.test
***     Executed FactorEqualityTest
*** 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 ###
```

- Using bayesNet.getAllCPTsWithEvidence(evidenceDict), we retrieve all Conditional Probability Tables (CPTs) adjusted for the given evidence.
- For each variable in the eliminationOrder, we combine all factors containing that variable by calling the joinFactorsByVariable function. This function specifically consolidates factors by the variable slated for elimination, ensuring all related dependencies are merged into a single factor.
- After joining, the function checks if the resulting factor has more than one unconditioned variable. If this condition is met, the current variable is eliminated using the eliminate function.
- The outcome of the elimination process is then reintegrated into the list of current factors, to be considered in the next iterations.
- Once all variables in the eliminationOrder have been addressed, any remaining factors are consolidated into a single factor using the joinFactors function. This final factor is then normalized using the normalize function to ensure that the total probability sums to one, thereby forming a valid conditional probability distribution.
- The resulting normalized factor, which represents the probabilities of the query variables conditioned on the provided evidence, is returned as the output of the inference process.

Q5



```
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 ###
```

a. normalize

- Uses the total() method to compute the sum of all values in the distribution. If this sum equals zero, the method exits early to prevent division by zero, avoiding undefined behavior.
- If the total is not zero, the method iterates over each key in the distribution, dividing the value associated with each key by the total sum to normalize the values.

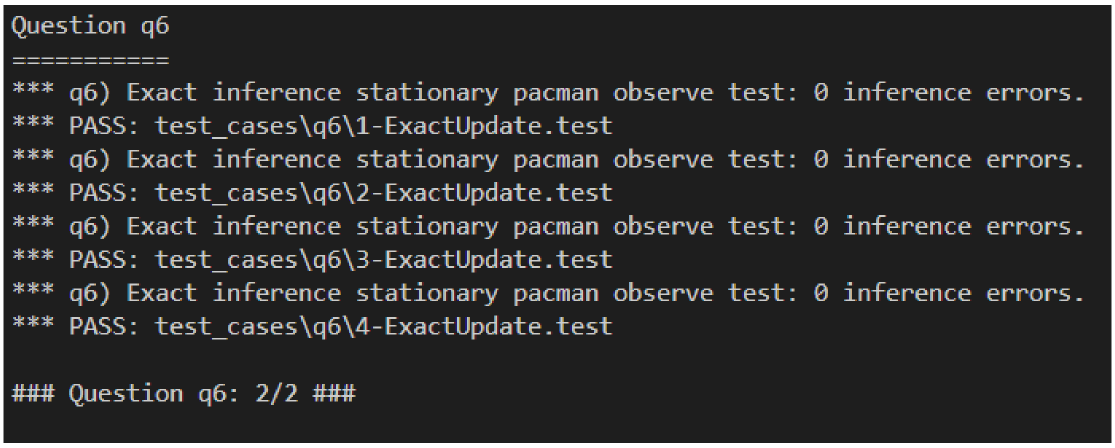
a. sample

- Verify if the distribution is empty or if the sum of all values is zero, returning None in either case as no valid sampling can be performed.
- Generates a random number r between 0 and the total sum of the distribution.
- Iterates over each entry in the distribution, accumulating weights. If the accumulated weight exceeds or equals r, the corresponding key is returned.

b. getObservationProb

- If the ghost's position is the same as the jail position:
  - Returns 1.0 if noisyDistance is None, indicating a correct assumption that no distance should be measurable.
  - Returns 0.0 if noisyDistance is not None, reflecting the impossibility of measuring a distance to a ghost that is jailed.
- If noisyDistance is None and the ghost is not in jail, returns 0.0, indicating an error or missing measurement.
- Calculates the true Manhattan distance between Pacman and the ghost.
- Uses busters.getObservationProbability(noisyDistance, trueDistance) to determine the probability of observing the noisy distance given the actual Manhattan distance.

Q6



