Math 640: Final Project Writeup

Ramesh Balaji, Nuray Kutlu, Daniel Elwell

Generating a Single Puzzle

The general approach we took to create our snake charmer puzzle is as follows:

1. Determine a starter word, and have the length of your puzzle (in words), maximum and minimum lengths of words (in letters), and the maximum and minimum lengths of the overlaps between words (in letters).
2. Using your starter word to start, generate a chain of puzzle length - 1 words by iteratively picking a random overlap amount O between the minimum and maximum overlap constraints, a random length L for the next word between the minimum and maximum word length constraints and find a word of length L that ends with the last O letters of the previous word by searching through the ENG() database, if it exists. This search is performed using the Followers function written by Dr. Z. If there is no such word, then decrease your overlap amount up to your minimum overlap amount until you find a word that works at that length. If there are no words at the minimum overlap, we fail. We do not decrease the word length because we want the puzzle to have a more even distribution of word lengths.
3. Lastly, when you get to your last word, attempt to find a finishing word that ends with your previous word and starts with your starter word with the same overlap and length constraints. If you iterate through all possible words and can't find any that match, loosen the length constraint by 1 (first for the overlap between the penultimate word, then for the overlap between the first word) until you find a matching finishing word. If there are no such overlaps, we fail again.

Generating a Large Number of Puzzles

We wrote a function that given an input JSON file with the parameters outlined in the previous section, along with the number of puzzles *n* to generate, the Maple program calls our puzzle generation function multiple times until the *n* puzzles are generated, repeating if the puzzle generating function returns FAIL. The Maple program outputs a JSON file for each generated puzzle in a specified directory. This function is called PuzzlesToJSON.

Our random approach is prone to failure. To solve this problem:

1. We parallelized our PuzzlesToJSON function, such that instead of looping *n* times to generate *n* puzzles, we create *n* threads in Maple using `Threads:-Create`**,** where each thread is repeatedly calling GeneratePuzzle. This significantly increased performance by working on multiple things at the same time.
2. We used the Rutgers HPC cluster, Amarel, and allocated a Slurm job requesting a CPU with 48 cores to scale our compute power so we could generate a large number of puzzles.

Then, we were able to generate 30 “easy” puzzles, 15 “medium” puzzles and 7 “hard” puzzles. The parameters for each puzzle is as follows:

- *Easy: https://github.com/yuv418/math640_snakecharmer/blob/master/easy.json*
- *Medium: https://github.com/yuv418/math640_snakecharmer/blob/master/medium.json*
- *Hard: https://github.com/yuv418/math640_snakecharmer/blob/master/hard.json*

Drawing the Puzzles

To draw the puzzles and allow people to play them interactively, we wrote a Python script using the Pygame library for rendering. The main puzzle view is constructed by drawing 2 circles using the Pygame draw.circle function, then by drawing "spokes" between the circles using the Pygame draw.line function. Lastly, the start labels and hints are drawn with Pygame's built-in text renderer. We also added functionality for reading keyboard input so that the user can play the puzzle interactively. Once the user solves the puzzle, their solution turns green to indicate it is correct. We compile this Python script to web assembly (WASM) and use Pygame's WebGL implementation so that the puzzles can be played interactively on a browser. This generates a static site so we could easily add it to the Math 640 website.

Generating Hints

We use the Facebook LLaMa3 large language model to generate hints for each word. We wrote a Python script to read the Puzzle JSON and used the `ollama` program to locally run this language model on a Rutgers iLab server to generate hints without paying for ChatGPT credits. The Python script is also parallelized to compute the hints for multiple words at the same time, to decrease the time it took to generate hints.

Our prompt was as follows:

> Imagine that you are writing a crossword puzzle. As the writer of the crossword puzzle, you must come up with a coherent, direct yet subtle, and clear hint to help the person playing the puzzle guess a word. Use the most common interpretation of the word when

> writing your hint. Do not interpret words as company names. For example, the word "dane" could be interpreted as "Dane," and a good hint for the word "dane" would be "A native of Denmark." However, the word "stat" could be interpreted as the UNIX command "stat," but this interpretation is too esoteric. Interpreting "stat" as "an abbreviation of the word statistic" is okay. The hint must be at least 6 words and at most 10 words. You cannot use the word in your hint. Do not say anything other than the hint. Please provide me with a hint for the word "**INSERT WORD HERE**."

