# Question 1

## Problem Formulation

### Component of Markov Decision Process

|  |  |
| --- | --- |
| State, S | All possible position the Pacman can occupy, which represent in a x-y coordinate, where s∈S denote any individual state |
| Action, A | {North, South, East, West}, where denote any individual action |
| Goal State | Either one of the food dot positions on the maze |
| Transition function, | With a noise value of 0.2, there is an 80% chance of moving in the intended direction, and a 10% chance of sliding to both the left and right. |
| Reward Function, | A reward of +510 for goal state, and reward of -500 for end state (state where Pacman die), no reward for unreachable state (wall) and all the non-terminal state has a reward of -1. |
| Discount Factor (γ) | Number between 0 and 1, represent how much agent values future rewards over immediate rewards, this value is differed between different map size, which chosen based on the experiment below. |

Implementation

Value iteration updates the value function V(s) iteratively for each state based on the Bellman equation. Here’s an outline of the algorithm

1. **Initialize** a 2D zero array same shape (width and height) as the maze, .
2. **Update**  for each state using the Bellman Optimality equation

This equation is divided into two parts,

* + - The summation which calculates the q value for a given state and action, represent by the pseudocode below
  + The maximum value for different a,

***FUNCTION*** *computeQValueFromValues(s, a):*

*q ← 0*

*transitions ← possible (s', p) reachable from s by a*

*Loop for each (s', p) in transition*

*reward ← reward given for the combination of (s,a,s')*

*q ← p \* (reward + γ \* )*

*q += probability \* (reward + discount \* v(s))*

*RETURN q\_value*

*END*

Find the q value with function above for all possible actions from current state and find the maximum among all the actions.

1. **Loop until convergence ( repeating step 2 )**
   * In our case, the number of iterations is specified, so just loop for the iteration specified for each map:
     + **Small: 50 iterations**
     + **Medium: 100 iterations**
     + **Large: 150 iterations**

***FUNCTION*** *valueIteration():*

*Loop for each iteration specified*

*Loop for each state s*

*A ← possible action from current state*

*V(s) ← computeQValueFromValues(s, a)*

*END*

1. **Extract the Optimal Policy**:
   * After convergence, the optimal policy can be extract from the . It can be derived by choosing the action that maximizes the expected cumulative reward for each state.
   * This equation can be represented by the pseudocode below

***FUNCTION*** *computePolicyFromValues(s):*

*A ← possible action from current state*

*max\_a ← a where computeQValueFromValues(s, a)*

*RETURN a*

*END*

## Experiment on Gamma value

**Objective**:

To investigate how varying the discount factor affects the convergence speed and policy quality in a value iteration process.

**Background**:

The discount factor (γ) in an MDP determines the importance of future rewards: higher values prioritize long-term gains and often lead to more optimal policies but slower convergence, while lower values focus on immediate rewards, resulting in faster convergence but potentially suboptimal policies.

**Metrics**:

* Convergence Speed: number of iterations to reach a threshold
* Policy quality: average score and win rate of Pacman over 40 runs

**Parameters**:

* Discount Factors: {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99, 0.9999}
* Convergence criteria: , this threshold is just a checkpoint on the program without stopping the program as the iteration is already fixed.

**Methodology:**

1. Run the Q1Agent with all the discount factors state above and observe its average score and number of iteration for it to reach the threshold.
2. Run step 1 for each map size {bigMaze, mediumMaze and smallMaze}.

Determine optimal gamma value:

* Calculate the average score for each map size across all discount value.
* Then for each discount value, calculate how many maps had score over the average score calculated above.

