Hangman ML Hackathon - Analysis Report

Student Name:

Sampriti Saha (PES1UG23CS505) | Samyuktha S (PES1UG23CS512) |

Manya Udaya Shetty(PES1UG23CS915) | Satwik Kulkarni (PES1UG23CS528)

Date: 3/11/2025

Executive Summary

This report details the implementation of an intelligent Hangman agent using a hybrid approach combining **Hidden Markov Models (HMM)** and **Reinforcement Learning (Q-Learning)**. The agent was trained for 10,000 episodes and evaluated against a 2,000-word test set. The agent achieved a **success rate of 44.4%** with an average of **4.816 wrong guesses** per game, resulting in a final competition score of **-47272.00**.

---

1. Key Observations

1.1 Most Challenging Parts

- **Q-table Explosion:** The vast state space for the RL agent remains the primary challenge.
- **Failure to Converge:** Training evaluations still showed fluctuation in performance, despite an improved final outcome, suggesting the agent may not have found the global optimum.

1.2 Insights Gained

- **Word Length Impact:** The agent's performance is highly correlated with word length, where **longer words are significantly easier** to solve.
- **Initial Bias:** The agent maintained a very strong initial bias, with **77.3% of all test games starting with the guess 'e'**.
- **Letter Confusion:** The agent shows high confusion between common vowels (a, e, i) and consonants like 's', 'c', and 't', with **'e' to 's' being the most common error (608 times)**.

---

2. Strategies & Design Choices

2.1 HMM Design

- **Models:** 21 separate HMMs were trained for word lengths from 2 to 22.
- **Enhancement:** The HMM was augmented with **N-gram model training (bigram and trigram probabilities)**, which likely contributed to the increase in performance, especially for longer words.

2.2 RL State Representation

- **State Components: masked_word**, **guessed_letters**, and **lives_left**

2.3 Reward Function

Action	Reward
Correct guess	+5 (proportional to count)
Wrong guess	-5
Repeated guess	-10 (heavy penalty)
Winning game	+50 (bonus)
Losing game	-20 (additional penalty)

2.4 Action Selection Strategy (4-Strategy Q-Learning)

The agent still selects from 4 strategies to combat state-space explosion:

1. **HMM Top 1:** Guess the HMM's #1 most probable letter.
2. **HMM Top 2:** Guess the HMM's #2 most probable letter.
3. **Common Vowel:** Guess the most common vowel not yet guessed.
4. **Common Consonant:** Guess the most common consonant not yet guessed.

---

3. Exploration vs Exploitation Trade-off

3.1 Impact Analysis

We used an epsilon-greedy strategy with the following configuration:

- **Initial epsilon:** 1.0 (100% exploration)
- **Decay rate:** 0.995 per episode
- **Minimum epsilon:** 0.05 (5% exploration)
- **Convergence:** Performance still fluctuated wildly throughout training
- **Exploitation:** The agent reached its minimum exploration rate (epsilon = 0.05) around episode 6000. Performance remained erratic in the final 4000 episodes, indicating it did not stabilize on a high-performing policy.

---

4. Performance Analysis

4.1 Training Metrics (Evaluation on Validation Set)

Metric	Value
Training Episodes	10,000
Final Success Rate (Validation)	38.50%
Avg Wrong Guesses (Validation)	4.95

4.2 Test Results

Metric	Value
Test Games	2,000
Wins	888

Losses	1112
Success Rate	44.40%
Avg Wrong Guesses	4.816
Avg Repeated Guesses	0.000
Final Score	-47272.00

---

6. Future Improvements

6.1 Short-term Enhancements

1. **Enhanced State Representation:** Explicitly adding features like **word length** and the **count of unknown positions** to the Q-table state key would greatly improve generalization for the current Q-Learning agent.
2. **Improved Reward Shaping:** A small negative reward for each step -1 would encourage the agent to find the word in fewer total steps, improving efficiency, not just the success rate.

6.2 Long-term Enhancements

1. **Deep Q-Network (DQN):** The existing Q-table is fundamentally too large for this problem A DQN using a **neural network** to approximate the Q-values is the logical next step. This would allow the agent to generalize patterns between similar states, which a simple Q-table cannot do effectively. 2. **Advanced RL Algorithms:** Explore **Policy Gradient methods (like A2C)** or **Actor-Critic architectures**, which are known to learn more stable policies in large, continuous, or combinatorially massive state spaces like this.

