1. Key Observations & Challenges

The most critical insight from this project is the **stark difference between the agent's performance during training versus its performance on the hidden test set.**

- **Training Performance (Cell 11):** During its 5,000-episode training run, the agent showed excellent convergence. It quickly achieved a **~90-93% win rate** with an average of only **~2.2 wrong guesses** per game. Based on the training metrics, the agent appeared to be highly successful.
- **Evaluation Performance (Cell 14):** When evaluated against the 2,000-word test set, the *exact same* agent's performance collapsed. The win rate plummeted to **33.45%**, and the average wrong guesses rose to **4.51**.

This discrepancy leads to two key observations:

1. **Massive Overfitting:** The tabular Q-learning agent has clearly overfitted to the training data. The state representation defined in state_to_tuple (which includes the specific masked_word and guessed_letters) creates an astronomically large and sparse state space. The agent's Q-table (with 41,866 entries) effectively memorized the correct actions for states seen during training but failed to generalize to the new, unseen states presented by the test set.
2. **HMM Dominance:** The ablation study (Cell 17) provides the most telling insight:
    - **RL+HMM Hybrid:** 33.45% Success Rate
    - **HMM-Only Baseline:** 33.20% Success Rate
    - **Q-Learning Only:** 0.05% Success Rate

This shows that the Q-learning component provided **almost no value** during the evaluation. The agent's performance was almost entirely dependent on the HMM priors, which are blended into its action-selection policy. The Q-Learning Only agent, relying solely on its overfitted Q-table, failed completely. The primary challenge was not the HMM or the reward design, but the **infeasibility of a tabular Q-learning approach** for a state space this large.

2. Strategies: HMM & RL Design

HMM Design

The HMM was designed to be a probabilistic "oracle," and its architecture was well-suited for this task.

- **Structure:** The model correctly identifies the **position in the word** as the *hidden state* and the **letter (a-z)** as the *emission*.
- **Probabilities:**
    - **Emission Probs (_train_emissions):** These were trained as P(Letter | Position, WordLength), making them position-aware and specific to words of a certain length (e.g., 's' is more common at the end of a 5-letter word than a 3-letter word).
    - **Transition Probs (_train_transitions):** These were trained as P(Letter_i+1 | Letter_i), capturing standard English bigram frequencies (e.g., 'q' is likely followed by 'u').
- **Inference Strategy:** The "enhanced" part of this HMM is its dynamic, context-aware inference (infer_letter_probs).
    1. First, it **filters the corpus** down to only words that match the current masked pattern (e.g., _PP_E_).
    2. It then **re-computes letter frequencies** from this *filtered subset*, providing highly relevant, dynamic probabilities.
    3. Finally, it combines these dynamic probabilities with the pre-trained static emission and transition probabilities (for context from known neighbors) to produce a final, robust guess.

RL State & Reward Design

- **State (state_to_tuple):** The state was defined as a hashable tuple: (masked_word, tuple(sorted(guessed_letters)), wrong_count, available_words_count_bucketed)
    - **Why:** This captures the complete state of the game board, which is standard for tabular Q-learning. The available_words_count was a smart feature to give the agent a sense of certainty (many possible words vs. few). As observed, this state definition was ultimately the agent's downfall, as it was too granular and led to overfitting.
- **Reward (step function):** The agent used **dense reward shaping** to provide immediate feedback, which is crucial for learning.
    - **Correct Guess:** +10 (base) + 2 * new_reveals (bonus for multiple occurrences). This strongly encourages finding common letters.

- **Wrong Guess:** -30. This is a very strong penalty, correctly teaching the agent that wrong guesses are highly undesirable (costing 5 points in the final score).
- **Repeated Guess:** -5. A minor penalty for inefficiency (costing 2 points in the final score).
- **Win:** +1000. A massive terminal bonus for achieving the primary goal.

This reward structure is excellent and directly aligns with the scoring formula provided in the challenge.

3. Exploration vs. Exploitation

The agent's strategy for managing this trade-off was a sophisticated, multi-part system:

1. **Epsilon-Greedy Policy:** During training, the agent used an epsilon-greedy policy (get_best_action). With probability epsilon, it would choose a random legal action (exploration).
2. **Adaptive Epsilon Decay:** Epsilon began at 1.0 (all exploration) and decayed by a factor of 0.9985 each episode down to a minimum of 0.05 (decay_epsilon). This allowed the agent to explore broadly at the start and then increasingly exploit its knowledge as it became more confident.
3. **Hybrid Exploitation:** The "exploitation" part of the policy was not a simple argmax(Q). Instead, it blended the agent's learned Q-values with the HMM's probabilities using a hmm_weight of 0.4.
    - **Policy = 0.6 * Q_value + 0.4 * HMM_prob**
    - This is a powerful technique. It ensures the agent *always* respects the HMM's statistical model, and the Q-learning's job is to learn the "residual value"—that is, when to override the HMM's top guess for a more strategic (or ultimately, more valuable) long-term play.

4. Future Improvements (with another week)

Given the core challenge was state-space explosion, all improvements would focus on **generalization**.

1. **State Abstraction (Feature Vector):** The first and most critical change would be to abandon the (masked_word, ...) state tuple. I would replace it with a **fixed-length feature vector**. This vector would include:

- word_length
- wrong_count
- blanks_remaining
- available_words_count (from HMM filter)
- A 26-element vector for guessed letters (0 or 1).
- A 26-element vector of the **HMM probability distribution** itself.
- Features like "vowel guessed %" or "max HMM prob."

2. **Deep Q-Network (DQN):** A tabular Q-table is not feasible. By replacing the state with a feature vector, I could use a **Deep Q-Network (DQN)**. A small neural network would take this feature vector as input and output 26 Q-values (one for each letter). This would allow the agent to **generalize** from states it has seen to new, unseen states, solving the overfitting problem.
3. **Experience Replay:** To train the DQN effectively, I would implement an **experience replay buffer**. This buffer stores (state, action, reward, next_state) transitions. The network would then be trained on random mini-batches from this buffer, breaking the correlation between steps and leading to much more stable and efficient learning.
4. **HMM Enhancements:** The notebook already identifies this, but I would implement it: train **position-specific bigrams** (e.g., P(Letter_i+1 | Letter_i, Position)) instead of the current position-agnostic ones. This would make the HMM priors, which are the only component currently working on the test set, even more accurate.