Explanation of a Trie-Based Crossword Puzzle Generator

Kiruthees me23b246

June 2025

Overview

This project generates a 5x5 crossword puzzle using a dataset of 5-letter words and their clues. It builds a prefix trie for fast word lookup and uses scoring and backtracking to fill the grid such that all rows and columns form valid words.

1. Trie Data Structure

The `TrieNode` and `Trie` classes manage the prefix trie:

- `TrieNode` contains a dictionary of children, a boolean flag to mark end-of-word, and a list of (word, clue) tuples.
- `insert(word, clue)` inserts a word into the trie character-by-character and stores clues.
- `search(prefix)` returns all words that start with the given prefix.

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_word = False
        self.words = []
```

2. CrosswordPuzzle Class

This is the core engine that builds the puzzle grid.

Initialization

Initializes an empty 5x5 grid, a clue list, and loads all 5-letter words from the trie.

```
self.puzzle = [[' ']*5 for _ in range(5)]
self.clues = [[' ']*5 for _ in range(2)]
```

Word Placement Strategy

- A random first word is placed in row 0.
- For each subsequent row, the program scores each candidate word based on how well it fits with the vertical prefixes formed so far.
- The best scoring word is chosen unless randomness is introduced.

Scoring Function

`evaluate_word` adds a candidate word to the current grid and checks how many vertical prefixes can still form complete valid words using the trie.

```
prefix_sums = [self.evaluate_vertical_possibilities(prefix) for prefix in vertical_prefixes]
```

Backtracking and Max Attempts

If no valid word can be placed in a row, the generator restarts from scratch. This continues up to `max_tries = 500`.

3. Clue Generation

After a valid puzzle is built:

- Horizontal words are collected row-wise.
- Vertical words are collected column-wise.
- For each word, a clue is retrieved from the trie using `search()`.

```
self.clues[0][i] = matches[0][1] if matches else "No clue found"
```

4. Display Functions

The `display_puzzle()` and `display_clues()` methods print the puzzle and its clues in a formatted grid.

5. Execution Flow

1. Load all (word, clue) pairs into a Trie.
2. Create a `CrosswordPuzzle` object.
3. Call `generate()` to attempt puzzle construction.
4. Display final puzzle and clues.

Conclusion

This system uses a Trie to efficiently evaluate crossword feasibility row-by-row and ensures that all vertical words formed during the process are valid. Randomness allows for varied puzzle generation on different runs.