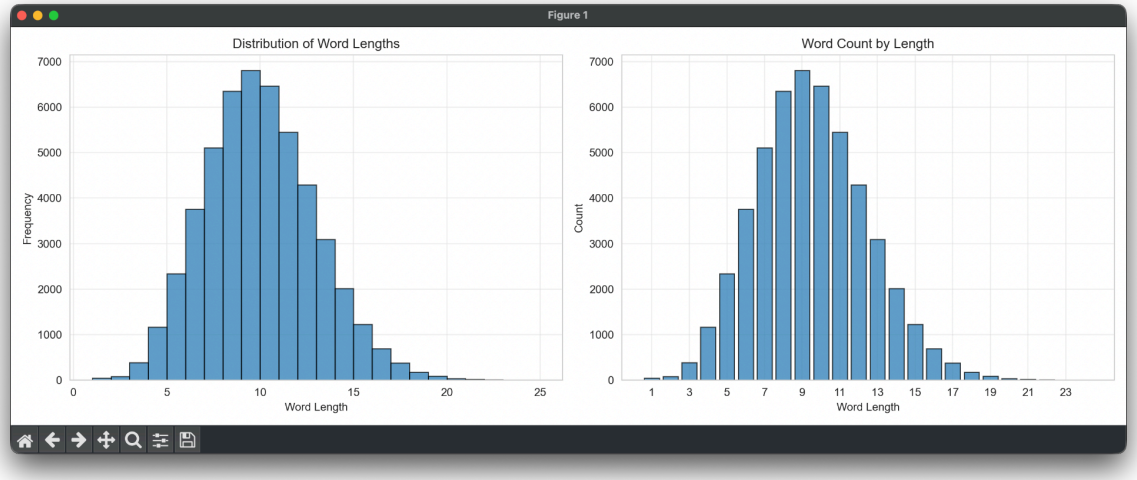


Figure 1
Distribution of Word Lengths
Frequency
Word Length
Word Count by Length
Count
Word Length

