2024 AI - HW#3 Ghostbusters

生醫電資所 碩一 r12945040 郭思言

- **Show your autograder results and describe the implementation details:**
  - **Overall**

```
Finished at 14:00:34

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

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.
```

  - **Question 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 ###


Finished at 14:02:10

Provisional grades
==================
Question q1: 2/2
------------------
Total: 2/2
```

Implementation details:

1. Define Constants: Pacman, two ghosts, two observations, the x and y ranges for the game, and the maximum noise allowed.
2. Initialize Variables: Initialize an empty list for `variables`, `edges`, and an empty dictionary `variableDomainsDict` to store variable domains.
3. Define Variables and Edges: Add the variables Pacman, Ghost0, Ghost1, Observation0, and Observation1 to the `variables` list. Define the edges between Ghost0 and Observation0, Pacman and Observation0, Pacman and Observation1, and Ghost1 and Observation1.
4. Define Variable Domains: Create a list `value` containing all possible positions (x, y) in the game grid. Assign this list to the variable domains for Pacman, Ghost0, and Ghost1.

5. Define Observation Domains: For each observation variable, create a set (`obsl0` and `obsl1`) to store the possible observed distances. Add 0 to both sets. Calculate the possible observed distances for each position of Pacman and the ghosts, adding `(obsdis + MAX_NOISE)` and `(obsdis - MAX_NOISE)` to the sets if the distance is greater than MAX_NOISE.
6. Assign Observation Domains: Assign the sets `obsl0` and `obsl1` to the variable domains for Observation0 and Observation1, respectively.
7. Create Empty Bayesian Network: Use the `bn.constructEmptyBayesNet` function to create an empty Bayesian network with the defined `variables`, `edges`, and `variableDomainsDict`.

- **Question 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 ###


Finished at 14:09:20

Provisional grades
==================
Question q2: 3/3
------------------
Total: 3/3
```

Implementation details:

1. Combining Unconditioned and Conditioned Variables: Combines the sets of unconditioned variables (`setsOfUnconditioned`) from all input factors into a single set (`setOfUnconditioned`). Combines the sets of conditioned variables (`setsOfConditioned`) from all input factors into a single set (`setOfConditioned`).
2. Identifying Unconditioned and Conditioned Variables: Variables that appear only in `setsOfConditioned` and not in `setsOfUnconditioned` are identified as conditioned variables (`out_setsOfConditioned`). Variables that appear only in `setsOfUnconditioned` and not in `setsOfConditioned` are identified as unconditioned variables (`out_setsOfUnconditioned`). Variables that appear in both `setsOfUnconditioned` and `setsOfConditioned` are also identified as unconditioned variables (`out_setsOfUnconditioned`).
4. Creating a New Factor: A new factor is created with the identified unconditioned and conditioned variables. The `variableDomainsDict` of the

new factor is formed by merging the `variableDomainsDict` of all input factors, ensuring that duplicate keys are removed.

5. Setting Probabilities: For each possible assignment of variables in the new factor: The probability of the assignment is calculated as the product of the probabilities from corresponding rows of the input factors. This is done by iterating over each input factor, multiplying the probabilities of the assignment in each factor, and then setting the probability of the assignment in the new factor.

6. Returning the New Factor

- **Question 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 ###


Finished at 14:43:46

Provisional grades
==================
Question q3: 2/2
------------------
Total: 2/2
```

Implementation details:

1. Removing the Variable: The function removes the `eliminationVariable` from the set of unconditioned variables to obtain the set of unconditioned variables for the new factor. The set of conditioned variables for the new factor remains the same as the input factor.

2. Creating a New Factor: A new factor is created with the updated sets of unconditioned and conditioned variables. The `variableDomainsDict` for the new factor is the same as the input factor.

4. Calculating Probabilities: For each possible assignment of variables in the new factor: The function iterates over all possible values of the `eliminationVariable` and calculates the sum of probabilities for all assignments that have the same values for other variables. This is done by setting the `eliminationVariable` to each value and summing the probabilities from the input factor for all corresponding assignments.

5. The new factor with the updated probabilities is returned.

- **Question 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 ###


Finished at 14:49:28

Provisional grades
==================
Question q4: 2/2
------------------
Total: 2/2
```

Implementation details:

