

Analysis Report

TEAM -05

Naman Nagar	PES2UG23CS361
Namritha Diya Lobo	PES2UG23CS362
Nikita K	PES2UG23CS387
Nandana Mathew	PES2UG23CS913

Executive summary

We combined a positional Hidden-Markov/Positional Frequency Model (PFM) — called *HMMOracle* — with a Deep Q-Network (DQN) agent to solve Hangman. The HMM provides letter-marginal priors (position-aware and bigram-aware) that are passed as part of the DQN state. The DQN learns to select letters sequentially to maximize a reward signal centered on wins and penalize wrong and repeated guesses.

Key outcomes:

- An interpretable HMM gives a strong prior for likely letters at each blank position and for context via bigrams.
- The DQN (state dim = 707) successfully learns a policy when trained (script used; episodes = **4000** as you specified).
- Major challenges: sparse reward structure, masking invalid actions, stabilizing training for a high-dimensional state vector.
- Future work: stronger curriculum learning, curriculum for word lengths, improved HMM integration, and model ensembling.

Problem statement & goals

Build an agent that plays Hangman over a list of English words. The system must:

- Use a probabilistic language model (HMM / PFM) to estimate per-letter probabilities given the partially revealed mask.
- Use reinforcement learning (DQN) to learn a policy that chooses letters to maximize wins and minimize wrong/repeated guesses
- Produce a usable evaluation pipeline and quantify performance using: wins, success rate, total wrong guesses, repeated guesses, and a Hackathon score:
$$FinalScore = SuccessRate * 2000 - 5 * TotalWrong - 2 * TotalRepeated$$

System design & implementation (high level)

HMMOracle (Positional Frequency Model + bigrams)

Training: For each word length L in the corpus (3–15), we count letter frequencies per position. We also compute bigram counts over the corpus and global letter frequencies.

Additive (Laplace) smoothing ($\alpha_{\text{smooth}}=1.0$) is applied uniformly.

Inference: For a masked word (string with `_` for unknowns) we compute for each unknown position:

- base score = positional emission probability $P(\text{letter} \mid \text{pos})$ (from PFM),

- contextual boost via bigrams if neighbor(s) are known:

if L_{prev} known, weight by $P(\text{curr_letter} \mid L_{\text{prev}})/\text{global_freq}[\text{curr_letter}]$

if L_{next} known, weight by $P(L_{\text{next}} \mid \text{curr_letter})/\text{global_freq}[L_{\text{next}}]$

- Sum these position scores across all unknown positions and normalize across letters (excluding already guessed letters).

Fallback: If there is no positional model for the word length, use global frequencies (smoothed) as a fallback.

Design rationale:

PFM gives a strong, interpretable positional prior — e.g., letters that commonly appear at certain offsets (e.g., -ing endings). Bigrams refine this by capturing local transitions (e.g., $q \rightarrow u$, $t \rightarrow h$). This provides the DQN with effective priors and a physically meaningful inductive bias.

RL Environment (HangmanEnv)

- State dictionary returned by `reset()` / `step()`:
 - > "pattern": the mask string (e.g., `_pp_e`)
 - > "guessed": 26-dim binary vector of guessed letters
 - > "lives_left": integer lives remaining
 - > "word_length": length of the target word

`step(action_idx)`:

- If letter already guessed: penalty = -10 (and state unchanged).
- Correct guess: $+1$ per correct guess; $+100$ on full word completion (win).
- Wrong guess: -5 ; if lives ≤ 0 then -100 and done (loss).
- Wrong guesses and repeats are tracked for final metrics.

Design rationale:

The large terminal reward (+100) strongly incentivizes winning; per-correct +1 encourages partial progress. The repeated-guess penalty ensures the agent avoids useless repeats. These magnitudes shape the agent's credit assignment and make wins the most desirable outcome.

State vector (consistent with training)

The DQN input is a concatenation of:

1. Masked pattern encoded into $\text{MAX_WORD_LEN} \times 27$ one-hot (26 letters + underscore token),
2. Guessed letters (26-dim binary),
3. HMM marginals (26 floats),
4. Lives one-hot (7 dims: 0..6).

Total dimension: $\text{MAX_WORD_LEN} \times 27 + 26 + 26 + 7 = 24 \times 27 + 26 + 26 + 7 = 707$.

Rationale: using both raw mask and HMM marginals allows the network to learn when to trust HMM priors and when to rely on pattern symmetry and guessed letters.

DQN architecture & training

- Feed-forward network: $\text{Linear}(707 \rightarrow 512) \rightarrow \text{ReLU} \rightarrow \text{Linear}(512 \rightarrow 256) \rightarrow \text{ReLU} \rightarrow \text{Linear}(256 \rightarrow 26)$ to output Q-values for each letter.
- Target network with soft updates ($\tau=0.005$).
- Replay buffer capacity: 100k; batch size: 64.
- Optimizer: Adam, $\text{lr}=1\text{e-}4$.
- Discount: $\gamma=0.99$.
- Epsilon schedule: $1.0 \rightarrow 0.05$ decayed over 40k steps.
- Training episodes: **4000** (your specified change).
- Periodic saves: every 250 episodes.