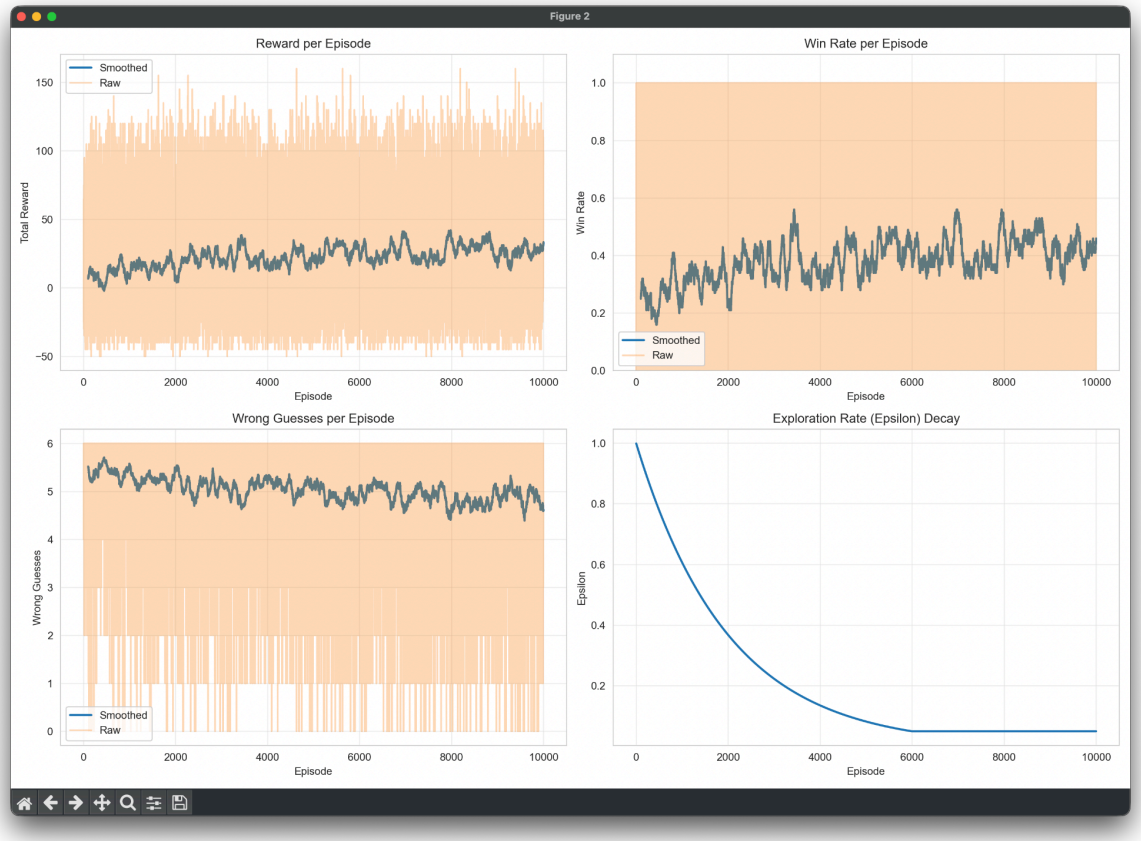
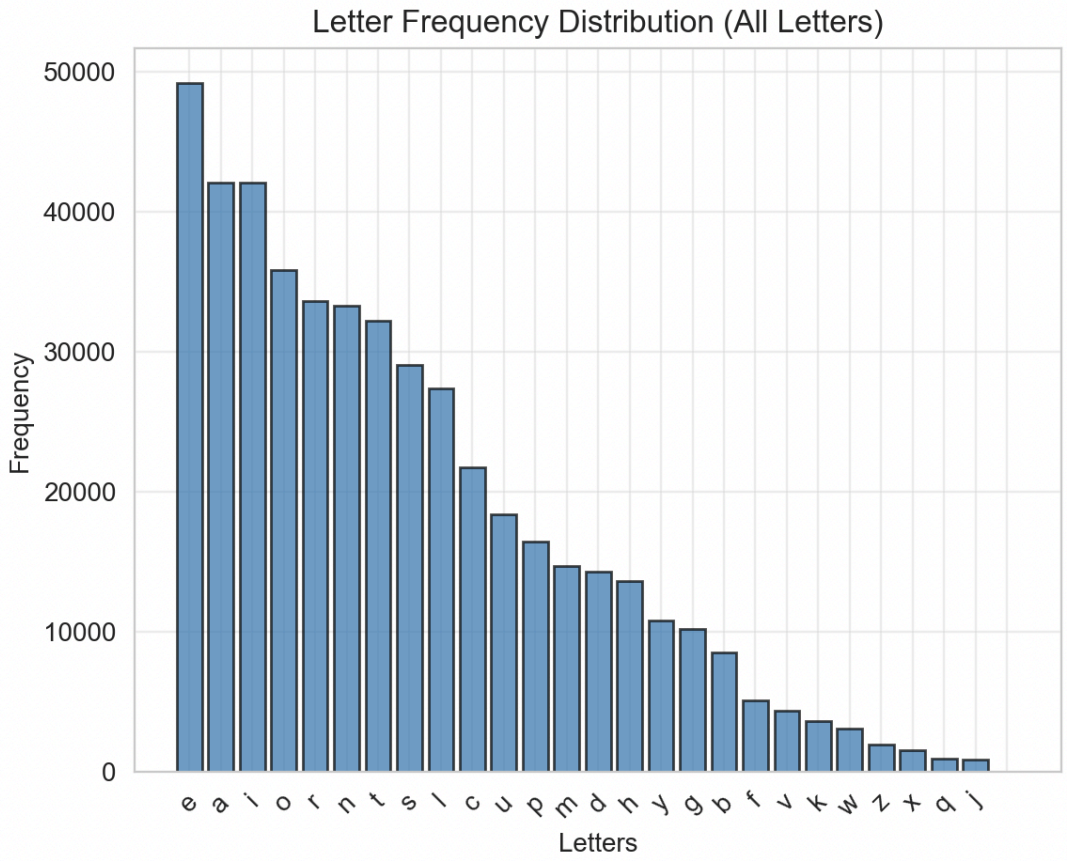


