ML ISE

Great! Let me break it down step by step in a beginner-friendly way so you fully understand what your guide is asking:

**✅ What you have done so far:**

You’ve implemented **Q-learning** in a simple **5x5 GridWorld** with a few obstacles. You trained a robot (agent) to move from the start to the goal, and you’re tracking:

* Rewards
* Steps per episode
* Success rate
* Best paths

**🔁 Now what you need to do:**

Your guide wants you to make the project more **realistic, challenging, and complete**. Here’s what that means:

**1. 🚀 Increase the Environment Complexity**

You can change:

* **Grid size** (e.g. from 5x5 to 10x10 or 15x15)
* **Add more obstacles** or arrange them in more challenging patterns
* **Change the goal position**
* You can even add **multiple goals** or **penalty zones**

🧠 Why?

* This makes the environment more challenging, and helps test how well each algorithm performs in more realistic scenarios.

**2. 🔧 Add Hyperparameter Tuning**

Hyperparameters in your Q-learning code are:

| **Parameter** | **Meaning** | **Example** |
| --- | --- | --- |
| alpha | Learning rate – how fast the agent updates knowledge | 0.1, 0.2, 0.5 |
| gamma | Discount factor – how much it values future rewards | 0.8, 0.9, 0.99 |
| epsilon | Exploration rate – how much it tries random actions | 0.1, 0.2, 0.3 |

**Tuning** means: you try different combinations of these values and see which gives the best results.

You can try using:

* Grid Search (try all combinations manually or with loops)
* Plot how different values affect success, rewards, etc.

🧠 Why?

* Good tuning can make a weak algorithm perform much better.

**3. 🤖 Repeat for All Four Algorithms**

You need to do all of the above not just for Q-learning, but also:

1. **SARSA** (a slightly different way of updating Q-values)
2. **DQN** (uses deep neural networks)
3. **D3QN** (improves DQN with double Q-learning and dueling architecture)

You’ll implement and tune all four on the same environments.

**4. 📊 Do Comparative Analysis**

Make a final **report or notebook** where you compare all algorithms on things like:

* Success rate
* Total rewards
* Steps per episode
* Training time
* Stability of learning
* How they react to increased grid size or more obstacles
* How sensitive they are to hyperparameter tuning

**Visualizations you can use:**

* Line plots (reward/steps vs episodes)
* Bar charts (compare average performance)
* Heatmaps (Q-values)
* Path visualizations

**🧩 Example Plan:**

| **Algorithm** | **Grid Sizes** | **Obstacles** | **Tuned Params** | **Visualizations** |
| --- | --- | --- | --- | --- |
| Q-Learning | 5x5, 10x10, 15x15 | Simple → Complex | α, γ, ε | ✔ |
| SARSA | 5x5, 10x10, 15x15 | Simple → Complex | α, γ, ε | ✔ |
| DQN | Same | Same | Learning rate, batch size, epsilon decay, etc | ✔ |
| D3QN | Same | Same | Same | ✔ |

If you want, I can help you:

* Create a **hyperparameter tuning setup**
* Design a **bigger GridWorld**
* Implement the other algorithms (SARSA, DQN, D3QN)
* Build comparison charts

Let me know what you want to tackle next!

Flow for all 4 algorithms

1) level 1 :basic implementation with

For **Level 1** in your Q-learning implementation, you've already covered the main parameters well. Here's a more detailed breakdown and a few additional aspects you could consider mentioning:

**1. Environment Setup (GridWorld Details):**

* **Grid Size**: 5x5 grid (total 25 states).
* **Start State**: The agent begins at the top-left corner, at coordinate (0, 0).
* **Goal State**: The agent's objective is to reach the bottom-right corner, at coordinate (4, 4).
* **Obstacles**: The agent cannot move through these cells and will receive a negative reward for trying to move into them. Obstacles are placed at positions: [(1, 1), (2, 2), (3, 1)].

