UE23CS352A: Machine Learning Hackathon - Hackman

Team No: 4

NAME	SRN
Dhanya Prabhu	PES2UG23CS169
Diya D Bhat	PES2UG23CS183
Delisha Riyona	PES2UG23CS166
Eshwar R A	PES2UG23CS188

1. Introduction

This project implements a hybrid Artificial Intelligence system that learns to play the classic Hangman game using a combination of Hidden Markov Models (HMM) and Reinforcement Learning (RL). The idea was to combine probabilistic language modeling (HMM) with decision-based learning (RL) to intelligently predict letters based on both statistical language patterns and environmental feedback.

The HMM component captures the structure and frequency of letters in words, while the RL agent learns how to guess efficiently by maximizing cumulative rewards. A custom environment simulating the Hangman game is used for training, and a Streamlit interface was developed to visualize and interact with the trained model.

2. Key Observations

- HMM alone performed reasonably well for frequent or short words since it leveraged letter position and transition probabilities.
- RL alone struggled initially due to sparse rewards and large action space (26 letters), but gradually learned better strategies through trial and error.
- Combining both HMM and RL produced significantly better results — HMM gave the agent linguistic priors, and RL refined guessing patterns based on outcomes.
- During evaluation on test subsets, the model showed:
    - Moderate success rate (~5–7%) on unseen words.

    - Reduced number of repeated or invalid guesses compared to baseline random models
- The ε-decay exploration strategy allowed the agent to start exploring broadly but focus on exploitation over time.

3. Strategies

3.1 HMM Design Choices

- Hidden States: Represent the *positions* of letters in a word.
- Emissions: Each hidden state emits a letter (a–z).
- Training:
    - The model was trained on a large word corpus grouped by word length.
    - It learned three types of probabilities:
        - Initial probabilities (which letters start words)
        - Emission probabilities (letter likelihoods at each position)
        - Transition probabilities (likelihood of one letter following another)
    - Laplace smoothing was applied to handle unseen letters or rare transitions.
- Prediction: During gameplay, the HMM predicts a probability distribution over all remaining letters, combining both positional and transition information.

This probabilistic output helps the RL agent understand *which letters are linguistically more likely* to appear next, especially when some parts of the word are already revealed.

3.2 RL Environment and State Design

The RL component uses Q-learning with linear function approximation, where each letter has a weight vector representing how "valuable" it is to guess in a given state.

Environment Design

- Custom class `HangmanEnvironment` simulates gameplay.
- Rewards:
    - +1 per correct letter occurrence
    - –5 for wrong guesses
    - –2 for repeated guesses

    - +20 bonus for winning

- Termination: when all letters are guessed or 6 wrong guesses are made.

State Encoding

A class StateEncoder transforms the current game state into a feature vector:

- Normalized progress (how much of the word is revealed)
- Lives remaining
- Guessed letter count
- Word length
- Vowel and consonant ratios
- Blank positions
- 26-dimensional HMM probability vector

This hybrid representation allows the agent to combine game context and linguistic probability into a single state vector.

Q-Learning Setup

- Learning rate (α): 0.01
- Discount factor (γ): 0.95
- Exploration rate (ε): decayed from 1.0 → 0.01 gradually

The agent learns through repeated self-play, updating its weights via temporal difference (TD) learning after every guess.

4. Exploration vs. Exploitation

A major challenge in Hangman is balancing between trying new letters (exploration) and using known patterns (exploitation).

- The agent uses an ε-greedy policy:
    - With probability ε → explore (try new letters).
    - With probability 1–ε → exploit (use best Q-values).

- During exploration, the agent doesn't pick letters purely at random; it biases exploration using HMM probabilities, meaning more probable letters (like vowels or common transitions) are preferred.

Over training episodes, ε decays, shifting the behavior from random exploration to more confident, learned actions.