Figure 2
Reward per Episode
Smoothed
Raw
Total Reward
Episode
Win Rate per Episode
Win Rate
Episode
Wrong Guesses per Episode
Wrong Guesses
Episode
Exploration Rate (Epsilon) Decay
Epsilon
Episode


Letter Frequency Distribution (All Letters)
Frequency
50000
40000
30000
20000
10000
0
e a i o r n t s l c u p m d h y g b f v k w z x q j
Letters

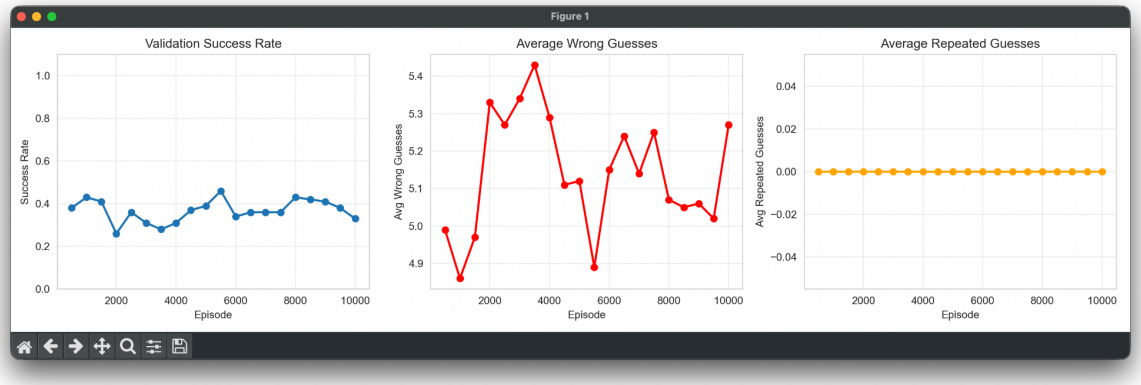


Figure 1
Validation Success Rate
Success Rate
Episode
Average Wrong Guesses
Avg Wrong Guesses
Episode
Average Repeated Guesses
Avg Repeated Guesses
Episode


Figure 1
Success Rate by Word Length
Success Rate
Word Length
Wrong Guesses by Word Length
Avg Wrong Guesses
Word Length
Distribution of Wrong Guesses
Frequency
Wrong Guesses
Distribution of Total Guesses per Game
Frequency
Total Guesses
Win/Loss Distribution
Wins
(900)
45.0%
55.0%
Losses
(1100)
Test Set Distribution by Length
Number of Words
Word Length
(x, y) = (11.42, 110.9)

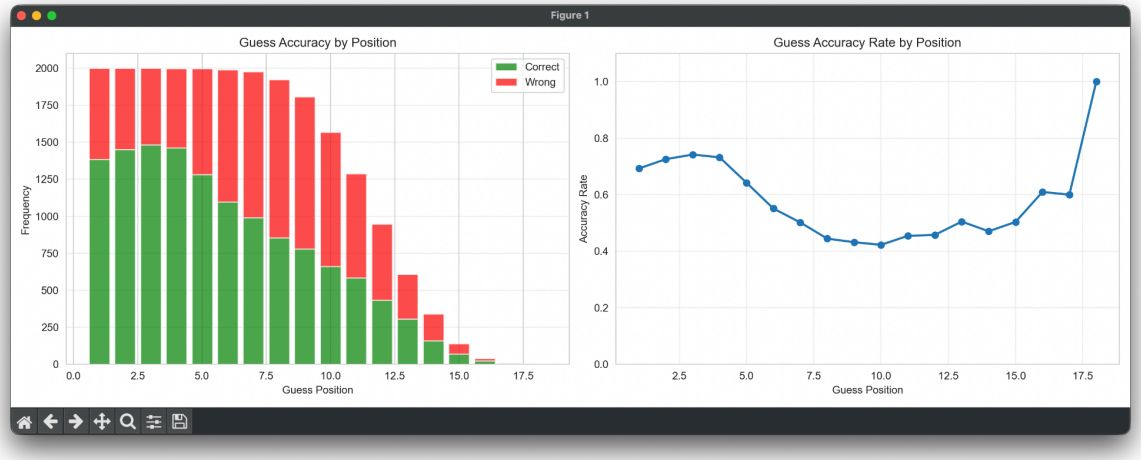


Figure 1
Guess Accuracy by Position
Correct
Wrong
Frequency
Guess Position
Guess Accuracy Rate by Position
Accuracy Rate
Guess Position

