

Final Project Doc

As We May Speak

Lennox Krause

(All code which is not in this documentation is visible in the final project code)

The beginning

- In the Beginning, I was quite confused. Unlike in tech basics II, I had no idea on what to do for my final project.
- So, I asked perplexity AI. But I didn't like any of the suggestions perplexity gave me.
- Instead, I thought I could just use my idea from week 5, a game where you guess a book through the semantic similarity between the words you put in and the words of the book.
- To convert this idea to a fully-fledged game and final project, I still needed to add more game-like components and also more language generation and analysis methods we learned in class

12.01.2026 – Added randomly generated sentence:

```
tokens = [t.text for t in doc if not t.is_space]
transitions = defaultdict(list)
for w1, w2 in zip(tokens, tokens[1:]):
    transitions[w1].append(w2)
def generate_sentence(max_len=20):
    current = random.choice([w for w in tokens if w[0].isupper()])
    sentence = [current]

    for _ in range(max_len - 1):
        next_words = transitions.get(current)
        if not next_words:
            break
        current = random.choice(next_words)
        sentence.append(current)
        if current.endswith('.!', '!?', '?'):
            break

    return " ".join(sentence)
```

- So I tried to add a random sentence generation, like we did in week 3
- Right now, just very simple sentence generation – saving every single word and the next word as bigrams and then generating the next word out of all next words used in a book until a sign signalling the end of a sentence appears (".", "!", "?")
- This creates nonsensical sentences, which do give some clues on the book in question, but a reasonable sentence would obviously be better
 - Known Issues: Nonsensical Sentences, score system still not balanced (generating a whole sentence costs as much as getting semantic similarity to a single word)
- After presenting this in class, I realised that there is a way easier and better way to generate sentences and ngrams through markovify, which is what I added later on

14.01.2026 – Removing Names, too much clues!

- SpaCy already labels character names as “PERSON”, so I could just take these and replace them with anonymised versions of themselves
the result:

(write "s" to stop, write "c" to see generated sentence in style.)

Word in the grammar of text: Dorian

[('misty', 0.5145702958106995), ('empress', 0.42978018522262573), ('morrow', 0.4111132025718689), ('theodore', 0.4111132025718689), ('lain', 0.41075706481933594)]

Guesses left: 2

(write "s" to stop, write "c" to see generated sentence in style.)

Word in the grammar of text: Frankenstein

[('ray', 0.7494681477546692), ('paul', 0.7494681477546692), ('mark', 0.7494681477546692), ('hill', 0.7494681477546692), ('ford', 0.7494681477546692)]

Guesses left: 1

(write "s" to stop, write "c" to see generated sentence in style.)

Word in the grammar of text: c

Yes ” Mowgli .

Which of the following books is the one in question:

[‘At the Mountains of Madness’, ‘The Picture of Dorian Gray’, ‘Frankenstein’, ‘The Jungle Book’, ‘Platos Republic’]

Guess:

The words in style of text now work perfectly, but the randomly generated sentences are broken (Mowgli’s still even in there)

17.01.2026 – Custom Books, leaderboard, tidying up

- First, I added a small disclaimer in the start text about how to add custom books and how some features might not be optimised for this
 - I probably will change this whole starting paragraph later
- For the custom books to be fully implemented, I had to add os as a library and get the book names from the directory (os.listdir(“books”))
- Next, I wanted to fundamentally change how the game works. Instead of getting just 3 semantic clues, I wanted to enable the player to use as much clues as he wants.
 - This was really important, as I was adding more and more analysis tools, which wouldn’t all fit in 3 rigid guesses
- For using the semantic similarity tool, the player gets 0.5 clues penalty. For using a generated sentence, he gets 1
 - I also added a one-time option, listing the player the most common words (without stop words), giving 2 clue penalties (further implementation of SpaCy)
- Now, the goal would be to have as little clues needed for guessing the right book as possible

- And I also wanted to add a leaderboard to enable competing against friends
- To make the leaderboard organised and tidy, I structured it as a class with two different functions, add() and show()
- In add(), the users name (user input), the book chosen, the clues needed, and the current time is added to the leaderboard
- Afterwards, all entries are sorted by least clues used and time of playing (the newest entry beats the oldest in case of a tie right now, need to change that)
- In show(), the entries are shown sorted and in some crude form of a table