### Small Maze

**Results:**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0.2** | **0.3** | **0.4** | **0.5** | **0.6** | **0.7** | **0.8** | **0.9** | **0.99** | **0.9999** |
| **small1\_1** | 10 | 12 | 15 | 19 | 22 | 25 | 28 | 32 | 46 | 47 |
| **small1\_2** | 10 | 12 | 15 | 19 | 22 | 25 | 28 | 32 | 46 | 47 |
| **small1\_3** | 10 | 12 | 15 | 19 | 22 | 25 | 28 | 32 | 46 | 47 |
| **small1\_4** | 10 | 12 | 15 | 19 | 22 | 25 | 28 | 32 | 46 | 47 |
| **small2\_1** | 10 | 12 | 16 | 20 | 24 | 23 | 27 | 30 | - | - |
| **small2\_2** | 10 | 12 | 16 | 20 | 24 | 23 | 27 | 30 | - | - |
| **small2\_3** | 10 | 12 | 16 | 20 | 24 | 23 | 27 | 30 | - | - |
| **small2\_4** | 10 | 12 | 16 | 20 | 24 | 23 | 27 | 30 | - | - |
| **small3\_1** | 10 | 12 | 15 | 18 | 24 | 27 | 31 | 39 | - | - |
| **small3\_2** | 10 | 12 | 15 | 18 | 24 | 27 | 31 | 39 | - | - |
| **small3\_3** | 10 | 12 | 15 | 18 | 24 | 27 | 31 | 39 | - | - |
| **small3\_4** | 10 | 12 | 15 | 18 | 24 | 27 | 31 | 39 | - | - |

*Table 1: Number of iterations to reach the convergence threshold on Small Maze*

From the table above, it is obvious that 0.99 is the discount factor that result the convergence to happen at the edge of the iteration specified. Hence it can be chosen as a candidate as the optimal gamma value for small size maze. For further confirmation, we compare the average score on to

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Map Instance** | **0.8** | **0.9** | **0.95** | **0.99** | **0.9999** | **Average Score of Maze** |
| **small1\_1** | 224.275 | 224.275 | 224.275 | 486.35 | 486.35 | 329.105 |
| **small1\_2** | 296.45 | 296.45 | 296.45 | 296.45 | 296.45 | 296.45 |
| **small1\_3** | 242.2 | 242.2 | 460 | 484.9 | 484.9 | 382.84 |
| **small1\_4** | 220.95 | 220.95 | 464.15 | 488.725 | 488.725 | 376.7 |
| **small2\_1** | 164 | 164 | 163.475 | 322.025 | 322.025 | 227.105 |
| **small2\_2** | 95.475 | 120.3 | 118.575 | 92.325 | 92.325 | 103.8 |
| **small2\_3** | 276.525 | 276.525 | 276.525 | 319.05 | 319.05 | 293.535 |
| **small2\_4** | 124.55 | 124.55 | 219.25 | 410.275 | 410.275 | 257.78 |
| **small3\_1** | 264.625 | 263.975 | 484.6 | 484.6 | 484.6 | 396.48 |
| **small3\_2** | 259.525 | 259.175 | 479.8 | 479.8 | 479.8 | 391.62 |
| **small3\_3** | 196.65 | 196.65 | 196.65 | 348.875 | 348.875 | 257.54 |
| **small3\_4** | 226.025 | 499.3 | 499.3 | 499.3 | 499.3 | 444.645 |
| **Number of instances larger than average** | 1 | 3 | 7 | 11 | 11 |  |

*Table 2: Average Score on Small Maze on*

*Graph 1: Average Score on Small Maze on*

From the Graph 1, the purple line and blue line has the same value for every maze instance, hence the blue line is completely covered by the purple line (). It can easily observe that each data point of the blue line is above other lines. It shows that the discount factor **0.99** is optimal for small size map.

Besides that, in Table 2, with , it has the most map instance larger the average score on each maze. 11 of the map instances exclude small2\_2 has score larger than the average score on each maze size.