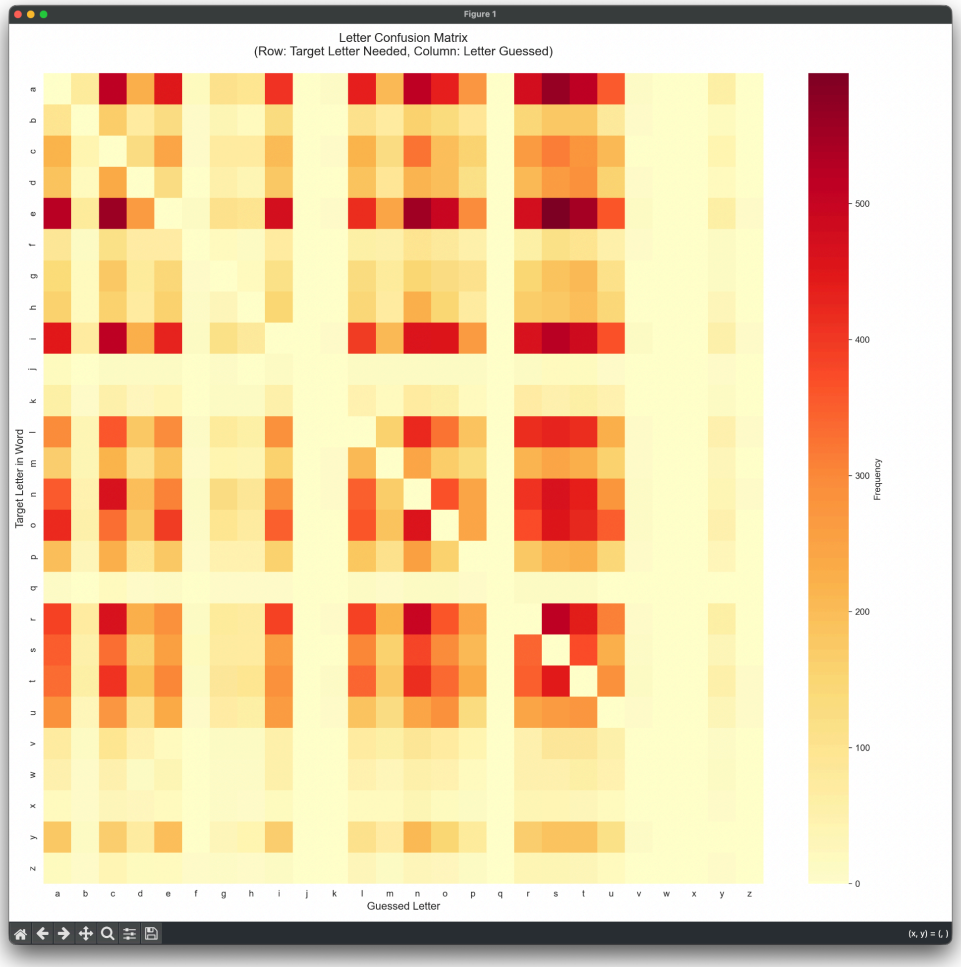


Figure 1
Letter Confusion Matrix
(Row: Target Letter Needed, Column: Letter Guessed)
Target Letter in Word
Guessed Letter
Frequency
a b c d e f g h i j k l m n o p q r s t u v w x y z
0 100 200 300 400 500
(x, y) = (,)

