Analysis Report: Hangman AI Agent

Name	SRN
B Teja Deep Sai Krishna	PES2UG23CS135
B Koushik	PES2UG23CS133
Gagan S S	PES2UG23CS192
Darshan N	PES2UG24CS808

Section: 5C

Analysis Report: Hangman AI Agent

1. Strategies: HMM Design, RL State, and Reward

A. Hidden Markov Model (HMM) Design Choices

The HMM's purpose is to act as a **probabilistic oracle** for letter prediction. The design focused on capturing local and positional linguistic features while ensuring **generalization** and **pickling safety**.

HMM Component	Implementation Detail	Rationale
HMM Structure [cite: 20]	A single, non-linear HMM structure [cite: 20] is trained to capture multiple features based on **word length** and **position**.	Training separate HMMs for 50,000 words would be too complex and memory-intensive. A feature-rich single model with length-specific sub-models is more efficient[cite: 49].
Hidden States [cite: 21]	**Implicitly defined by word length and relative position:** 'start' (first 2 chars), 'middle', and 'end' (last 2 chars).	This captures the bias of letters (e.g., 'S' at the end, 'Q' at the start) without creating an unmanageable number of explicit states.
Emissions [cite: 22]	The **letter** being observed at a given position/context.	Directly models the probability of a letter appearing.
Probabilities	Positional/Relative Position Probs, Length-Specific Letter Freq, Bigram/Trigram Probs, Vowel/Consonant Pattern Scores.	**Ensembling of multiple features** counters sparse data and improves prediction robustness over a single HMM chain.
Generalization	**Laplace Smoothing ($\alpha=0.1$):** Applied aggressively to all bigram, trigram, and positional	

	probabilities. This prevents zero probabilities and improves the model's performance on unseen contexts, mitigating overfitting to the training corpus.	

HMM Linguistic Analysis (based on execution output): The analysis shows the agent's core linguistic knowledge:

- **1st Top Global Letter: a** (8 occurrences)
- **3rd Top Global Letter: e** (5 occurrences) *(Note: These figures are based on the dummy corpus created in the notebook's main execution block, which contained simple words like 'apple', 'banana', 'orange', etc.)*

B. RL Agent State, Action, and Reward Design

The RL agent's function is to use the HMM's probabilities to choose the **optimal action**. A Q-Learning agent was implemented.

Component	Implementation Detail	Rationale
State ($\mathbf{S}$) [cite: 29]	A tuple: **(Abstracted Masked Word Pattern, Remaining Lives, # Guessed Letters)**.	State abstraction is **critical** to manage the massive state space of possible masked words[5]. The pattern abstracts long blank runs (B+) and single blanks between known letters (X). Remaining lives and the number of guesses provide key decision-making context.
Action ($\mathbf{A}$) [cite: 31]	Guessing any single letter from the set of **available, un-guessed letters**.	This adheres to the problem requirement[cite: 31].
Reward ($\mathbf{R}$) [cite: 32]	*Win:* $\mathbf{+50}$ *and* $\mathbf{+1}$ per correct letter occurrence[cite: 33]. *Lose:* $\mathbf{-50}$[cite: 34]. *Wrong Guess:* $\mathbf{-5}$[cite: 34]. *Repeated Guess:* $\mathbf{-2}$[cite: 34].	The reward function is heavily skewed to **maximize wins** (± 50) while providing a smaller penalty for wrong guesses (-5) and an even smaller one for inefficiency (-2). This balances the immediate loss of a life/efficiency with the ultimate goal of winning, in line with the final scoring formula[cite: 43].

2. Key Observations and Overfitting Mitigation

Most Challenging Parts:

1. **State Space Explosion:** The combination of masked words, guessed letters, and remaining lives creates an immense state space, making simple Q-learning impossible. The solution was the **Abstracted Masked Word Pattern** (e.g., aB+e for a__e), which aggressively limits the number of unique states the Q-table must learn.
2. **Overfitting (The HMM-RL Conflict):** The RL agent tends to overfit the small-batch training data, relying purely on memorized Q-values, which leads to poor generalization on the validation set.

Overfitting Mitigation Strategies (Tuning for Generalization)

Strategy	Implementation Detail (Tuned Hyperparameter)	Rationale
HMM Influence	**Combined Score: $0.2 * Q_{value} + 0.8 * HMM_{Prob} * 10$** (in select_action)	The HMM's general linguistic knowledge is weighted **four times higher** than the RL's context-specific Q-value. This forces the agent to rely primarily on the robust, smoothed HMM probabilities, using the Q-value only for minor, state-specific adjustments.
Q-Learning Regularization	**Regularization factor: 0.05** (in update_q_value)	A regularization term ($\lambda * Q_{current}$) was added to the Q-value update. Increasing this factor aggressively **penalizes high Q-values** (memorization), making the agent more sensitive to the HMM's influence and preventing Q-value inflation.
Early Stopping [cite: 63]	**Patience Limit = 3:** Training stops if the validation win rate does not improve for 3 consecutive evaluation intervals.	Directly combats overfitting by halting the training process when performance on unseen data begins to decline, preventing memorization of the training set.

3. Exploration vs. Exploitation Trade-off

The agent uses an epsilon-greedy strategy, which was refined to leverage the HMM model during exploration.

Strategy	Implementation Detail	Rationale
Exploitation	**Combined Score (0.2 * Q-value + 0.8 * HMM-Prob)** is used to select the best action.	The agent exploits its knowledge by picking the action with the highest *combined* score, balancing learned utility (Q-value) with linguistic probability (HMM).
Exploration (ϵ-greedy)	**$\epsilon_{start}=0.3$, $\epsilon_{decay}=0.9995$** (Tuned for slower decay).	Starts with a **30% chance** of exploration to quickly gather diverse data. The slow decay ensures exploration continues late into the training, preventing premature convergence to suboptimal policies.
HMM-Informed Exploration	When exploring ($\mathbf{random.random() < \epsilon}$), the action is chosen based on a **weighted random sample from the HMM's probabilities**, instead of a uniform random choice.	This makes the exploration process **semi-intelligent**. Even random guesses are informed by the linguistic likelihood provided by the HMM, drastically increasing the quality of exploration and improving sample efficiency.

4. Future Improvements (Given one more week)

1. **Deep Q-Network (DQN) Implementation:** The current Q-learning struggles with state abstraction. Replacing the Q-table with a **DQN** would allow a more complex state representation (e.g., one-hot vector of the masked word and HMM probabilities) without requiring aggressive manual state abstraction, fundamentally solving the state space problem and allowing the agent to learn finer distinctions between similar game states.
2. **Context-Aware HMM Emission:** Enhance the HMM to include an "unseen" token for letters not yet guessed. This would more accurately model the probability of a letter appearing in a *blank* spot, conditioning the probability on the known letters in the word.
3. **Dynamic Reward Scaling:** Implement a reward system where the penalty for a wrong guess increases as the number of remaining lives decreases. This would guide the agent to be **more cautious** when close

to losing the game, aligning its strategy with the final score's heavy penalty for total wrong guesses.