While the LLM managed to make some of the hints confusing or difficult to guess, we were overall able to guess words from the hints for the puzzle. One reason the hints were difficult to guess were because some of the words in the word list are not commonly known. Of course, the other reason is because LLMs are known to hallucinate definitions.

This also means that all the puzzles are difficult, including the “easy” ones. Of course, the difficulty increases.

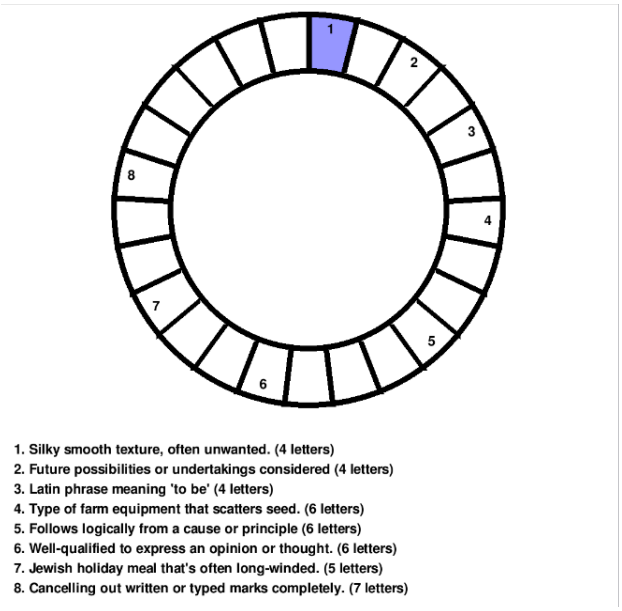
Converting Maple Format to JSON for Python

In order to create puzzles and convert results attained by generating a puzzle into a JSON format, we first read in a directory and a constraint file in JSON that contains information on word lengths, word overlaps, and the number of puzzles we want to generate with the given constraints into the given directory. We unpack the constraint file and put in the constraints as parameters to the GeneratePuzzle function and create the desired number of puzzles. Each time the loop to create a puzzle runs, the GeneratePuzzle function returns an array of the words in the puzzle, along with an array of starting positions for each of the words. This information is used to create a JSON file that contains an array of the starting positions of words, the words in the puzzle, and

the length of the puzzle. This JSON file is then exported into the directory specified. This creates the specified number of puzzles and exports them in JSON format into the specified directory.

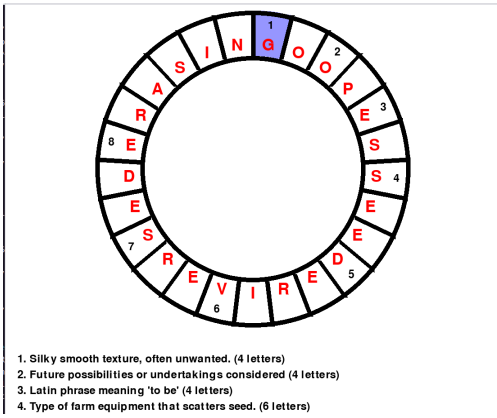
Final Results

Our puzzle book is here: http://snakecharmer.surge.sh/book.html

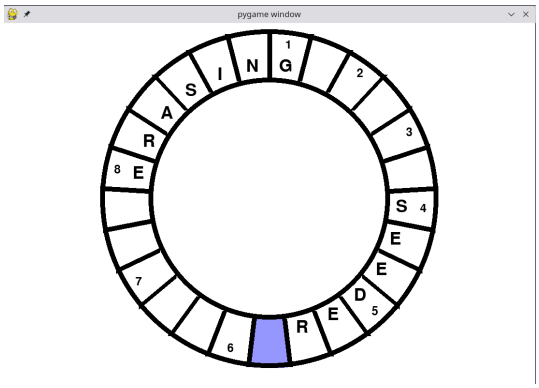


This is the final product.

The answer (obtained by pressing space):



You can guess by using arrow keys (left/right) and typing a letter.



And press space to check your work.