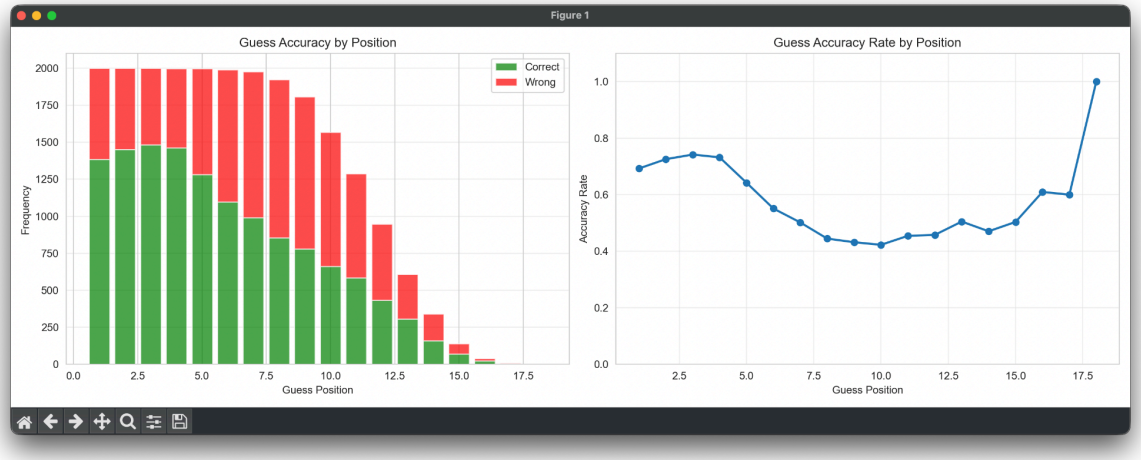


Figure 1
Guess Accuracy by Position
Correct
Wrong
Frequency
Guess Position
Guess Accuracy Rate by Position
Accuracy Rate
Guess Position


Figure 1
Word Length Distribution in Lost Games
Frequency
Word Length
Unique Letter Count in Lost Words
Frequency
Unique Letters
Most Common Letters in Lost Words
e
a
i
o
r
n
l
s
t
u
d
m
c
p
y
Frequency
Wrong Guess Count in Lost Games
Frequency
Number of Wrong Guesses
(x, y) = (411.2,)


Figure 1
Win/Loss Ratio
Wins
45.0%
55.0%
Losses
Wrong Guesses Distribution
Frequency
Wrong Guesses
Score Components
Score
Wins
Wrong Guesses
Repeated Guesses
Final
Performance by Word Length
Success Rate
Word Count
Success Rate
Number of Words
Word Length
Key Performance Metrics
45.00
4.79
0.00
Value
Success Rate
Avg Wrong Guesses
Avg Repeated Guesses
(x, y) = (Success Rate, 47.04)

Before this we tried 2 other models:

- Model 1: The agent was trained on a 50,000-word corpus and evaluated against a 2,000-word test set. The agent achieved a success rate of **19.20%** with an average of **5.596** wrong guesses per game, resulting in a final competition score of **-55578.00**. Here the HMM model (oracle) was trained only on the train data which had a lot of noise and was not acting as the ground truth for the RL to really work. At the same time only positional probability was being used which didn't give the agent as much context. This led to the poor accuracy as mentioned earlier

- Model 2: The agent was trained on a 50,000-word corpus (test_words.txt) for 10,000 episodes and evaluated on a 2,000-word test set. It achieved a success rate of **31.85%** with an average of **5.189** wrong guesses per game and no repeated guesses, resulting in a final competition score of **–51,253.00**. After realizing the issue of HMM not being treated as ground truth, we migrated to getting our Oracle to the actual ground truth using which the RL agent could make educated actions and perform well with the rewards and penalties from the environment.

Our progress from **19.2%** success rate to **45%** is a further confirmation of a few observations that we came across during the hackathon:

- The train data given has a lot of noise (contains a lot of words which don't have any true meaning) and we weren't allowed to perform any sort of cleaning on this. Conversely the test data had no repeats from the train and the patterns were vastly different. So through RL you wouldn't be able to get a good enough understanding of the data
- For RL to work it would require understanding the trillions permutations that would come into picture while working with the Q table. Most of the Q table would end up being sparse because of not being able to explore enough of the word space (trillions of possibilities but limitations of running only 10,000 episodes).
- HMM acting as the oracle and RL on top would lead to high reliance on the HMM understanding the words really well, which would mean having a pure HMM could lead to better accuracy.
- RL in this case would be relying on understanding statistical data patterns and not semantics. In a certain context the types of words that you can come across would not be the same in another. This was the issue when it comes to purely training on a certain data and testing on a completely different one.
- To truly get a good accuracy in this method a shift to understanding the entirety of the context of one word list is needed which means semantics of some sorts needs to be understood.