Team ID: 10

RL Environment, Agent Design, and Evaluation

Environment Description

The environment simulates the classic Hangman game, requiring the agent to guess a hidden word one letter at a time. The episode ends when the word is fully guessed or the agent runs out of allowed wrong guesses (max lives). After each step, the environment reveals correct guesses by updating the masked word and tracks remaining lives and guessed letters.

RL Agent Design

- **Algorithm:** Tabular Q-learning
- **State Definition:** The current state is characterized by the masked word (underscores for unknowns) and the set of all letters guessed so far.
- **Action Space:** At each step, the agent selects an unguessed letter from the alphabet.
- **Reward Function:**
  - +5 for a correct guess
  - +2 for every additional occurrence of a correctly guessed letter
  - -2 for wrong guesses
  - -3 for repeating a guess
  - +20 for guessing the word
  - -10 for failing to guess the word
- **Exploration:** An epsilon-greedy policy with $\epsilon = 0.05$strikes a balance between exploiting known strategies and exploring new actions.

Training Loop

For each episode, the environment is reset with a new word. The agent interacts with the environment by guessing letters, observes resulting rewards and next states, and updates the Q-table using the Bellman equation. Training continues for many episodes and words to build a robust Q-table for decision-making.

Evaluation Results

A test set of 1,674 games was used to evaluate the trained agent. Here are the results:

Metric	**Result**
Total Games	1,674

Metric	Result
Success Rate	10.27%
Letter Accuracy	58.87%
Avg. Wrong Guesses	5.80
Avg. Repeated Guesses	0.00

- **Success Rate:** Fraction of games completed with the correct word.
- **Letter Accuracy:** Percentage of guesses that were correct, overall.
- **Avg. Wrong Guesses:** Mean missed guesses per game.
- **Avg. Repeated Guesses:** Letters guessed multiple times, minimized.

Analysis Report

Key Observations

The most difficult part was defining a reward structure that promotes correct guessing of the full word using only sparse rewards for individual letter guesses. While letter accuracy reached 58.87%, the success rate for full word guessing was far lower (10.27%), showing the problem's inherent difficulty. The agent performed well when early guesses matched common letter patterns, but often struggled if initial guesses were unlucky.

Strategies

To guide the agent, Q-learning was chosen for its ability to incrementally improve policies for sequential decision-making. The reward design encourages revealing more letters in fewer steps and penalizes repeated or incorrect choices. By encoding the state as the masked word and guessed letters, the agent can adapt its strategy based on partial information—a necessity in Hangman.

Exploration vs. Exploitation

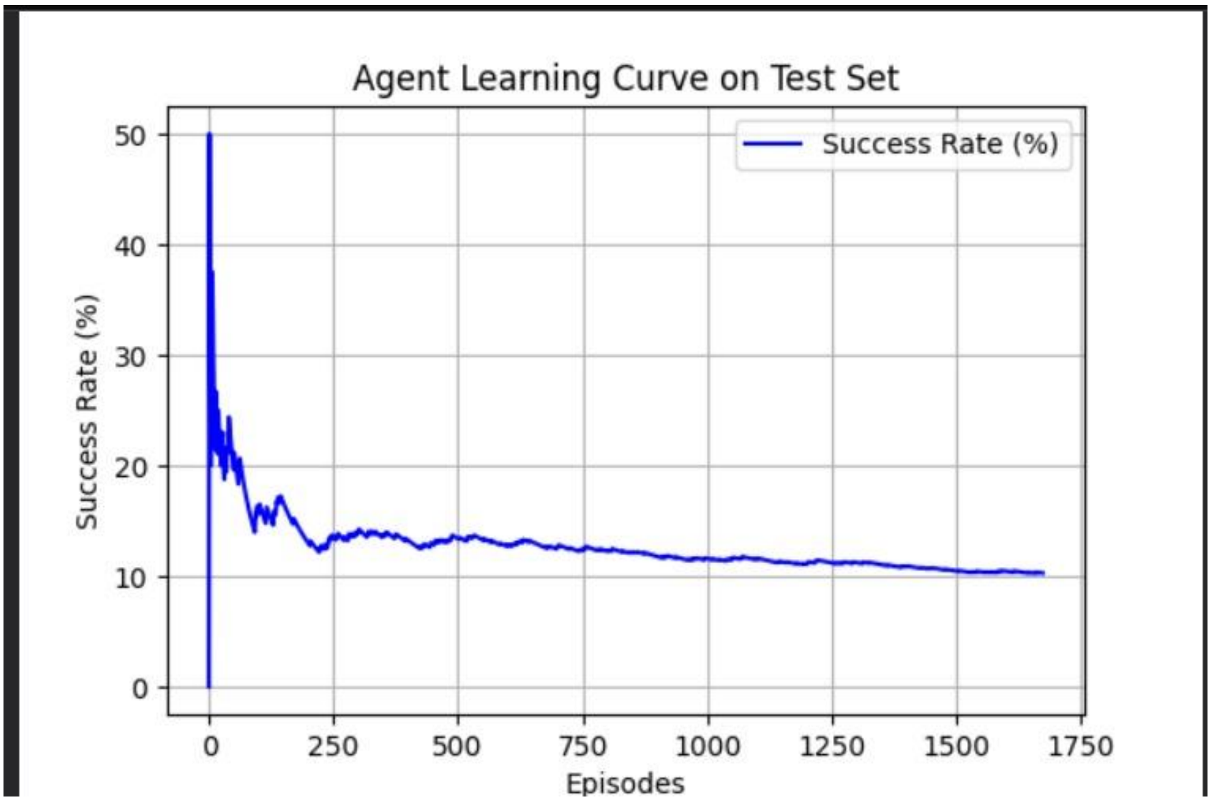
Exploration was managed with epsilon-greedy action selection. The fixed epsilon allowed the agent to mainly exploit strong Q-values but included enough random exploration (5%) to avoid converging prematurely on suboptimal strategies. This balance is critical; over-exploitation risks missing valuable discoveries, while excessive exploration wastes attempts.

