TEAM ID: 2

TEAM MEMBERS:

- **CHERUKURI VENKATA KARTIK(PES2UG23CS148)**
- **G S S SURYA PRAKASH(PES2UG23CS192)**
- **DRISHTI GOLCHHA(PES2UG23CS185)**
- **FAIRLY SORATHIYA(PES2UG23CS189)**

Key Observations & Challenges

Based on the training and evaluation results, here are the main takeaways:

- **Primary Challenge: State Space Explosion.** The chosen state representation (`masked_word|guessed_letters|lives_remaining`) is extremely large. The resulting Q-table is highly sparse, meaning the agent has simply not seen most game states enough times to learn an optimal policy.
- **Low Win Rate:** This sparsity is the most likely cause for the low final win rate on the test set (**30.05%** in Cell 8). The agent makes an average of **5.34 wrong guesses** per game (out of 6), indicating it's struggling to generalize.
    - **A Clear Success:** The agent perfectly learned **not to repeat guesses** (0 repeated guesses in 2,000 test games).
    - **Robust Test Data:** The test set had **0% overlap** with the training corpus (Cell 2). This is a great practice, as it ensures the 30.05% win rate is a true measure of the agent's ability to generalize to unseen words.

Strategies: HMM, RL State, and Reward Design

HMM Design

The agent's "intuition" comes from a `HybridHangmanHMM` . This was designed as a **weighted ensemble model** .
Strategy: It combines four different probability sources, with weights tuned to prioritize context:

1. **Positional Probs (40% weight):** Guesses based on letter frequency *at a specific index* (e.g., 'E' is common at the end of a word). This model is trained for each word length.

2. **Bigram Probs (30% weight):** Provides context from the preceding letter (e.g., 'U' is likely after 'Q').
3. **Trigram Probs (20% weight):** Provides even richer context (e.g., 'E' is likely after 'TH').
4. **Global Freq. (10% weight):** A simple fallback based on overall letter frequency in the corpus.
   This hybrid approach captures multiple linguistic patterns simultaneously. It uses **Laplace smoothing** to prevent zero-probability guesses for n-grams that weren't in the training data

RL Reward Design

We used **custom reward shaping** to give the agent more granular feedback than the environment's simple win/loss rewards.

- **Winning Fast:** `+10 + (2 * lives_remaining)` for a win. This gives a higher reward for winning with more lives left.
- **Good Guesses:** `+2 + revealed_now` for a correct guess. This specifically rewards guesses that reveal *multiple* letters at once (e.g., guessing 'S' in `S_SS_`).
- **Penalties:** A small penalty for a wrong guess (`-1.5`) is distinct from a game-ending loss (`-8`). A minor "stagnation" penalty (`-0.5`) discourages wasting turns.

EXPLORATION V/S EXPLOITATION

The exploration–exploitation trade-off was managed using an ε-greedy (epsilon-greedy) strategy. At the beginning of training, the agent is encouraged to explore by choosing random actions (letter guesses) with a high probability value of espsilon. This allows the agent to gather diverse experiences and understand the environment’s dynamics without relying solely on its initial Q-value estimates.

As training progresses, epsilon is gradually decayed after each episode (for example, epsilon ← epsilon × 0.995) until it reaches a predefined minimum threshold (such as 0.1). This gradual decay ensures that the agent transitions from exploration to exploitation, where it increasingly selects the action with the highest Q-value predicted by its policy. In this phase, the agent leverages its learned knowledge to make informed, high-reward decisions.

This approach provides a balance:

Early training: High exploration allows discovery of new, potentially better strategies.

Later training: Increased exploitation enables the agent to focus on the most effective letter-guessing policy.

Additionally, the exploration process is influenced by the HMM output probabilities, ensuring that even during random exploration, the choices remain linguistically relevant. This balance

between exploration and exploitation allowed the model to improve its success rate while minimizing wrong and repeated guesses over time.

Future Improvements (If I Had Another Week)

The 30% win rate is a good start, but it can be dramatically improved. The main bottleneck is the massive state space.

1. **Dramatically More Training:** The most obvious fix. 5,000 episodes is not enough. I would increase this to **50,000 or 100,000 episodes** to give the Q-table a chance to populate.
2. **State Abstraction (Function Approximation):** The current state hash is too specific. I would replace the `defaultdict` Q-table with a **feature-based approach**.
    - **Model:** A **Deep Q-Network (DQN)** or even a simple linear model could learn weights for these features. This would allow the agent to **generalize** to states it has never seen before, which is the current model's biggest weakness.
3. **Enhance the HMM:** The HMM could be made much stronger. I would incorporate the `_get_matching_words` logic directly into the HMM's prediction. Instead of global n-grams, it would first filter the 50,000-word corpus down to *only* the words that match the current pattern (e.g., `_OOR_EED`) and *then* calculate letter frequencies. This is much more accurate.