1. Joining and Eliminating Factors:
   (1) The function iterates through the `eliminationOrder` and performs joining and eliminating operations on the factors in `currentFactorsList`.
   (2) For each variable in the `eliminationOrder`, it joins all factors that contain that variable using `joinFactorsByVariable`, creating a new factor (`joinedFactor`).
   (3) If the `joinedFactor` has more than one unconditioned variable, it eliminates the `joinVariable` using `eliminate`, creating an eliminated factor (`eliminatedFactor`).
   (4) The `eliminatedFactor` is appended to the `currentFactorsList`.
2. Returning the Result:
   (1) After all variables in the `eliminationOrder` have been processed, the function normalizes and returns the result of joining all factors in `currentFactorsList`.
   (2) The result is a factor representing the conditional probability distribution `P(queryVariables | evidenceDict)`.

- **Question 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 ###


Finished at 14:56:07

Provisional grades
==================
Question q5: 1/1
------------------
Total: 1/1
```

Implementation details:

5a:

1. normalize:
    (1) Iterates over all keys in the distribution and divides each value by the total value of the distribution, converting them to floating-point numbers to ensure accurate division.
    (2) The method iterates over all keys in the distribution and divides each value by the total value of the distribution, converting them to floating-point numbers to ensure accurate division.
2. sample:
    (3) First creates two lists: `s_seq` to store the keys and `s_weights` to store the normalized weights of each key.
    (4) Then, generates a random number `x` between 0 and 1 and iterates over the keys, decrementing `x` by the weight of each key until `x` is less than or equal to 0.
    (5) When `x` becomes less than or equal to 0, it returns the corresponding key, effectively sampling from the distribution based on the weights.

5b:

1. Noisy Distance is None and Ghost is in Jail: If `noisyDistance` is `None` and the `jailPosition` is equal to the `ghostPosition`, return a probability of 1. This indicates that the ghost is in jail, so the noisy distance observation should be 0 (even if it was mistakenly reported as something else).
2. Noisy Distance is None and Ghost is not in Jail: If `noisyDistance` is `None` and the `jailPosition` is not equal to the `ghostPosition`, return a probability of 0. This indicates that the ghost is not in jail, so the noisy distance observation should not be 0.
3. Noisy Distance is not None and Ghost is in Jail: If `noisyDistance` is not `None` and the `jailPosition` is equal to the `ghostPosition`, return a probability of 0. This indicates that the ghost is in jail, so the noisy distance observation should be 0 (even if it was mistakenly reported as something else).
4. Calculate Observation Probability: If none of the above conditions are met, calculate the observation probability using the `getObservationProbability` function from the `busters` module, which takes the `noisyDistance` and the Manhattan distance between Pacman and the ghost's position as inputs. Return the calculated observation probability.

- **Question 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 ###


Finished at 15:12:10

Provisional grades
==================
Question q6: 2/2
----------------
Total: 2/2
```

Implementation details:

1. Retrieve Pacman and Jail Positions:

(1) Get Pacman's current position using `gameState.getPacmanPosition()`.

(2) Get the jail position using `self.getJailPosition()`.

2. Update Beliefs:

(1) Iterate over all possible ghost positions in `self.allPositions`.

(2) For each ghost position, update the belief by multiplying the current belief for that position by the observation probability calculated using `self.getObservationProb(observation, pacman_position, ghost_position, jail_position)`. This effectively updates the belief based on the observation.

(3) After updating all beliefs, normalize the beliefs using `self.beliefs.normalize()` to ensure they sum to 1.

- **Question 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 ###


Finished at 15:18:13

Provisional grades
==================
Question q7: 2/2
----------------
Total: 2/2
```

Implementation details:

1. Initialization: Create a new empty `DiscreteDistribution` object `new_prob` to store the predicted beliefs after the time step.

2. Update Beliefs for Each Possible Ghost Position:

(1) Iterate over all possible ghost positions in `self.allPositions`. For each

position, obtain the position distribution `new_pos_dist` using `self.getPositionDistribution(gameState, position)`.
(2) This distribution represents the likely new positions for the ghost from the current position. For each possible new position `pos` in `new_pos_dist`, update the belief in `new_prob` by adding the product of the probability of `pos` in `new_pos_dist` and the current belief for the ghost position.
(3) Repeat this process for all possible new positions for the current ghost position.
3. Normalize and Update Beliefs:
(1) After updating `new_prob` with the predictions for all ghost positions, normalize `new_prob` to ensure that the sum of probabilities is 1.
(2) Update `self.beliefs` to be equal to `new_prob`, effectively updating the agent's beliefs based on the predicted positions after the time step.

- **Question 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 ###


Finished at 15:30:09

Provisional grades
==================
Question q8: 1/1
------------------
Total: 1/1
```

