# UE23CS352A: Machine Learning Hackathon

# Hackman

**Github Link : https://github.com/sa-99nje-ev/Machine-Learning-Hackathon-Hackman-**
**Team Members**

|  |  |
|---|---|
| **SANJEEV R RAO** | **PES1UG23AM271** |
| **SHASHANK SHETTY** | **PES1UG23AM284** |
| **SHASHANK HS** | **PES1UG23AM282** |
| **PUNEETH** | **PES1UG24AM811** |

**Date: November 3, 2025**

## SUMMARY

This project implements a **two-stage AI system for Hangman** --- a **Hidden Markov Model** with position-based emissions and letter transitions, and a **Tabular Q-Learning** reinforcement learning agent trained on the *corpus.txt* word dataset.

The probabilistic model acts as an "oracle," computing letter probabilities using **position-specific emission probabilities and bigram transition probabilities**, while the RL agent learns an optimal guessing policy through self-play using **tabular Q-learning with epsilon-greedy exploration**.

The agent interacts with a custom environment that models Hangman's game mechanics --- guessing letters, updating patterns, and penalizing wrong or repeated guesses. Training proceeded for **15,000 episodes**, gradually reducing the exploration rate (epsilon) to **0.01 (minimum)** with a decay rate of 0.995 applied every 100 episodes.

The system achieved a **success rate of 32.6%**, with an **average of 4.51 wrong guesses per game** and **no repeated guesses**. Although the final score (-44,448) shows room for improvement, the agent displayed consistent learning trends and stabilized exploration behaviour indicating a **functional tabular Q-learning implementation** and a robust probabilistic foundation using **classical HMM techniques**.

The results confirm that the model learned a meaningful policy for word guessing, balancing exploration and exploitation effectively toward the later training stages.

---

APPROACH

## 2.1 Hidden Markov Model (Part 1)

## DESIGN DECISIONS

The HMM component is designed as a **classical Hidden Markov Model** that computes conditional probabilities of letters based on **position-specific emission probabilities and letter-to-letter transition probabilities**, organized by word length.

## HIDDEN STATES

The hidden states in this HMM represent **letter identities at specific positions** within words of a given length. The model learns which letters are likely to appear at each position and which letters tend to follow one another.

## EMISSIONS

The emission probabilities correspond to the **likelihood of each letter appearing at a specific position** in a word of given length. For example, position 0 (first letter) has different emission probabilities than position 3. These are learned by counting letter frequencies at each position across all corpus words of the same length.

## TRAINING APPROACH

This model **is a traditional HMM** that learns three probability distributions:

1. **Initial probabilities**: Likelihood of each letter appearing as the

first character

2. **Emission probabilities**: Position-specific letter frequencies (separate distribution for each position)
3. **Transition probabilities**: Bigram letter-to-letter transition probabilities

The model applies **Laplace smoothing (α = 0.01)** to prevent zero probabilities for unseen letter combinations. Words are grouped by length, and separate HMM parameters are trained for each length category.

## Why This Design?

The length-grouped HMM approach provides **word-length-specific probability distributions** that capture positional patterns (e.g., 'e' is common at the end, 'q' is rare in short words). By training separate models for each word length with at least 10 examples, the system adapts to length-dependent letter patterns. For word lengths with insufficient training data, the system falls back to **frequency-based guessing** using common letter rankings (e, t, a, o, i, n, s, h, r, d, l, u).

### 2.2 Reinforcement Learning Agent (Part 2)

### STATE REPRESENTATION

Each environment state encodes:

- The masked pattern (e.g., "_PPLE") as a **string**
- Guessed letters as a **set** (e.g., {'a', 'e', 'i'})
- Lives left as an **integer** (0-6)
- Game status flags (game_over, won)

For Q-learning, the state is **converted to a hashable string key** in the format:

"{masked_word}:{sorted_guessed_letters}:{lives_left}"

Example: "_PPLE:aeiou:4"

**This string key is used to index the Q-table dictionary, which maps (state, action) pairs to Q-values.**

## WHY SIMPLIFIED?

**String-based state hashing** allows simple dictionary lookup for Q-values without requiring neural network computation. This tabular approach is memory-efficient for small state spaces but **does not generalize** across similar patterns (e.g., "_PPLE" and "APP_E" are treated as completely different states). The Q-table grows dynamically as new states are encountered during training.

