Analysis Report: Hangman Solver Project

Team Members

Name	SRN
Mohammed Musharraf	PES2UG23CS915
Mohammed Shehzaad Khan	PES2U23CS349
Mohammed Bilal	PES2UG23CS344
Mohammed Aahil	PES2UG23CS342

Summary

We developed a Hangman solver that couples Hidden Markov Models with Deep Reinforcement Learning. After much trial and error, our final system achieved:

- **94.40% success rate** on 2000 test games
- Average of **2.13 wrong guesses** per game
- **Zero repeated guesses**

---

1. Key Observations

What Was Really Challenging?

Pure RL Performance Was Disappointing

- After 5000 episodes of training (19 minutes on GPU), our RL agent won only 7% of games
- The 619-dimensional state space made learning really difficult
- Agent got lost in all that information
- Sparse rewards meant mostly negative feedback, making it hard to learn what works

The Breakthrough: Word Filtering

- We added a word filtering system that matches current pattern with corpus words
- When 20 or fewer words match, we use letter frequency from those words directly

- This simple approach boosted success rate from 7% to 94.40%
- **Lesson learned**: Sometimes straightforward rule-based approaches beat fancy machine learning, especially with good domain knowledge

What We Learned

Hybrid Approaches Work Much Better

- Our hybrid system outperformed pure RL by 13x
- Optimal weighting we found:
    - 50% word filtering
    - 30% HMM predictions
    - 20% RL Q-values

Position Matters in English Words

- HMM learned certain letters appear more often at certain positions
- Example: Pattern "A___E" in 5-letter words commonly has I, O, R, U, L in middle
- Used Laplace smoothing to handle unseen scenarios
- Performance peaked on medium words (10-15 letters with 95%+ win rate)

Exploration Strategy Was Too Conservative

- Started with random exploration, gradually decreased over 700 episodes
- In retrospect, should have reduced randomness faster
- Agent needed to start using what it learned sooner

---

2. Our Strategy and Design Choices

HMM Design

Architecture Decisions

- **Hidden States**: Each position in word (0, 1, 2, ..., length-1)
- **Emissions**: Letters A-Z
- Built 24 separate HMMs (one for each word length from 1-24)

Why This Design?

- English has strong positional patterns:
    - Q almost always followed by U
    - Words often end in E
    - Starting letters usually consonants (S, C, P)

- Treating each position separately lets model learn these patterns
- Short and long words have different structures (3-letter vs 15-letter behave differently)

Training Process

- Counted how often each letter appears at each position
- Converted counts to probabilities
- Added smoothing value (1.0) so unseen patterns get tiny non-zero probability
- This prevents model from being overly confident about impossible scenarios

Reinforcement Learning Design

State Representation (619 dimensions total)

Our RL state encoded five pieces of information:

1. **Masked word** (540 dims)
    - 20 positions with one-hot encoding
    - 27 possibilities per position (26 letters + underscore)
    - Padded shorter words
2. **Guessed letters** (26 dims)
    - Binary values for each letter
3. **Lives remaining** (1 dim)
    - Normalized between 0 and 1
4. **HMM probabilities** (26 dims)
    - Probability distribution from HMM
5. **Word filter probabilities** (26 dims)
    - Distribution from matching corpus words

In hindsight, 619 dimensions was probably overkill and made learning harder.

Reward Function

We designed rewards to encourage good behavior and punish bad behavior:

- **Correct guess**: +10 points per position revealed
    - Vowels appearing multiple times give more reward
    - +100 bonus for winning the game

- **Wrong guess**: -15 points
    - -100 penalty for losing the game
- **Repeated guess**: -20 points
    - Heavy penalty because it wastes a turn

Action Selection (Hybrid Agent)

Our final agent makes context-based decisions:

- If 20 or fewer matching words: Pick most common letter from those words
- Otherwise: Combine information with weights:
    - 50% word filtering
    - 30% HMM predictions
    - 20% neural network Q-values

---

3. Balancing Exploration and Exploitation

Epsilon-Greedy Approach

Our Strategy

- Started with epsilon = 1.0 (100% random exploration)
- Multiplied epsilon by 0.995 after each episode
- Reached minimum of 0.01 around episode 700
- **Problem**: Way too slow - agent spent too long making random guesses

Making Exploration Smarter

HMM-Guided Exploration

- 70% of exploratory guesses followed HMM probabilities
    - Explored common letters (E, A, I, O) more often
    - Avoided wasting time on rare letters (X, Z, Q)
- 30% remained completely random for true exploration
- This improved sample efficiency significantly

