ML- Hackathon(Team 7)

Mem1: Nidhi N | SRN :PES2UG23CS384 | Mem2: Muskan Goenka | SRN:PES2UG23CS355

Mem3: Anish Nagula | SRN: PES2UG23CS358| Mem4: Nidhi CR | SRN: PES2UG23CS382

1. Key Observations

• Primary Challenge:
State Space Explosion The most significant challenge was the design of the Reinforcement Learning (RL) agent.
The state representation (f"{masked_word}|{guessed_letters}|{lives_remaining}") is astronomically large. As a result, the agent rarely, if ever, visits the same state twice. This "state space explosion" renders the Q-table learning mechanism ineffective, as the table remains sparse. 20,000 training episodes were insufficient to build a meaningful policy.

• Insight: The HMM as the "Real" Agent The final test score (30.15% win rate, -51,459 score) is almost entirely attributable to the Hidden Markov Model (HMM) heuristic, not the RL agent. In the get_action method, when the Q-table has no entry for a given state (which is almost always), the agent defaults to choosing the letter with the highest HMM-guided probability.
The training phase essentially just confirmed that the HMM is a better than random guesser, but the RL agent itself did not learn an improved policy on top of it.

• Challenge: The Corpus vs. Test Set Gap The HMM's primary strategy filtering its internal wordlist for possible matches is highly effective only when the secret word exists in corpus.txt.
The test.txt set contains many words not in the training corpus. When no matches are found, the HMM's logic falls back to a weaker positional and bigram heuristic.
The final score, with an average of 5.27 wrong guesses per game (out of 6 lives), indicates that this fallback heuristic is not robust enough to solve unseen words.

2. Strategies

• HMM Design (The Heuristic) Models:

The HMM is structured as a dictionary of models, one foreach word length (e.g., 2 to 25).
Each length specific model contains:

1. A list of all words of that length from corpus.txt.

2. Positional probabilities (unigrams) for each letter at each index (e.g., 'E' is common in the 4th slot of a 5-letter word).

3. Bigram probabilities (e.g., the probability of 'H' appearing after 'T').

Prediction Logic:

The predict_letter_probabilities function uses a two-stage process:

1. Dictionary Matching (High Priority): It first filters its word list for all words that match the current masked_word. It then calculates letter frequencies based only on this subset of matching words. This is the strongest signal.

2. Heuristic Fallback (Low-Priority): If the dictionary match yields no results (or the list is too large, per the < 2000 limit), it falls back to a hybrid score: (positional_score + 0.1) * (bigram_score + 1.0). This fallback, which uses the bigram data, was the primary improvement over the initial baseline model.

• RL State Design Representation: The state was defined as a unique string hash: f"{masked_word}|{''.join(sorted(guessed_letters))}|{lives_remaining}".

Rationale: This representation was chosen because it is fully Markovian; it contains all information necessary to determine the next optimal action without needing any prior history.

• RL Reward Design Terminal Rewards: +100 for winning the game, -100 for losing the game(running out of lives).
Step Rewards: +5 for a correct guess that did not win the game. -10 for an incorrect guess.
Penalties: -20 for a repeated guess (an action that was already taken).
Rationale: The rewards were structured to heavily prioritize the final win, while strongly penalizing incorrect or inefficient (repeated) guesses.
The negative step reward for a wrong guess is double the positive reward for a correct one, encouraging precision.

3. **Exploration vs. Exploitation**

The exploration/exploitation trade-off was managed using an HMM-guided Epsilon Greedy strategy. Epsilon (ε): Epsilon began at 1.0 and decayed very slowly (decay rate 0.9999) over 20,000 episodes, finishing at 0.135. Exploration (High ε): When exploring (with probability epsilon), the agent did not choose a purely random letter. Instead, it used the HMM's predict_letter_probabilities distribution to make a weighted random choice. This is a form of "guided" or "smart" exploration.

Exploitation (Low ε): When exploiting (with probability 1 epsilon), the agent chose the action with the highest Q-value. As noted, because the Q-table was sparse, this "exploitation" almost always meant defaulting to the HMM's single best guess, as that was used to initialize the Q-values.

4. **Future Improvements**

If given more time, the following improvements would be prioritized:

1.Abandon Q-Learning or Redesign the State:
The current Q-learning approach is infeasible.
Option A: Abandon RL: The most practical solution is to remove the RL agent entirely and focus all efforts on optimizing the HMM heuristic.
The problem is better framed as a probabilistic search problem, not a policy-learning problem.
Option B: Feature Based State: If RL is a requirement, the state representation must be abstracted. Instead of the full string, the state could be a feature vector like (word_length, lives_remaining, num_blanks, most_common_bigram_pattern).
This dramatically smaller state space would make Q-learning (or a DQN) viable.

2.Improve the HMM Fallback Heuristic: The 30.15% win rate proves the fallback heuristic is weak.Trigram/N-gram

Models: Incorporating trigram probabilities (e.g., probability of 'G' following 'IN') would provide significantly more contextual power than bigrams alone.

Score Normalization: The current heuristic (Pos + 0.1) * (Bigram + 1.0) is arbitrary.
A better approach would be to normalize the positional probability distribution, normalize the bigram distribution, and combine them with a tuned weighted average (e.g., Score = 0.4*Pos_Score + 0.6*Bigram_Score).

3.Use a Larger Training Corpus: The corpus.txt (50,000 words) is too small.
The HMM's beststrategy (dictionary matching) fails every time the test.txt word is not in its list.
Using a standard English dictionary file (e.g., 400,000+ words) as the corpus would make this primary strategy far more robust and reduce the reliance on the weaker fallback heuristic.