### Medium Maze

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.99 | 0.9999 |
| medium1\_1 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium1\_2 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium1\_3 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium1\_4 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium2\_1 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium2\_2 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium2\_3 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium2\_4 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium3\_1 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |
| medium3\_2 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |
| medium3\_3 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |
| medium3\_4 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |

*Table 3: Number of iterations to reach the convergence threshold (Medium Maze)*

From the table above, it is obvious that 0.9999 is the discount factor that result the convergence to happen at the edge of the iteration specified. Hence it can be chosen as a candidate as the optimal gamma value for small size maze. For further confirmation, we compare the average score on to

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Map Instance** | **0.8** | **0.9** | **0.99** | **0.9999** | **Average Score of Maze** |
| **medium1\_4** | 91.2 | 91.2 | 407.8 | 407.8 | 217.84 |
| **medium2\_1** | 408.95 | 408.95 | 476.225 | 468.175 | 434.25 |
| **medium2\_2** | 136.8 | 136.8 | 136.8 | 174.125 | 144.265 |
| **medium2\_3** | 196.2 | 196.2 | 196.2 | 245.725 | 206.105 |
| **medium2\_4** | 262.675 | 262.675 | 481.05 | 481.05 | 350.025 |
| **medium3\_1** | 417.85 | 210.425 | 372.175 | 417.85 | 367.23 |
| **medium3\_2** | 441.5 | 211.525 | 441.5 | 441.5 | 395.505 |
| **medium3\_3** | 474.925 | 474.925 | 474.925 | 474.925 | 474.925 |
| **medium3\_4** | 413.55 | 189.85 | 349.375 | 413.55 | 355.975 |
| **medium1\_1** | 133.05 | 133.05 | 350.2 | 350.2 | 219.91 |
| **medium1\_2** | 60.125 | 60.125 | 402.575 | 402.575 | 197.105 |
| **medium1\_3** | 113.275 | 113.275 | 380.3 | 380.3 | 220.085 |
| **Number of instances larger than average** | 4 | 1 | 9 | 12 |  |

*Table 4: Average Score on Medium Maze on different Discount Factor*

*Graph 2: Average Score on Medium Maze on different Discount Factor*

From the Graph 2, it can easily observe that most of the data point of the purple line () is above other lines. It shows that the discount factor **0.9999** is optimal for small size map.

Besides that, in Table 4 with , all the map instances has a score larger than the average score calculated.

### Big Maze

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.99 | 0.9999 |
| medium1\_1 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium1\_2 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium1\_3 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium1\_4 | 10 | 13 | 16 | 19 | 25 | 31 | 37 | 43 | 59 | 60 |
| medium2\_1 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium2\_2 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium2\_3 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium2\_4 | 10 | 12 | 16 | 20 | 24 | 31 | 46 | 41 | 0 | 0 |
| medium3\_1 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |
| medium3\_2 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |
| medium3\_3 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |
| medium3\_4 | 10 | 13 | 16 | 21 | 28 | 39 | 60 | 64 | 86 | 0 |

*Table 5: Number of iterations to reach the convergence threshold (Big Maze)*

From the table above, it is obvious that 0.9999 is the discount factor that result the convergence to happen at the edge of the iteration specified. Hence it can be chosen as a candidate as the optimal gamma value for small size maze. For further confirmation, we compare the average score on to

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Map Instance** | 0.7 | 0.8 | 0.99 | 0.9999 | **Average Score of Maze** |
| big1\_1 | 151.575 | 148.475 | 284.75 | 301.875 | 221.66875 |
| big1\_2 | 154.2 | 123.425 | 308.75 | 329.45 | 228.95625 |
| big1\_3 | 250.475 | 250.475 | 303.9 | 351.05 | 288.975 |
| big1\_4 | 226.025 | 226.025 | 226.025 | 377.925 | 264 |
| big2\_1 | 156.15 | 156.15 | 462.275 | 462.275 | 309.2125 |
| big2\_2 | 258.975 | 258.975 | 326.1 | 456.55 | 325.15 |
| big2\_3 | 345.55 | 345.55 | 342.675 | 474.5 | 377.06875 |
| big2\_4 | 361.225 | 361.225 | 478.2 | 478.2 | 419.7125 |
| big3\_1 | 168.85 | 168.85 | 168.85 | 168.85 | 168.85 |
| big3\_2 | 10.025 | 10.025 | 10.025 | 10.025 | 10.025 |
| big3\_3 | 83.35 | 57.55 | 83.35 | 181.075 | 101.33125 |
| big3\_4 | 178.375 | 178.375 | 178.375 | 198.625 | 183.4375 |
| **Number of instances larger than average** | 2 | 2 | 8 | 12 |  |