Stabilizing Learning

Experience Replay

- Stored past experiences in buffer (last 10,000 transitions)
- Randomly sampled during training
- Helped break up patterns and stabilize learning

Target Network Updates

- Updated target network every 10 episodes
- Prevented learning targets from moving around too much
- Made training more stable

Why Training Performance Stayed Low

Despite all techniques, training win rate stayed around 7% because:

- Huge state space (619 dimensions requires exponentially more samples)
- Sparse rewards (mostly negative feedback)
- Training on random words but testing on corpus words
- Distribution mismatch between training and testing

The Interesting Part: This low training performance did not actually matter much. The word filtering component was doing most of the heavy lifting. The RL agent was more like a backup system for edge cases rather than the main decision maker.

---

4. What We Would Do Differently

If We Had Another Week

1. Simplify State Representation

- Use LSTM networks to compress masked word: 540 dims → 64 dims
- Keep only top-5 most probable letters from HMM and word filter
- Total reduction: 619 dims → ~100 dims
- Benefits: Faster training, better generalization

2. Improve Reward Function

- Add information gain reward (how much did guess reduce uncertainty?)
- Give bonus points for revealing higher percentage of word
- Extra credit for guessing common vowels early when most useful

3. Implement Curriculum Learning

- Phase 1: Easy words (4-6 letters)

- Phase 2: Medium words (7-10 letters)
- Phase 3: All lengths
- Agent learns basic patterns before tackling complicated cases

4. Try Prioritized Experience Replay

- Sample experiences where agent was most surprised (high TD error)
- Learn more from mistakes and unusual situations
- Faster convergence on critical game states

If We Had a Few More Weeks

1. Attention Mechanism

- Replace one-hot encoding with learned embeddings
- Network learns which positions to focus on
- Automatically learns patterns like words ending in "E" or "ING"

2. Ensemble Approach

- Combine multiple agents:
    - Pure HMM agent
    - Pure word filtering agent
    - DQN agent
    - N-gram model
- Different agents vote on next letter
- Each agent might excel at different word types

3. Trie Data Structure for Word Filtering

- Current approach: Linear search through words (slow)
- Trie approach: Much faster lookups (100x speedup)
- Critical for real-time game play

If We Had a Month or More

1. Transformer Model

- Train small transformer from scratch on corpus
- Use character-level tokens
- Better at capturing language patterns than HMM
- Might predict likely letters more accurately

2. Meta-Learning

- Train agent to learn how to learn
- Quickly adapt when seeing unfamiliar word patterns
- Better handling of rare word types
- Few-shot learning capabilities

3. Explainability Tools

- Visualize what attention mechanism focuses on
- Decompose Q-values to understand decision contributions
- Extract human-readable rules from trained network
- Currently neural network is a black box

---

5. Final Thoughts

Results Summary

Metric	Training	Testing
Win Rate	7.18%	**94.40%**
Avg Wrong Guesses	-	2.13
Repeated Guesses	-	0

The huge difference between training (7%) and testing (94%) shows the real power was not in reinforcement learning, but in the combination of techniques, especially word filtering.

Main Takeaways

1. Combination Beats Individual Techniques

- Word filtering provided most value
- HMM added useful linguistic knowledge
- RL agent helped with edge cases
- Together they achieved 94%, while RL alone was stuck at 7%

2. Simple Solutions Sometimes Beat Complex Ones

- When we have good domain knowledge (like a word list), use it directly
- Filtering is more effective than learning everything from scratch
- Neural networks are not always the answer

3. RL as Support, Not Center

- RL plays a supportive role in our system
- Handles unusual situations and fine-tunes decisions
- Not meant to learn entire strategy
- Makes sense for problems with structured knowledge available

Comparing Different Approaches

If we had built different systems:

- **Pure HMM**: Estimated 60-70% success (missing word filtering benefits)
- **Pure Word Filtering**: Estimated 85-90% success (struggles when multiple words match)
- **Pure RL**: Actually got 7.18% (as we saw during training)
- **Our Hybrid Approach**: **94.40% success**

What This Project Taught Us

Good System Design Often Trumps Fancy Machine Learning

- When domain knowledge and structured information are available, use them explicitly
- Do not hope neural networks will figure everything out on their own
- The path from 7% to 94% was not about bigger networks or longer training

The Winning Formula

- Use explicit reasoning and simple rules where possible
- Capture patterns with probabilistic models
- Refine details using learning algorithms
- This combination proved far more powerful than any single technique

The journey taught us that for structured problems like Hangman, intelligent system design that leverages multiple approaches is the key to success.