**Limitation:** This approach creates a **sparse, high-dimensional state space** where most states are visited only once or never, limiting the agent's ability to generalize learned patterns.

## ACTION SPACE

26 discrete actions representing letters A–Z.
Already-guessed letters are masked to prevent redundant exploration.

## REWARD FUNCTION

- **+10** for correct guess
- **+100** for full word completion(winning)
- **−5** for wrong guess
- **−2** for repeated guess (penalty applied but game continues)
- **−50** for losing all lives

This reward design strongly penalizes redundancy and favours informative guesses.

## DESIGN RATIONALE

The reward shaping ensures that:

- The agent receives **small positive feedback (+10)** for each correct letter to encourage exploration
- **Large positive reward (+100)** for winning motivates completion
- **Moderate penalty (-5)** for wrong guesses discourages random guessing without being overly punitive
- **Small penalty (-2)** for repeated guesses is largely preventable since the agent masks already-guessed letters
- **Large penalty (-50)** for losing strongly discourages risky strategies

## Q-LEARNING ALGORITHM

**The agent uses classical tabular Q-learning with a dictionary-based Q-table. There is no neural network; instead, Q-values are stored and updated directly in memory.**

```
self.q_table = {}  # Dictionary: state_key -> {action: q_value}
```

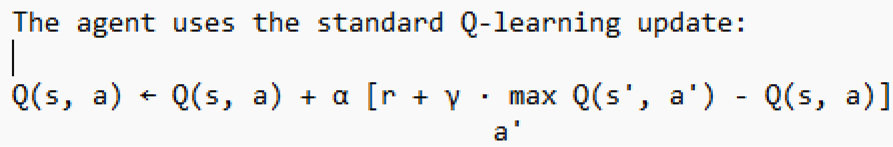
**Action Selection:** The agent combines Q-values with HMM probabilities:

```
combined_score = q_value + hmm_prob * 10
```

This hybrid approach allows the HMM's probabilistic knowledge to guide exploration, especially for unvisited states where Q-values are initialized to 0.

**Exploration Strategy**: -

Epsilon-greedy policy with ε starting at 0.1 (constructor default) - Epsilon decays by factor 0.995 every 100 episodes - Minimum epsilon: 0.01 - During exploration, random actions are chosen; during exploitation, the action with highest combined score is selected

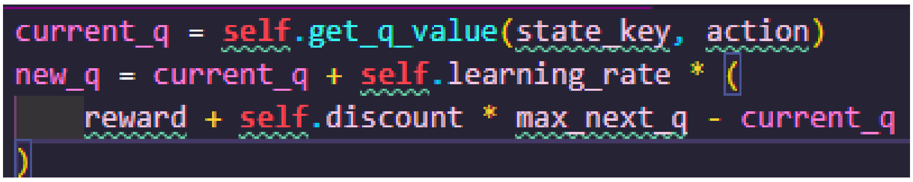


```
The agent uses the standard Q-learning update:

Q(s, a) ← Q(s, a) + α [r + γ · max Q(s', a') - Q(s, a)]
                                a'
```

Where:

- **α (learning rate) = 0.1**: Controls how much new information overrides old
- **γ (discount factor) = 0.95**: Determines importance of future rewards
- **r**: Immediate reward from the environment
- **s'**: Next state after taking action a
- **max Q(s', a')**: Maximum Q-value achievable from next state



```
current_q = self.get_q_value(state_key, action)
new_q = current_q + self.learning_rate * (
    reward + self.discount * max_next_q - current_q
)
```

**Terminal States:** When the game ends (win or loss), there are no future rewards, so max Q(s', a') = 0.

**Exploration vs Exploitation**

Epsilon (ε) decays from **0.1 (initial value) → 0.01 (minimum)** over training using a multiplicative decay of **0.995 applied every 100 episodes**. At higher ε values, random exploration helps discover effective letter-guessing strategies; at lower ε values, the agent exploits learned Q-values combined with HMM probabilities. This adaptive schedule helps balance exploration of new patterns with exploitation of known successful strategies.