```
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 ###
```

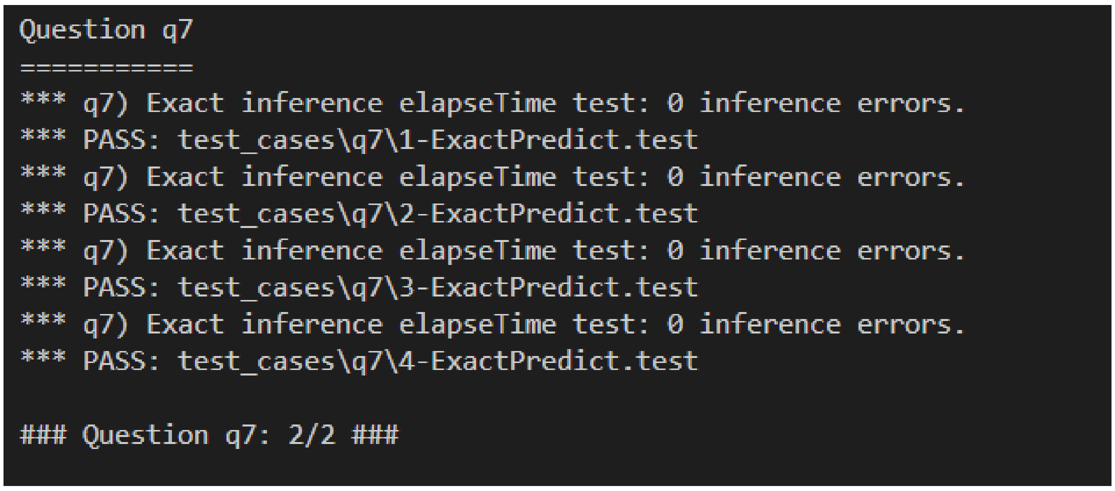
- Initialize the variables :

1. Retrieves the current position of Pacman from the game state.
2. Retrieves the jail position of the ghost to determine if the ghost is captured and thus unobservable.
3. Accesses the current belief state, representing the estimated probability distribution of the ghost's positions.
4. Retrieves a list of all legal positions the ghost can occupy.

- Start updating

1. For each possible position of the ghost, calculates the likelihood using the newly constructed function ‘getObservationProb’.
2. Retrieves the prior belief for each ghost's position before the observation.
3. Updates the belief by multiplying the prior belief for each position by the corresponding calculated likelihood.
4. After updating the beliefs for all positions, the entire belief distribution is normalized to ensure that the sum of the probabilities equals 1.

Q7

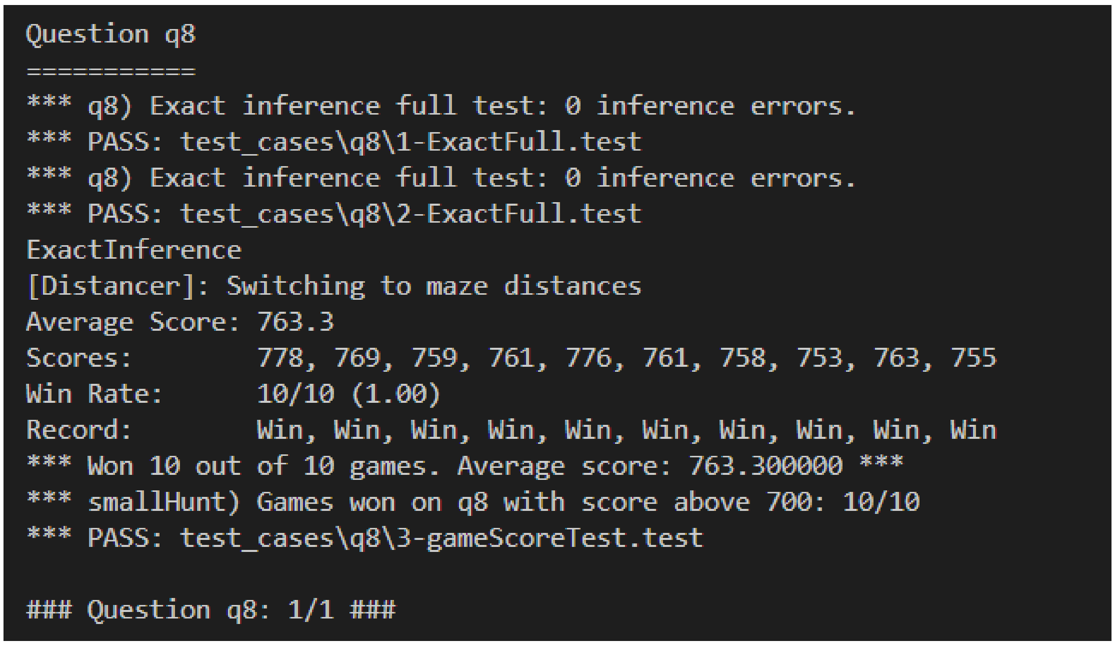