19.01.2026 – Analysing the text corpus

- I created a small program based on week 3s lesson with the task of analysing the text corpus of the books
- That way, I could maybe automate the text cleaning and improve the outputs of the game
 - The code for this is visible in the vocab_cleaner file
 - These were the outputs for “At the Mountains of Madness”:

most common words:

```
[('lake', 108), ('great', 91), ('old', 81), ('ones', 70), ('feet', 66), ('city', 66), ('camp', 65),
('land', 64), ('like', 61), ('certain', 59)]
```

most common adjectives:

```
[('great', 110), ('other', 91), ('certain', 59), ('high', 54), ('vast', 48), ('more', 45),
('many', 44), ('such', 43), ('new', 41), ('low', 37)]
```

most common verbs:

```
[('have', 98), ('make', 94), ('see', 92), ('seem', 89), ('find', 68), ('come', 65), ('be', 55),
('think', 50), ('leave', 49), ('know', 48), ('take', 43), ('do', 42), ('give', 38), ('look', 37),
('form', 36), ('show', 34), ('say', 33), ('tell', 31), ('keep', 29), ('use', 28)]
```

most common nouns:

```
[('thing', 95), ('foot', 87), ('city', 86), ('mountain', 75), ('man', 75), ('camp', 65), ('land',
63), ('place', 63), ('time', 58), ('world', 57), ('part', 56), ('plane', 53), ('ice', 51),
('sculpture', 46), ('course', 44), ('wind', 44), ('year', 44), ('woman', 43), ('sea', 42),
('point', 40)]
```

most common persons:

None ! They are called man woman and person instead to avoid giving to much info, which is why I marked these words green

- These were the outputs for “Platos Republic”:

most common words:

`[('said', 1044), ('true', 486), ('yes', 448), ('man', 422), ('good', 419), ('state', 370),
('like', 261), ('replied', 252), ('certainly', 235), ('men', 231)]`

most common adjectives:

`[('true', 497), ('other', 453), ('good', 396), ('great', 236), ('same', 228), ('own', 212),
('many', 163), ('such', 151), ('more', 127), ('unjust', 112)]`

most common verbs:

`[('say', 1511), ('have', 804), ('be', 333), ('make', 324), ('do', 269), ('reply', 261), ('see',
248), ('know', 215), ('think', 203), ('let', 200), ('take', 183), ('speak', 159), ('mean',
156), ('go', 148), ('come', 146), ('give', 142), ('suppose', 141), ('find', 126), ('tell', 121),
('ask', 115)]`

most common nouns:

`[('man', 653), ('thing', 257), ('soul', 231), ('nature', 230), ('one', 216), ('justice', 208),
('life', 194), ('way', 187), ('knowledge', 152), ('state', 148), ('art', 139), ('pleasure',
134), ('sort', 133), ('truth', 130), ('woman', 126), ('time', 126), ('good', 124), ('friend',
121), ('reason', 119), ('city', 117)]`

most common persons:

I

- I labelled them depending on the insight I thought they gave. This is obviously quite subjective, but as I have read most of the books, it is the best measurement I could find
- Green means that the words are quite unique to the style and topic of the book, yellow means that they don't give that much insight but are still better than nothing, and red means that they are quite irrelevant
- Nouns obviously give the most information, followed by adjectives, and verbs give the least
- Originally, I wanted to compare the book corpus to a list of common English words, to filter out words that give little insight
- But these lists always included words which I still found to be insightful and I didn't know where to draw the line.
- So instead, I just manually filtered out the capitalised nouns (All character names the anonymisation missed)
- A bit of randomness and unnecessary information in the project answers makes the game interesting

Next steps:

- Quadrigrams?

- Saving the Leaderboard in external file so it is usable over different sessions
- Adding some kind of double or nothing-like option at the end: Which sentence is real, which one generated, which sentence is from this book, which is a semantically similar one from another book
- Further improving text generation and corpus

- Maybe adding an adjective counter and an adjective median for the books?
 - Similar further analysis tools

20.2.2026 – Finally adding real ngram Markov logic

- I took another look at the content of week 3s session, and realised that Ngram text analysis and Markov chain text generation can be made way easier than I thought
- After a bit of testing, I decided on doing trigram generation with a hundred tries

21.2.2026 – making endgame logic work by adding another if clause to else clause, doing the same for clue system to avoid double clue gen

- I wanted to change up the endgame logic so that the player does not automatically lose when their input is without perfect grammar and spelling
- When I added another condition for that, the player would automatically win and lose
 - The reason for this was simple:

```
if book.lower() not in (b.lower() for b in
                       books): # if the book is not part of
                     the book list, ask him again
    print("Your guess seems not to be on the list. Did you
spell it correctly?")
else:
    # WARNING: simple else clause - in all other cases, the
    guess was seen as incorrect, even when guessing the correct
    book
    print("You didn't guess correctly!")
    print("The correct book was " + random_choice)
    won = False # you lose
    result = False
```

- Luckily, this error made me realise how to fix the same problem in my clue system, where the one letter commands were double counted as semantic similarity queries
 - After adding another if clause to both logic loops, any double firing of clauses was prevented

23.2.2026 – Added new logic to find giveaway words still unnoticed by the name anonymiser:

- As stated earlier, for further text analysis I just focused on finding and eradicating all character and place names from the book, if they were to telling
- For that, I just listed the 10 most frequent words which are uppercase nouns and not part of generic replacements ->The English language has the perfect feature for filtering out telling names, only names are capitalised, so even if they are not labelled as persons, I can easily find them

Final Steps:

- All that was missing now was cleaning up the code, writing this doc, and uploading everything

- While cleaning up, I also changed up the tutorial a bit because it was hard to follow. I made the ticks slower and added a user input to continue