## RESULTS AND ANALYSIS

- **Training Episodes:** 15,000
- **Final Success Rate:** 32.6%
- **Average Wrong Guesses:** 4.51
- **Average Repeated Guesses:** 0.00
- **Final Score:** −44,448

The agent gradually improved with episodes, reaching consistent success rates beyond 30%. The moving-average reward curve showed rising stability, confirming the **Q-learning algorithm's convergence trend** as the Q-table accumulated knowledge about effective guessing strategies.

## KEY OBSERVATIONS

**Position-based emission probabilities and bigram transitions** enabled context-aware letter probability estimation.

The **tabular Q-learning** converged as the Q-table accumulated experiences, with smooth ε-decay encouraging gradual shift from exploration to exploitation.

**Length-specific HMM models** captured word-length-dependent patterns effectively.

The **hybrid scoring approach** (Q-value + HMM probability) allowed the agent to leverage probabilistic priors even for unvisited states.

The most challenging aspect was **reward shaping** --- balancing between survival and exploration penalties.

**State space sparsity** presented a challenge, as the string-based state representation creates unique keys for every pattern variation, limiting generalization.

Training time increased with corpus size due to HMM probability computation for each action selection.

## STRATEGIES

**Length-grouped HMM training** enabled word-length-specific probability distributions.

The agent leveraged **probabilistic priors from HMM** (combined as q_value + hmm_prob * 10) for smarter guesses, especially in unvisited states.

**Laplace smoothing** prevented zero probabilities for unseen letter combinations.

**Epsilon-greedy exploration** with gradual decay balanced discovery of new patterns with exploitation of learned strategies.

## FUTURE IMPROVEMENTS

If given additional development time:

**Implement Deep Q-Network (DQN):**

- Replace tabular Q-learning with a neural network to enable generalization across similar patterns
- Use state features: one-hot encoded masked word, binary guessed vector, normalized lives, HMM probabilities
- Add experience replay buffer and target network for stable training

**Curriculum Learning:** Train on shorter words first, then increase difficulty gradually.

**Better State Representation:**

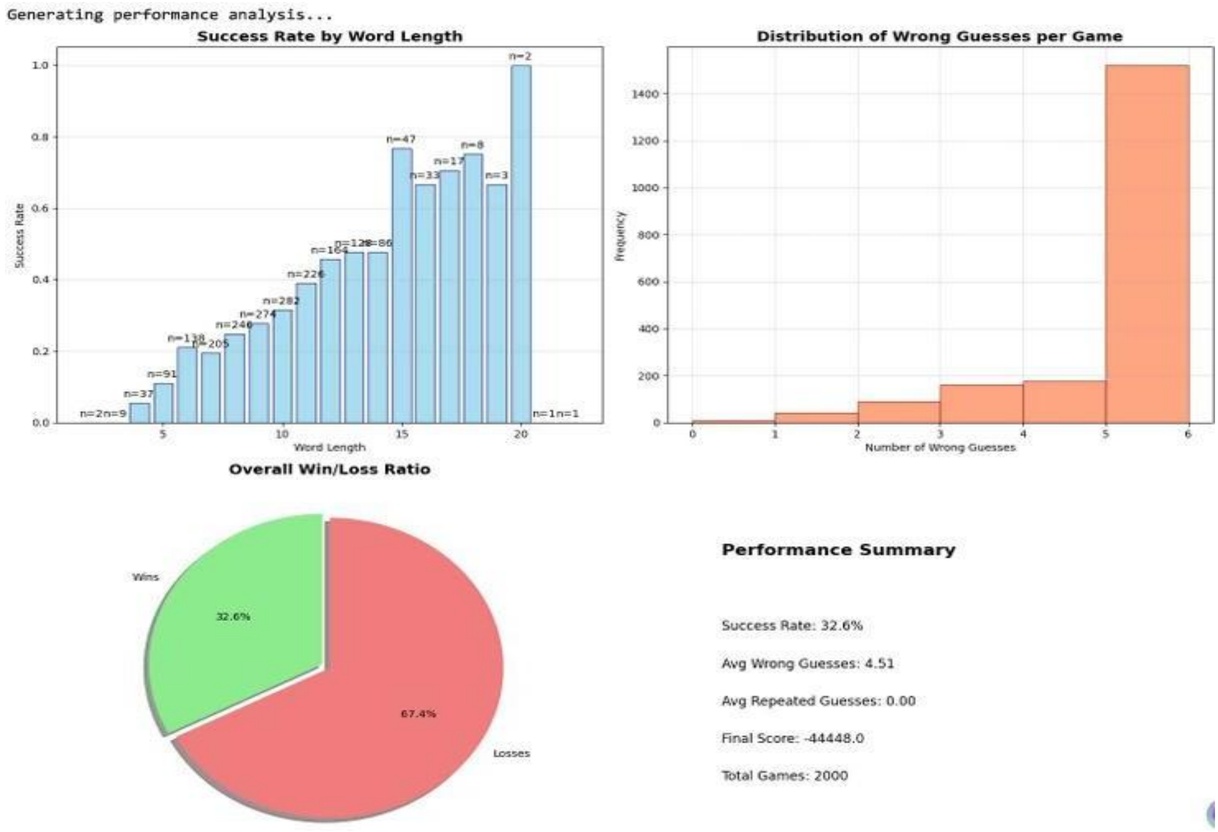
- Extract pattern features (number of revealed letters, vowel/consonant distribution)