**2. Q-learning Hyperparameters:**

* **Episodes (500)**: This is the number of training iterations the agent will go through to improve its policy. The higher the number, the better it can learn, but it will also take longer.
* **Learning Rate (α = 0.1)**: This determines how much the agent adjusts its Q-values based on new experiences. If α is too high, it might "forget" older knowledge, while if it's too low, it may learn too slowly.
* **Discount Factor (γ = 0.9)**: The agent needs to account for future rewards. A higher γ means the agent cares more about future rewards, while a lower γ means it focuses more on immediate rewards.
* **Exploration Rate (ε = 0.2)**: This defines how often the agent will choose a random action instead of exploiting what it knows. With ε = 0.2, 20% of the time, the agent will explore random actions, and the rest of the time, it will follow the best-known action according to its Q-table.

**3. Q-table:**

* **Q-table initialization**: Initially, the Q-table is filled with zeros, representing that the agent doesn't know any information about the environment. The Q-table is updated over time as the agent learns.

**4. Action Space:**

* The agent has 4 possible actions: up, down, left, and right. These correspond to how the agent can move within the grid.
* Action selection follows an **epsilon-greedy policy**:
  + With probability epsilon, the agent explores by choosing a random action.
  + With probability 1 - epsilon, the agent exploits by choosing the action with the highest Q-value for the current state.

**5. Reward System:**

* **Goal state (4, 4)**: If the agent reaches the goal, it receives a reward of **+10**.
* **Obstacle**: If the agent attempts to move into an obstacle, it stays in the same position and receives a **penalty of -5**.
* **Empty space**: If the agent moves to a free cell, it gets a **penalty of -1** to encourage the agent to find the shortest path.

**6. Training Loop:**

* The agent iterates through episodes. Each episode involves the agent starting from the initial state, selecting actions, receiving rewards, and updating its Q-table based on the reward received and the estimated future rewards.

**7. Evaluation:**

* **Total Reward per Episode**: Tracks how much reward the agent accumulates in each episode.
* **Steps per Episode**: Tracks how many steps the agent takes to reach the goal (if it does).
* **Success Rate**: The percentage of episodes where the agent successfully reaches the goal.
* **Training Time**: Measures how long it takes for the agent to complete all the episodes.
* **Robot path visualization**
* **Heatmap for q table**

2) Level 2 inc Complextity

 **Increase the grid size** — makes the world bigger.

 **Add more obstacles** — forces the robot to think harder.

 **keep the start as 0,0 and goal position end of grid**

**✅ Environment Setup (Same as Q-learning Level 2)**

* **Grid Size:** 10x10
* **Start:** (0, 0)
* **Goal:** (9, 9)
* **Obstacles:** 20 randomly placed obstacles (excluding start/goal)

**✅ SARSA Parameters (Match Q-learning)**

* **Episodes:** 500
* **Alpha (Learning rate):** 0.1
* **Gamma (Discount factor):** 0.9
* **Epsilon (Exploration rate):** 0.2

**✅ Required Outputs for Comparison**

Make sure your SARSA Level 2 prints and visualizes:

* **Success Rate**
* **Training Time**
* **Average Steps per Episode**
* **Average Reward per Episode**
* **Q-table Heatmap**
* **Reward per Episode Plot**
* **Steps per Episode Plot**
* **Path Visualization**

3)level 3 : hyperparameter tunning

Result of level 2 and 3

The results you're seeing for **Level 3** (with hyperparameter tuning) are actually showing an improvement in terms of **steps per episode** (lower steps), but the **average reward** is still negative. It means the agent is reaching the goal faster but still facing penalties during its path.

**Summary of Results:**

* **Level 3 (with tuning)**:
  + **Success Rate**: 100.00% (good, the agent is successfully reaching the goal)
  + **Training Time**: 0.06 seconds (fast training, which is good)
  + **Avg Steps per Episode**: 25.03 (better than level 2)
  + **Avg Reward per Episode**: -16.34 (still negative, but reducing)
* **Level 2 (without tuning)**:
  + **Success Rate**: 100.00% (good, the agent is successful)
  + **Training Time**: 0.10 seconds
  + **Avg Steps per Episode**: 44.48 (higher than level 3)
  + **Avg Reward per Episode**: -38.81 (more negative than level 3)

**What can we conclude:**

* **Level 3 is more efficient** than Level 2 in terms of steps per episode and reward, despite still having negative rewards.
* The **tuned hyperparameters** (alpha, gamma, and epsilon) are making the agent learn faster and more efficiently.