Implementation details:

1. Compute Most Likely Ghost Positions: For each living ghost, find its most likely position using `argMax()` on the corresponding position distribution in `livingGhostPositionDistributions`. Store these most likely positions in the `targets` list.
2. Choose Closest Ghost: Initialize `true_target` as the first target in the `targets` list. Iterate over all targets and compute the distance from Pacman's position to each target using the `distancer.getDistance()` method. Update `true_target` to the closest target based on the computed distances.
3. Choose Action to Reach Closest Ghost: Initialize `best_action` as the first legal action in the `legal` list. Iterate over all legal actions and compute the

distance from the successor position (position after taking the action) to the `true_target` using `distancer.getDistance()`. Update `best_action` to the action that brings Pacman closest to the `true_target` based on the computed distances.

4. Return Best Action: Return the `best_action`, which is the action that brings Pacman closest to the closest ghost according to maze distance.

- **Question 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 ###


Finished at 15:40:10

Provisional grades
==================
Question q9: 1/1
------------------
Total: 1/1
```

Implementation details:

1. initializeUniformly:

(1) Initialize an empty list `self.particles` to store the particles.

(2) Iterate `self.numParticles` times to create the particles.

(3) Append particles to `self.particles` by cycling through `self.legalPositions` using the modulo operator (`%`) to ensure even distribution.

(4) Shuffle the list of particles using `random.shuffle(self.particles)` to ensure uniformity.

2. getBeliefDistribution:

(1) Initialize a `DiscreteDistribution` object `belief_distribution` to store the belief distribution.

(2) Iterate over each particle in `self.particles`.

(3) Increment the count for the corresponding position in `belief_distribution` for each particle.

(4) Normalize `belief_distribution` to ensure that the sum of probabilities is 1.

(5) Return the normalized `belief_distribution`.

- **Question 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: 182.9
Scores:        190, 189, 196, 195, 139, 197, 149, 194, 187, 193
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: 182.900000 ***
*** oneHunt) Games won on q10 with score above 100: 10/10
*** PASS: test_cases/q10/6-ParticleUpdate.test

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


Finished at 15:40:49

Provisional grades
==================
Question q10: 2/2
------------------
Total: 2/2
```

Implementation details:

1. Initialization:

(1) Get Pacman's current position using `gameState.getPacmanPosition()`.

(2) Get the jail position using `self.getJailPosition()`.

2. Update Beliefs with Observation:

(1) Obtain the current belief distribution `predictions` using `self.getBeliefDistribution()`.

(2) Iterate over all possible ghost positions in `self.allPositions`.

(3) For each ghost position, update the belief in `predictions` by multiplying the current belief for that position by the observation probability calculated using `self.getObservationProb(observation, pacman_position, ghost_position, jail_position)`.

3. Handle Special Case - Reinitialize Particles:

(1) Check if the total weight of the belief distribution is 0 using `predictions.total() == 0`.

(2) If all particles have zero weight, reinitialize the list of particles by calling `self.initializeUniformly(gameState)`.

4. Normalize Beliefs: If the total weight is not 0, normalize the belief distribution using `predictions.normalize()`.

5. Resample Particles: Update `self.particles` by resampling from the belief distribution. For each particle, sample a position from the belief distribution using `predictions.sample()` and repeat this process `self.numParticles` times.

- **Question 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: 378.2
Scores:        370, 383, 376, 379, 383
Win Rate:      5/5 (1.00)
Record:        Win, Win, Win, Win, Win
*** Won 5 out of 5 games. Average score: 378.200000 ***
*** smallHunt) Games won on q11 with score above 300: 5/5
*** PASS: test_cases/q11/6-ParticlePredict.test

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


Finished at 15:44:17

Provisional grades
==================
Question q11: 2/2
------------------
Total: 2/2
```

Implementation details:

1. Sampling Next State:

(1) Iterate over each particle in `self.particles`.

(2) For each particle, use `self.getPositionDistribution(gameState, particle)` to get the position distribution for the next state given the current game state and the particle's current state.

(3) Sample a new position from the position distribution using `.sample()`.

(4) Update the particle with the sampled new position.