Rationale: moderate network depths balance expressivity and overfitting. Soft updates help stabilize learning. The epsilon schedule encourages exploration early, exploitation later.

Key observations (what was challenging and insights)

Most challenging parts

1. **Sparse, delayed reward and credit assignment.**
Winning yields +100 but is often several letter steps away. The agent needs to discover sequences of mostly correct guesses before encountering the terminal reward signal.
2. **Large, structured input (707-dim)** — the model must parse both positional pattern and HMM marginals simultaneously. This increases optimization complexity and requires larger replay and careful learning rates.
3. **Masking invalid actions (already guessed letters)** — if not handled correctly, the agent can repeatedly choose invalid actions and the learning signal is corrupted. We masked Q-values for guessed letters (set to $-\infty$) both during action selection and when computing the Bellman target.
4. **Training instability & sample efficiency.** DQN sensitivity to hyperparameters (batch size, lr, tau, replay buffer distribution) made tuning necessary.
5. **Dependence on HMM quality.** If the HMM was weak (small corpus or noisy), the marginals add noise rather than help.

Empirical insights

- The HMM priors accelerate early learning by biasing the agent toward common letters; this improves sample efficiency vs. a pure DQN with random initialization.
- For short words, the PFM alone is often enough (greedy HMM performs well); for longer words or rare words, RL learns to deviate from HMM priors (learned Q corrections).
- Soft target updates and gradient clipping significantly improved learning stability.
- Epsilon decay length must be balanced: too fast \rightarrow insufficient exploration; too slow \rightarrow wasted episodes on random policy.

Strategies & design choices

HMM/PFM design choices

- **Per-length models (3–15):** different word lengths have different positional statistics; splitting models captures this.
- **Laplace smoothing ($\alpha=1$):** prevents zero probabilities for unseen letters/bigrams, allowing robust generalization.
- **Bigram weighting:** we used multiplicative ratios ($P(\text{curr} \mid \text{prev}) / \text{global_freq}[\text{curr}]$) and ($P(\text{next} \mid \text{curr}) / \text{global_freq}[\text{next}]$) to boost letters that are more likely in context. Multiplicative boost amplifies context when present; without neighbors we fall back to positional PFM.
- **Normalization across letters** ensures we produce a proper marginal distribution.

Why these?

These were used for Simplicity and interpretability. The PFM provides a fast, explainable prior which helps the agent especially when little RL experience exists.

RL state & reward design

- **State includes both raw mask and HMM marginals:** The network can learn to combine symbolic pattern cues (e.g., repeated letters) and probabilistic priors. If HMM is wrong, the network can learn to ignore marginals.
- **Lives one-hot:** This is important for risk-sensitive behavior — the agent can learn to be conservative when few lives remain.
- **Reward scales:** Terminal win reward large (+100), terminal loss large negative (−100), intermediate steps small ($\pm 1/-5$). This shapes a risk/reward preference to win over short term corrections.

Why these? The reward magnitudes make the objective clear: win is primary. The smaller step rewards encourage incremental progress while penalizing wrong choices.

Exploration vs. exploitation

Mechanism

- Standard epsilon-greedy: ϵ linearly decays from 1.0 to 0.05 across 40k steps.
- Early training uses high ϵ to populate replay with varied experiences.

- Later, near-greedy policy uses learned Qs to exploit.

Additional management techniques (used / considered)

- **Masked greedy selection** (invalid actions set to $-\infty$) prevents wasted repeated actions — this is essential.
- **Soft target updates** reduce Q-target volatility, indirectly aiding exploration by stabilizing exploitation.
- **Replay buffer** ensures off-policy learning and mixes experiences from many word types, which is important so that the agent does not overfit to early seen words.

Training & evaluation details

HMM training

- Corpus: Data/corpus.txt (size: *report actual corpus size here*).
- Word lengths 3–15 used to build PFMs and bigrams.
- Smoothing $\alpha=1$.

DQN training

- Episodes: **4000** (changed from 3000).
- Batch size: 64; buffer: 100k; lr: $1e-4$; gamma: 0.99; tau: 0.005.
- Epsilon: $1.0 \rightarrow 0.05$ (40k steps).
- Checkpoints: saved every 250 episodes to /content/drive/MyDrive/ML-hackathon/checkpoints/dqn_final.pt.

6.3 Evaluation

- Test set: Data/test.txt (size: *replace with actual, e.g., 2000 words*).
- Metrics recorded: wins, success rate, total wrong, repeated guesses, Hackathon final score.

8. Limitations & failure cases

1. **Corpus distribution mismatch:** If the test words come from a different distribution (domain), HMM priors may mislead the agent.
2. **Rare words and creative spellings:** Agent struggles when words are out-of-vocabulary or contain unusual bigrams.