```
Question q7
===========
*** q7) Exact inference elapseTime test: 0 inference errors.
*** PASS: test_cases\q7\1-ExactPredict.test
*** q7) Exact inference elapseTime test: 0 inference errors.
*** PASS: test_cases\q7\2-ExactPredict.test
*** q7) Exact inference elapseTime test: 0 inference errors.
***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

- Create a new, empty DiscreteDistribution object to hold updated beliefs.
- For all positions, use the getPositionDistribution function to retrieve the new position probabilities. This function calculates the likelihood of transitioning to each new position from a current position.
- Update the beliefs for each new position by summing the results obtained from multiplying the existing beliefs by the corresponding new probabilities. This step involves aggregating contributions from all possible previous states to each new state based on the transition probabilities.
- Normalize the new beliefs distribution to ensure that the total of all beliefs equals 1. This step is crucial to maintain the probabilistic integrity of the beliefs, ensuring they represent a valid probability distribution.
- Finally, update the system's belief state with the newly normalized beliefs to reflect the most recent understanding of the state probabilities.

Q8

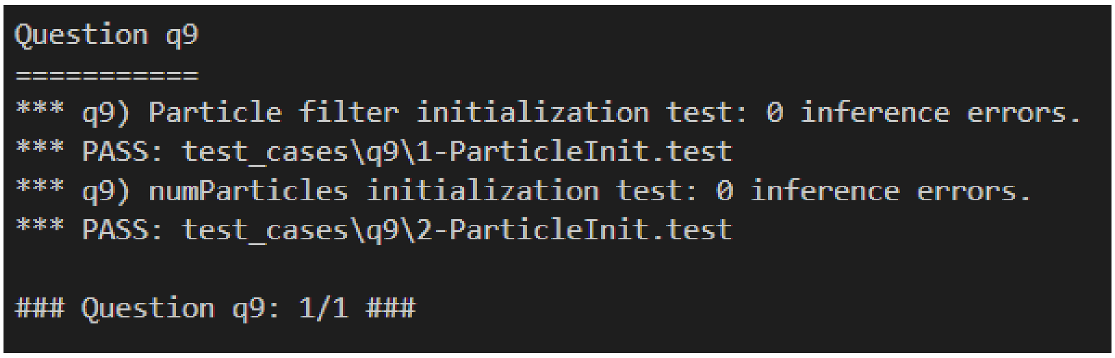


```
Question q8
===========
*** q8) Exact inference full test: 0 inference errors.
*** PASS: test_cases\q8\1-ExactFull.test
*** q8) Exact inference full test: 0 inference errors.
*** PASS: test_cases\q8\2-ExactFull.test
ExactInference
[Distancer]: Switching to maze distances
Average Score: 763.3
Scores:        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 ###
```

1. Retrieve Essential Game Variables:
   - Fetch Pacman's current position from the game state.
   - Gather all possible legal actions Pacman can take from his current position.
   - Identify which ghosts are still active in the game.
   - Access the probability distributions for the positions of all active ghosts.
2. Determine the most likely position for each living ghost by selecting the position with the highest probability from their belief distributions.
3. Identify the closest active ghost to Pacman by comparing the distances from Pacman to each ghost’s most likely position.
4. Analyze and select the best legal action that minimizes the distance between Pacman and the closest ghost.
5. Return the best action, which is the optimal move for Pacman to approach or strategically position himself relative to the nearest ghost.