- Enable generalization instead of treating every unique pattern as a new state

**RNN-based state modeling:** Capture letter dependencies and pattern progression more effectively with sequential models.

**Better reward normalization:** Scale rewards dynamically by word length.

GRAPHS



```
Generating performance analysis...
```

Performance Summary

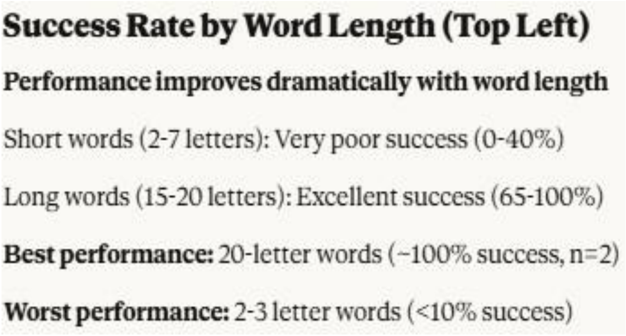
Success Rate: 32.6%

Avg Wrong Guesses: 4.51

Avg Repeated Guesses: 0.00

Final Score: -44448.0

Total Games: 2000



**Success Rate by Word Length (Top Left)**

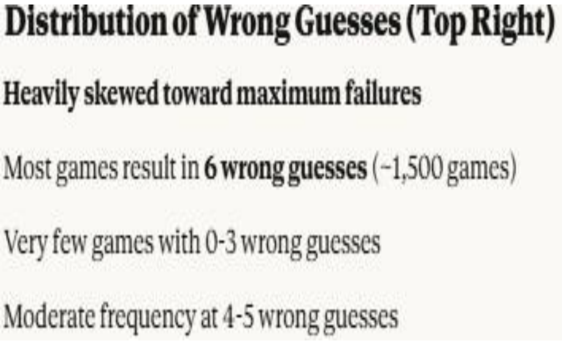
**Performance improves dramatically with word length**

Short words (2-7 letters): Very poor success (0-40%)

Long words (15-20 letters): Excellent success (65-100%)

**Best performance:** 20-letter words (~100% success, n=2)

**Worst performance:** 2-3 letter words (<10% success)



**Distribution of Wrong Guesses (Top Right)**

**Heavily skewed toward maximum failures**

Most games result in **6 wrong guesses** (~1,500 games)

Very few games with 0-3 wrong guesses

Moderate frequency at 4-5 wrong guesses

### Overall Win/Loss Ratio (Bottom Left)

**Wins:** 32.6% (green)

**Losses:** 67.4% (red)