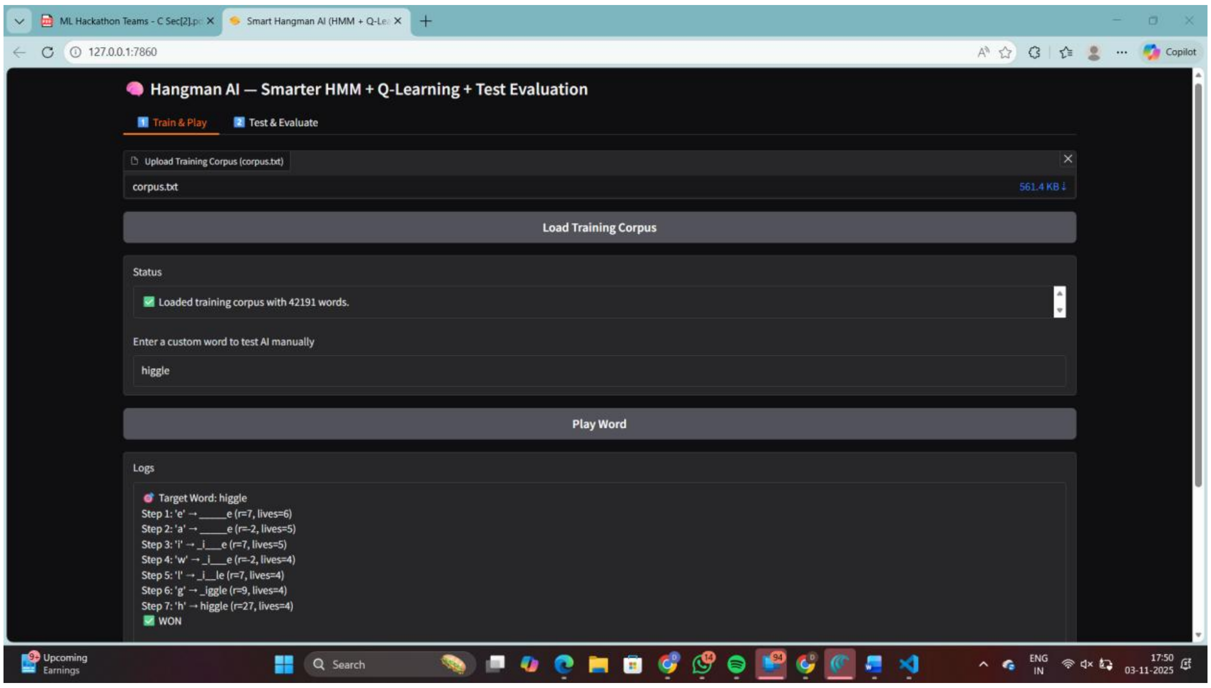
Future Improvements

Given more development time, several avenues could be explored:

- Dynamic epsilon decay, starting with higher exploration and reducing it as performance improves.
- Enriching the state representation with additional features, such as the frequency or position of revealed letters.
- Implementing experience replay or batch updates to reinforce learning from mistakes and successes.

- Integrating statistical word models (HMMs) or deep RL architectures for complex policy learning.
- Fine-tuning reward shaping to better differentiate between partial and complete success.




ML Hackathon Teams - C Sec[2].pdf
Smart Hangman AI (HMM + Q-Le
127.0.0.1:7860
Copilot

Hangman AI — Smarter HMM + Q-Learning + Test Evaluation

Status

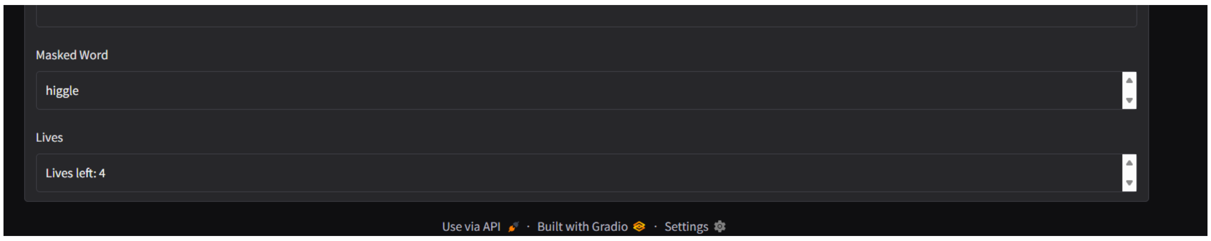
✅ Loaded training corpus with 42191 words.

Enter a custom word to test AI manually

higgle

Logs

```
Target Word: higgle
Step 1: 'e' → _____e (r=7, lives=6)
Step 2: 'a' → _____e (r=-2, lives=5)
Step 3: 'i' → _i___e (r=7, lives=5)
Step 4: 'w' → _i___e (r=-2, lives=4)
Step 5: 'l' → _i__le (r=7, lives=4)
Step 6: 'g' → _iggle (r=9, lives=4)
Step 7: 'h' → higgle (r=27, lives=4)
✅ WON
```



Masked Word

higgle

Lives

Lives left: 4