Q9



```
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 ###
```

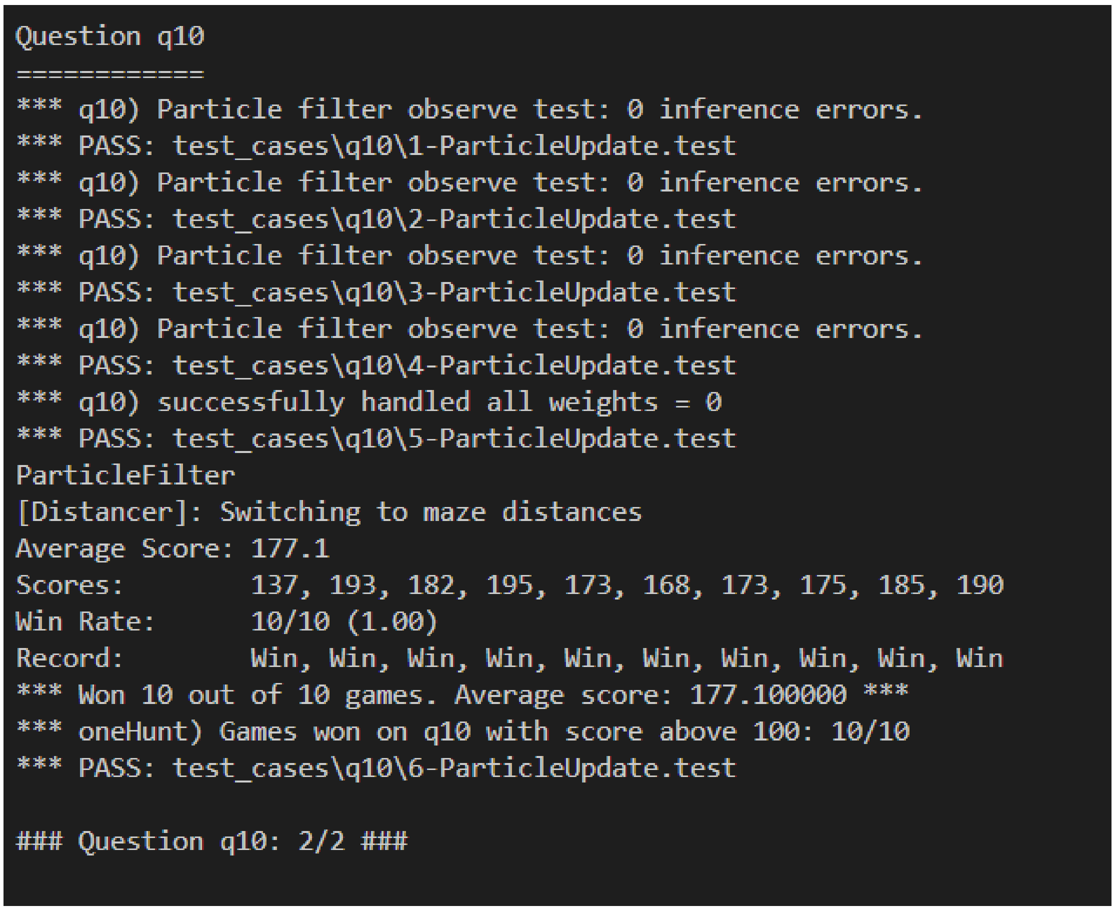
initializeUniformly Function:

- Start with an empty list of particles.
- Retrieve all legal positions where particles can be placed.
- Calculate the number of particles per position by dividing the total number of particles by the number of legal positions.
- Evenly distribute the initial set of particles across each legal position.
- If there are any remaining particles after even distribution (due to division remainder), randomly assign these excess particles to legal positions to ensure all particles are utilized.

getBeliefDistribution Function:

- Create a new DiscreteDistribution to represent the belief distribution.
- For each occurrence of a particle at a position, increment the count in the belief distribution by one.
- Normalize the belief distribution to ensure that the total sums up to one, thereby converting counts into probabilities.