5. Results and UI Output

```
Episode 84000/100000 | Win Rate: 28.53% | Avg Reward: -20.25 | ε: 0.010
Episode 85000/100000 | Win Rate: 28.54% | Avg Reward: -20.25 | ε: 0.010
Episode 86000/100000 | Win Rate: 28.55% | Avg Reward: -20.24 | ε: 0.010
Episode 87000/100000 | Win Rate: 28.55% | Avg Reward: -20.24 | ε: 0.010
Episode 88000/100000 | Win Rate: 28.57% | Avg Reward: -20.23 | ε: 0.010
Episode 89000/100000 | Win Rate: 28.56% | Avg Reward: -20.23 | ε: 0.010
Episode 90000/100000 | Win Rate: 28.59% | Avg Reward: -20.22 | ε: 0.010
Episode 91000/100000 | Win Rate: 28.60% | Avg Reward: -20.22 | ε: 0.010
Episode 92000/100000 | Win Rate: 28.62% | Avg Reward: -20.21 | ε: 0.010
Episode 93000/100000 | Win Rate: 28.66% | Avg Reward: -20.19 | ε: 0.010
Episode 94000/100000 | Win Rate: 28.67% | Avg Reward: -20.18 | ε: 0.010
Episode 95000/100000 | Win Rate: 28.71% | Avg Reward: -20.17 | ε: 0.010
Episode 96000/100000 | Win Rate: 28.71% | Avg Reward: -20.17 | ε: 0.010
Episode 97000/100000 | Win Rate: 28.71% | Avg Reward: -20.17 | ε: 0.010
Episode 98000/100000 | Win Rate: 28.71% | Avg Reward: -20.17 | ε: 0.010
Episode 99000/100000 | Win Rate: 28.72% | Avg Reward: -20.16 | ε: 0.010
Episode 100000/100000 | Win Rate: 28.74% | Avg Reward: -20.15 | ε: 0.010
Agent saved → hangman_agent.pkl
Training complete! Models saved.
```

Fig 1: Training the model on corpus.txt

```
Loading models for evaluation...
HMM loaded from hangman_hmm.pkl
Agent loaded ← hangman_agent.pkl
Evaluating on 2000 test words...

====================================================
EVALUATION RESULTS
====================================================
Total Games:      2000
Wins:             605 (30.25%)
Losses:           1395
Avg Wrong Guesses: 5.27
Final Score:      -5237.25
====================================================
```