*Table 6: Average Score on Big Maze on different Discount Factor*

*Graph 3: Average Score on Medium Maze on different Discount Factor*

From the Graph 3, it can easily observe that most of the data point of the purple line () is above other lines. It shows that the discount factor **0.9999** is optimal for small size map.

Besides that, in Table 5 with , all the map instances has a score larger than the average score calculated.

**Discussion**:

The relationship between discount factor (γ) and convergence speed reveals that higher γ values, such as 0.99 and 0.9999, generally lead to slower convergence but produce policies with higher reward scores. This effect is especially prominent in larger mazes, where convergence often nears or exceeds the iteration limits. Lower discount factors (e.g., 0.8, 0.9) allow faster convergence by focusing on more immediate rewards but sacrifice the quality of the long-term policy, as reflected in the lower reward scores across maze instances.

With higher γ values, the policy better optimizes for long-term rewards, evident from consistently higher scores in medium and big mazes, though at the expense of computational efficiency. This presents a trade-off: while increasing γ enhances policy quality by valuing distant rewards, it may also cause significantly slower convergence in larger state spaces.

**Conclusion:**

1. **Impact of Discount Factor on Convergence and Policy Quality**: Higher γ values slow convergence but produce better policies by emphasizing long-term rewards, with the effects more pronounced in larger mazes.
2. **Choosing Optimal γ**: For small mazes, **γ=0.99** balances convergence speed and policy quality. For medium and large mazes, **γ=0.9999** maximizes scores but requires more iterations, highlighting the trade-off between policy quality and convergence time.

# Question 2

## Problem Formulation

### Component of Markov Decision Process

|  |  |  |
| --- | --- | --- |
| State, S | All possible position the Pacman can occupy, which represent in a x-y coordinate, where s∈S denote any individual state | |
| Action, A | {North, South, East, West}, where denote any individual action | |
| Goal State | Either one of the food dot positions on the maze | |
| Reward Function, | A reward of +510 for goal state, and reward of -500 for end state (state where Pacman die), no reward for unreachable state (wall) and all the non-terminal state has a reward of -1. | |
| Discount Factor (γ) | Number between 0 and 1, represent how much agent values future rewards over immediate rewards | These values are differed between different map size, which chosen based on the experiment below. |
| Learning Rate (α) | Number between 0 and 1 that determines how much new information overrides old information in Q-value update. |
| Exploration Rate (ϵ) | Number between 0 and 1 that determines the probability of the agent choosing a random action rather than selecting action with higher Q-value. |

Implementation

Value iteration updates the value function V(s) iteratively for each state based on the Bellman equation. Here’s an outline of the algorithm

1. **Initialize** Q-table, which is a 3D NumPy array. Row and Columns represent a state in maze, within a cell, it then separated into 4 smaller cells, which represent 4 different actions {East, West, North, South}. The Q-value is then store within these cells.
2. For each episode:
   * Start from the initial position
   * Choose an action using exploration strategy