Q10

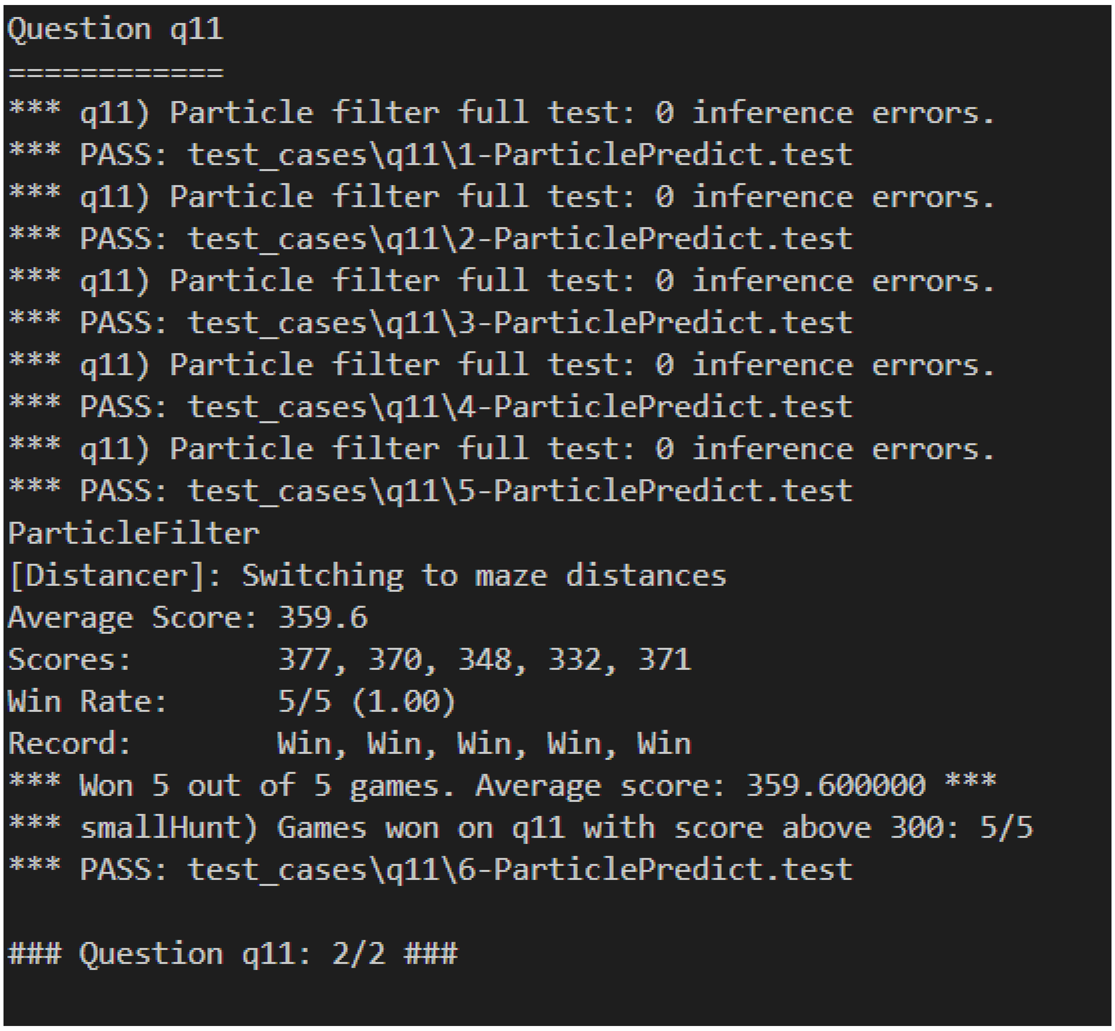


```
Question q10
============
*** q10) Particle filter observe test: 0 inference errors.
*** PASS: test_cases\q10\1-ParticleUpdate.test
*** q10) Particle filter observe test: 0 inference errors.
*** PASS: test_cases\q10\2-ParticleUpdate.test
*** q10) Particle filter observe test: 0 inference errors.
*** 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: 177.1
Scores:        137, 193, 182, 195, 173, 168, 173, 175, 185, 190
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: 177.100000 ***
*** oneHunt) Games won on q10 with score above 100: 10/10
*** PASS: test_cases\q10\6-ParticleUpdate.test

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

- Obtain Pacman's current position and the jail's position from the game state.
- Work with the current set of particles representing potential ghost positions and initialize a new, empty list for updated particles.
- Iterate through each particle, applying the getObservationProb method to calculate the probability of each particle's position relative to the latest observation.
- Store the updated probability values in the new particles list.
- Normalize the new particles list to ensure the sum of all probabilities equals one, maintaining a proper probability distribution.
- Update self.particles with this newly normalized list.
- Check for degenerate scenarios where all new particle probabilities are zero, which can occur if no plausible positions align with the observations.
- If a degenerate case is detected, reinitialize the particles uniformly across all legal positions using the initializeUniformly method to restore diversity to the particle distribution.

Q11



```
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: 359.6
Scores:        377, 370, 348, 332, 371
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
*** Won 5 out of 5 games. Average score: 359.600000 ***
*** smallHunt) Games won on q11 with score above 300: 5/5
*** PASS: test_cases\q11\6-ParticlePredict.test

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

- Begin by initializing a new particle list. This list will be used to store particles once they have been updated based on the new position probabilities.
- For each particle in the existing list, perform the following steps:
- Retrieve the probability distribution of potential new positions using the getPositionDistribution function, which calculates this distribution based on the particle’s current position.
- Sample a position from this distribution to represent the updated state of the particle.
- Append the newly sampled position to the updated particle list.