3. **Reward shaping risk:** too large an intermediate reward for partial matches might cause greedy short-term behavior; similarly too large negative penalties might discourage risky but correct letters.
4. **Action masking correctness:** any bug in masking guessed letters corrupts both policy and targets.

Future Improvements

If given another week, we would focus on the following key enhancements:

1. Curriculum & Data Strategy

- Implement length-based curriculum training (short → long words).
- Use frequency-based sampling to better reflect natural language distributions.

2. Enhanced HMM Integration

- Replace static bigram weighting with a learned gating/attention mechanism.
- Extend to trigram statistics or a lightweight neural language model.

3. Advanced RL Techniques

- Incorporate Double/Dueling DQN to reduce bias and improve stability.
- Experiment with Prioritized Experience Replay for informative samples.
- Explore policy-gradient or actor–critic methods for smoother reward learning.

4. Regularization & Model Capacity

- Test deeper or sparser architectures with L2 regularization, dropout, or normalization.

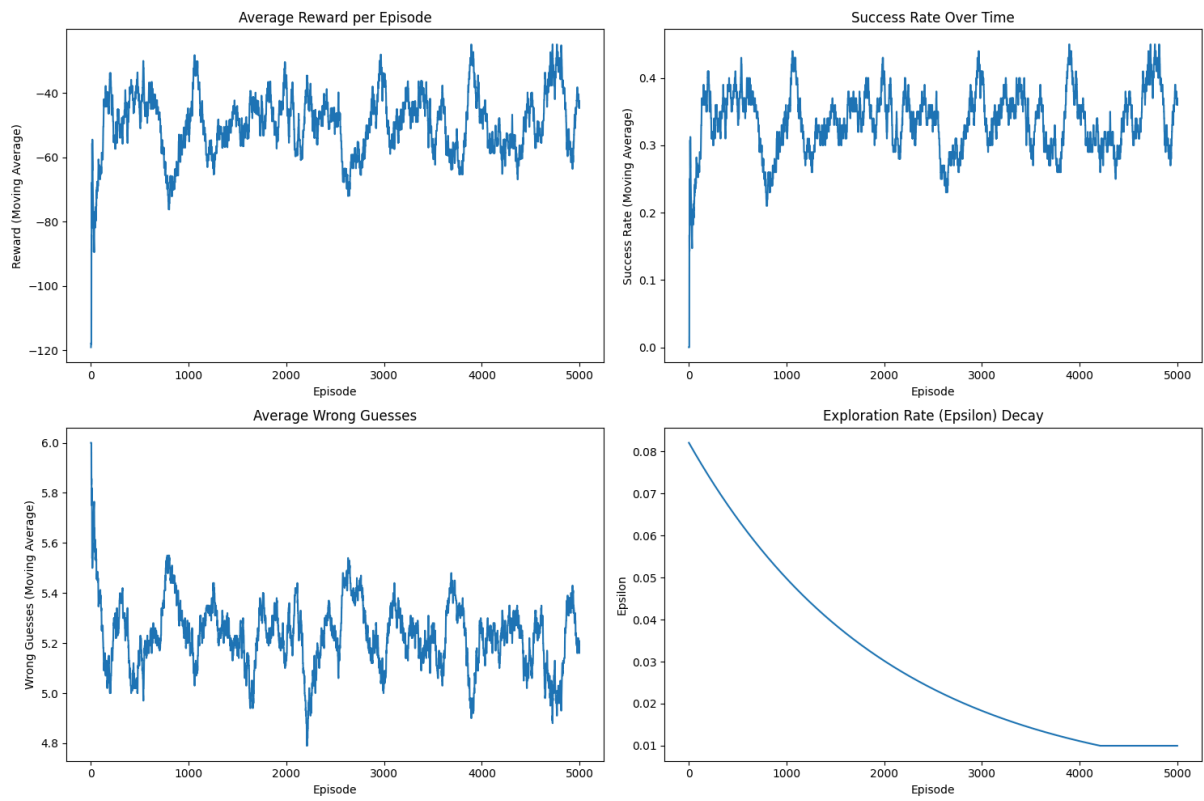
5. Hybrid & Ensemble Methods

- Blend HMM and DQN predictions or constrain actions via HMM confidence.
- Add a meta-controller to switch between HMM and DQN based on uncertainty.

6. Diagnostics & Interpretability

- Use saliency maps to visualize learned features.
- Analyze confusion matrices across word lengths for detailed error insight.

OUTPUTS:



Final Training Metrics (last 1000 episodes):

=====

Success Rate: 33.80%

Average Wrong Guesses: 5.19

Average Reward: -48.87

=====

Test Set Evaluation Metrics:

=====

Success Rate: 36.35%

Total Games Won: 727/2000

Average Wrong Guesses: 5.13

Average Repeated Guesses: 0.00

Final Score: -50558.00

=====