***FUNCTION*** *computeActionFromQValues(s):*

*A ←all legal action from current state s*

***if*** *A is none* ***then***

***RETURN*** *0*

*max\_a ← a with largest Q(s,a) for each a in A*

***RETURN*** *a*

***END***

***FUNCTION*** *epsilonGreedyActionSelection(s):*

*A ← all legal action from current state s*

*a ← none*

***if*** *A is none* ***then***

***RETURN*** *a*

***if*** *flipCoint with probability* ϵ *is true* **then**

*a ← random action among A*

***else***

*a ← computeActionFromQValues(s)*

***RETURN*** *a*

***END***

* + Update Q-value, Q(s,a) for the new action taken

***FUNCTION*** *computeValueFromQValues(s):*

*A ←all legal action from current state s*

***if*** *A is none* ***then***

***RETURN*** *0*

*q ← largest Q(s,a) for each a in A*

***RETURN*** *q*

***END***

***FUNCTION*** *update(s,a,s’):*

*Q(s,a) ←*

**Part 2: Testing for Q-Learning**

1. **Discount Factor (γ\gammaγ)**
   * **Objective**: Test the effect of different discount factors on the Q-learning policy quality and learning stability.
   * **Range**: Similar to value iteration, test values like 0.5, 0.7, 0.9, and 0.99.
   * **Metrics**:
     + **Convergence Speed**: Measure the number of episodes needed for the algorithm to stabilize.
     + **Policy Quality**: Evaluate the final policy for optimality in terms of cumulative reward or path efficiency.
   * **Expected Outcome**:
     + Higher γ\gammaγ values may improve policy quality but could increase training episodes and introduce instability in larger mazes.
     + Lower γ\gammaγ values might yield faster convergence but could result in less optimal paths.
2. **Learning Rate (α\alphaα)**
   * **Objective**: Investigate the impact of different learning rates on the speed and stability of Q-learning convergence.
   * **Range**: Test values such as 0.1, 0.3, and 0.5.
   * **Metrics**:
     + **Stability**: Measure the variability in episode reward over time, indicating stability.
     + **Convergence Time**: Track the number of episodes needed to reach a stable policy.
   * **Expected Outcome**:
     + Higher learning rates may lead to faster learning but could introduce more fluctuations in the policy.
     + Lower learning rates tend to smooth learning, potentially requiring more episodes to converge.
3. **Exploration Rate (ϵ\epsilonϵ) and Decay Strategy**
   * **Objective**: Analyze the effect of different exploration rates and decay strategies on learning efficiency.
   * **Range**: Initial ϵ\epsilonϵ values (e.g., 1.0, 0.5) and decay strategies (e.g., linear decay, exponential decay).
   * **Metrics**:
     + **Exploration-Exploitation Balance**: Assess how quickly Q-learning finds an optimal policy as ϵ\epsilonϵ decays.
     + **Policy Quality and Convergence**: Measure how the final policy performs in terms of reward after different ϵ\epsilonϵ decay schedules.
   * **Expected Outcome**:
     + A faster decay may cause Q-learning to converge quickly but risk suboptimal solutions.
     + A slower decay can enhance exploration, which is often beneficial in larger mazes.
4. **Episode Length and Total Steps to Convergence**
   * **Objective**: Observe the number of episodes and the average steps per episode needed for convergence.
   * **Metrics**:
     + **Convergence Episodes**: Track the total number of episodes needed to reach a stable, optimal policy.
     + **Steps per Episode**: Measure the average number of steps per episode as an indicator of policy efficiency.
   * **Expected Outcome**:
     + Longer episode lengths can indicate a need for more exploration, while a rapid decline in steps per episode typically signifies approaching optimal policy.
5. **Maze Size and Complexity**
   * **Objective**: Test how well Q-learning adapts to larger mazes and more complex structures.
   * **Range**: Small (5x5), Medium (10x10), Large (20x20).
   * **Metrics**:
     + **Convergence Episodes and Stability**: Compare the number of episodes and learning stability across different maze sizes.
     + **Policy Optimality**: Evaluate how the final policy performs in terms of finding the optimal path or achieving the highest possible reward.
   * **Expected Outcome**:
     + Q-learning may require more episodes and a careful balance of γ\gammaγ, α\alphaα, and ϵ\epsilonϵ parameters for larger mazes to achieve reliable convergence and policy quality.

# Question 3