TrexQuant Hangman Game

August 21, 2025

Dear Harrison and Team,

The designed Hangman solver leverages probabilistic and heuristic strategies to select informative guesses. Building on a full dictionary of 227k words, we account for word length, positional frequencies, and letter dependencies. Early guesses prioritise high-information vowels, followed by consonants statistically likely given remaining candidates. This length-aware, pattern-driven approach forms the foundation of our solver.

We score candidate letters using Shannon entropy to estimate expected information gain, combining global letter frequencies, positional distributions, bigram models, and regex-filtered candidate dictionaries. N-gram substring lookups and morphological heuristics (prefixes/suffixes) provide additional context. A vowel penalty is applied when words are unusually vowel-heavy (vowel ratio > 0.52), balancing exploration and avoiding wasted guesses. Fallback strategies use global frequencies when no strong candidate emerges.

Evaluated on test batches, the baseline solver achieved a 67% win rate (27/40), while our final approach reached 73% (80/110), demonstrating the effectiveness of layered probabilistic modelling, entropy-driven scoring, and targeted heuristics in dynamically reducing the candidate set. Unfortunately we overfit and only achieve $\sim 53\%$ in the test set.

Best regards,

Ciaran O'Connor

1 Introduction

Effective Hangman strategies require more than simple reliance on general English letter frequencies, which are skewed by short, common words such as *the*, *and*, and *of*. Our solver considers letter probabilities within the context of a full dictionary, accounting for word length and positional dependencies. This approach highlights high-frequency vowels (E, I, A) and strategically important consonants (S, R, T, N, L), enabling first guesses to be tailored to word length and improving overall efficiency.

Dynamic, pattern-based decision-making is essential. Recursive elimination methods filter the dictionary based on current guesses, iteratively updating candidate word lists and letter probabilities. In practice, early guesses prioritise high-information vowels, followed by consonants most likely given the remaining candidates.

Key parts of the solver design include:

- **Entropy-based scoring:** select letters that maximise information gain by reducing the expected candidate set.
- **Regex-based candidate filtering:** combine with most-common-letter selection within the current word length.
- **Adaptive weighting:** shift from global frequencies to positional and bigram information as the word mask fills.
- **Length-conditioned letter frequencies:** $P(\text{letter} \mid \text{length})$ prioritises likely letters for a given length.
- **Positional frequencies:** $P(\text{letter at position} \mid \text{length})$ exploits letter position patterns.
- **Vowel-consonant ratio adjustment:** penalizes overrepresented vowels rather than applying hard cutoffs.
- **Bigram position frequencies:** $P(\text{letter}_{i+1} \mid \text{letter}_i)$ informs guesses based on neighboring letters.

These principles form the foundation of a length-aware, dynamically responsive solver. The main sources of strategy included [SharkFeeder(2009)] and [Carroll(2012)].

2 Exploratory Data Analysis

The Hangman dictionary contains approximately 227,300 words, with lengths from 1 to 29 letters and an average length of 9.35. Most words fall in the 7–11 letter range, highlighting the importance of medium-length words in gameplay. Global letter frequency analysis identifies the most common letters as `e, a, i, o, n, r, s, t, l`, with vowels averaging 39% of letters in a word. Early guesses should therefore prioritise high-frequency vowels.

Positional and morphological patterns further guide letter selection. Common first letters include `s, p, c, a, m`, while frequent final letters are `s, e, d, y, n`. Frequent suffixes (`-s, -ed, -ing, -ly, -er, -al`) and prefixes (`un-, re-, co-, pr-`) allow partial word information to reduce the candidate set effectively.

Entropy analysis shows that for typical 7–12 letter words, positional entropy ranges from 3.7 to 4.3 bits, indicating which positions are more or less predictable. Low-entropy positions can often be guessed with high confidence. Effective strategies combine initial high-frequency letter guesses with subsequent entropy-driven decisions, informed by positional and morphological cues.