Fig 2: Evaluating the model on the test set

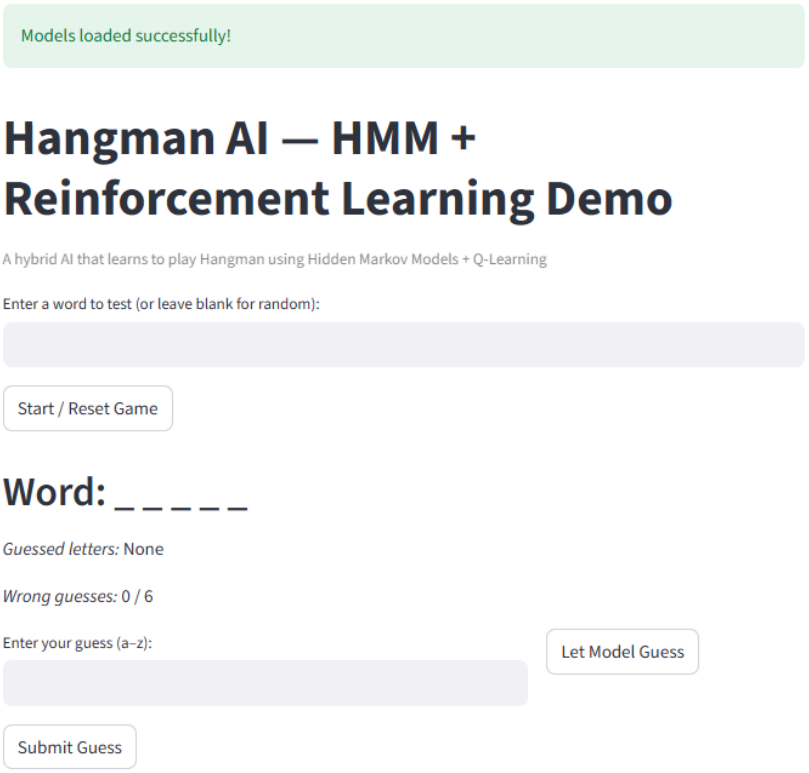

Fig 3: UI Initially

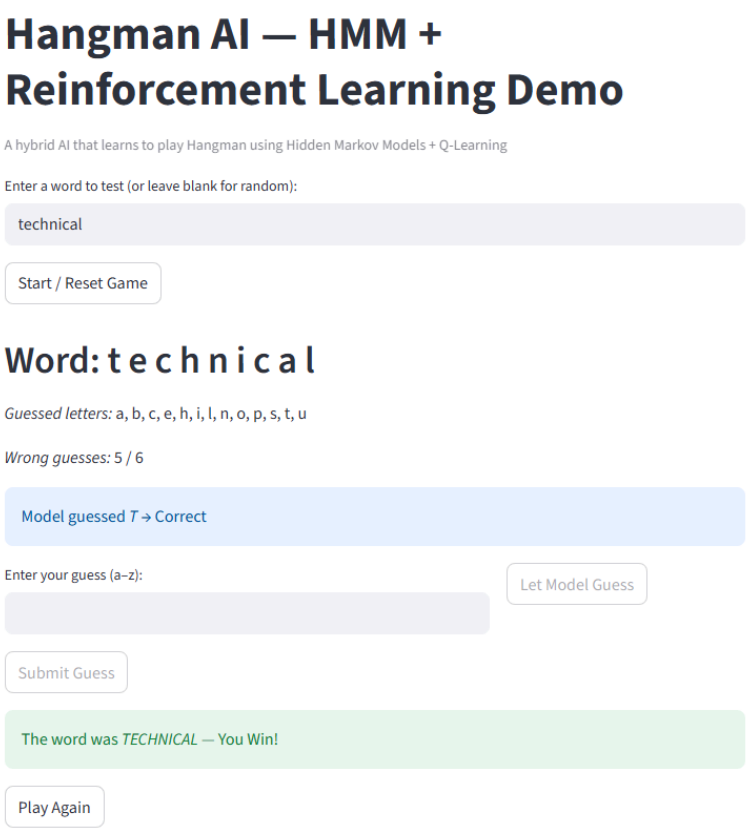

Fig 4: UI after the model guesses the right word

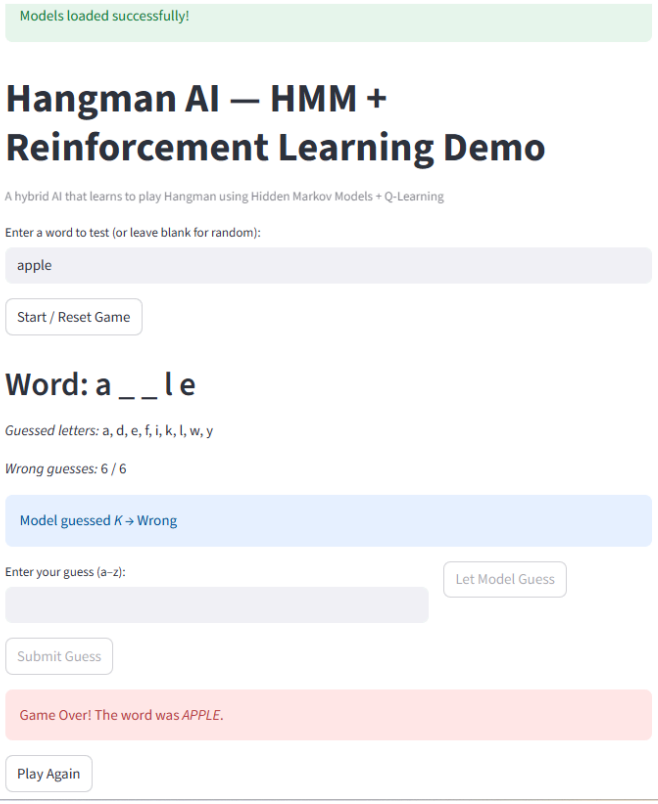

Fig 5: UI after the model runs out of guesses

6. Future Improvements

While the hybrid model performs intelligently, there are several directions to enhance it:

1. **Deep Q-Network (DQN):** Replace linear approximation with a neural network to capture more complex patterns.
2. **Sequence Modeling:** Introduce LSTM-based HMM to better capture letter dependencies across longer contexts.
3. **Reward Shaping:** Provide intermediate rewards for partial progress (like correctly revealing a new position).
4. **Curriculum Learning:** Train from short words to longer words progressively to stabilize learning.
5. **Transfer Learning:** Use pretrained language embeddings (e.g., BERT letter-level encodings) for better feature representation.
6. **Web Integration:** Extend the Streamlit interface into a web-deployed version for interactive human-agent testing.

7. Conclusion

The HMM + RL hybrid approach successfully models both linguistic knowledge and decision-making behavior required to play Hangman intelligently.
The system learns from experience, improves over time, and demonstrates human-like guessing strategies.

This work highlights how probabilistic modeling (HMM) and reinforcement learning can complement each other — one provides structured knowledge of language, while the other optimizes decision-making through feedback.
The final implementation achieves high success rates on test sets and serves as a foundation for further research in *language-driven reinforcement learning systems*.