Bigram and positional dependencies further refine predictions. For instance, `q` is followed by `u` nearly 98% of the time, whereas `t` has multiple likely successors (`i` 22%, `e` 21%, `o` 10%, `a` 10%, `r` 9%). Positional statistics reveal that 5-letter words often start with `s` (10%), and 8-letter words frequently end with `s` (20%) or `e` (16%). Seven-letter words commonly begin with `s` (10.6%), followed by `c, p, b`, reflecting English orthographic patterns.

Vowel ratio analysis shows that words longer than two letters have an average vowel fraction of 0.387, with the 95th and 99th percentiles at 0.545 and 0.600, respectively. Vowel-heavy words are rare; a 0.51 threshold marks statistically unusual density. This informs solver design, preventing overcommitment to vowels while maintaining efficient exploration of consonants and overall candidate reduction.

3 Methodology

3.1 Baseline Solver

Our initial Hangman solver integrates multiple probabilistic and heuristic strategies to select the most informative next guess. It combines global letter frequencies, positional distributions, n-gram substring matches, bigram continuations, and morphological cues such as common prefixes and suffixes. Each candidate letter is scored using Shannon entropy to estimate its expected information gain, ensuring guesses maximally narrow the solution space. While early experiments included additional adjustments such as vowel-ratio penalties, a simplified focus on basic vowel handling proved more consistent.

The solver builds three primary statistical models from a large training dictionary (227k words):

1. **Global frequency model:** counts occurrences of each letter across the entire dictionary.
2. **Positional frequency model:** tracks probabilities of each letter at each position for words of a given length.
3. **Bigram model:** estimates conditional probabilities of one letter following another.

Using these models, the solver employs multiple methods to generate candidate letters: filtering the current dictionary, searching n-gram substrings, evaluating positional and length-conditioned probabilities, and leveraging suffix/prefix heuristics. Each method produces a weighted candidate distribution scored with entropy-based metrics, with the highest-scoring letter ultimately chosen. Fallback strategies revert to global frequency-weighted guesses with optional vowel penalties when no clear candidate emerges.

Evaluated on test batches, this baseline solver achieves a $\sim 67\%$ win rate, serving as a reference point for later improvements.

3.2 Final Solver

Guided by exploratory data analysis on vowel ratios, frequency distributions, and conditional probabilities, the improved solver uses a layered strategy:

- Filter candidates with regex-matched dictionaries.
- Score letters via entropy-weighted frequency counts.
- Fall back to substring-based n-gram lookups or global frequency distributions when uncertainty is high.

This design balances statistical rigour with practical heuristics, allowing the solver to adapt dynamically while avoiding pitfalls identified during the trial phase. The entropy used for scoring is binary Shannon entropy:

$$H(p) = -\Big(p \log_2(p) + (1-p)\log_2(1-p)\Big),$$

where p is the fraction of candidate words containing a given letter.

Total Batch result: 80/110 wins (73% win rate)

4 Results

The solver achieved a success rate of 0.73 on the training set but only 0.527 on the final test set, reflecting a notable performance gap. This discrepancy highlights the limitations of relying heavily on dictionary-driven narrowing, since the evaluation explicitly excluded dictionary words from the final test set. As a result, strategies that exploit memorized word structures or patterns did not generalize effectively, and overfitting to the training dictionary reduced the solver's robustness when facing previously unseen words.

Solver Version	**Success Rate**
Initial Solver (Training Set)	0.67
Final Solver (Training Set)	0.73
Final Solver (Test Set)	0.53

Table 1: Comparison of solver performance across training and test sets.

A more general approach would have prioritized rules less dependent on specific word lists. For example, stronger use of positional letter frequencies, vowel-consonant balance thresholds, bigram and suffix probabilities, or entropy-based exploration could improve generalization. These principles operate at the level of English orthographic regularities rather than dictionary lookups, making them more adaptable to unseen words. Incorporating such heuristics would likely reduce overfitting and improve performance on test sets where dictionary overlap is deliberately prevented.

References

[Carroll(2012)] Carroll, J. S. (2012). A better strategy for hangman. `http://www.datagenetics.com/blog/april12012/index.html#google_vignette`. Accessed: 2025-08-20.

[SharkFeeder(2009)] SharkFeeder (2009). What in sam hell possessed me to waste my time with this. `http://www.sharkfeeder.com/hangman/`. Accessed: 2025-08-20.