# The Infinite Monster Engine

As a weird kid in a rural town, I heard about something strange and possibly sinister. I watched the Dungeons and Dragons cartoon on Saturday Morning TV. I saw the D&D advertisements in comic books. At the same time, moms on the evening news were gnashing their teeth about the influence of Satan. You mean I can pretend to be a wizard and there might be real demons involved? That’s a feature, not a bug. Still, I was poor and there was no internet, no way for me to actually get my hands on the books. D&D was a mystery to me. I didn’t know how it worked at all, so I just gathered some friends and we made up our own rules. I became a Dungeon Master. We played all the time, a handful of nerdy outcasts figuring out what features we needed to implement into this system as we went along. The game grew into this weird kludge of homebrewed calvinball.

In junior high, everything changed. We got our hands on a Monster Manual. Well, it was a photocopy of a photocopy of somebody’s cousin’s actual second edition Monster Manual. I’m not kidding. We kept it in a three-ring binder and I think I made my friends fight every single monster in that thing. Over time, you could tell which were the most popular. The Flail Snail was pristine paper, white and unbent. The page for the Drow was ragged and soft as an old dollar bill. I think someone spilled chili on it.

Now there are more monster books out there than you could ever use, but even then, prepping your game takes time. Pick just the right foes for the encounter and make sure they match the environment. Find something that’s not too powerful, but not going to end with a total party kill. If you’ve found the perfect set of monsters in a manual but they’re not the right difficulty rating, you may have to do some power scaling to even things out.

Or you could make an Infinite Monster Engine.

This has been my cheat code for almost a year now. It’s not complicated. Some of you are probably already doing it. Just tell your favorite Large Language Model (LLM) what you need. The better you style your prompt, the better results you get. But what if I wanted to automate as much of that as possible? Last time, I used the model to build a functional game of Asteroids (<https://codemag.com/Article/2411061/Can-an-LLM-Make-a-Video-Game>). Can we get it to build a factory of tabletop role-playing game encounters?

## The Game Master’s Preparations

I’m a planner, not a pantser. I want perfectly defined salvos of horror prepared for my players and I want to spend hours crafting that before a die is cast. Life is stupid sometimes, so the Powers That Be keep me from spending two weeks preparing for four hours. Not only that, but any Game Master (GM) knows that players will go out of their way to thwart your plans. Imagine: You’re mid-game, ready to unleash your own custom Tomb of Noteworthy Obstacles and Other Indignities upon your players, but then the rogue gets a wild hair. They’re not going to the Tomb. They’re going to explore the forest that you randomly mentioned and would prefer they ignore. If you’re agile, you can pull this off. You can pivot, but you weren’t prepared for this, and traditional encounter builders limit you to choosing between pre-made bad guys, leaving you with generic "Goblin #3" or "Wolf.”

### Building a Monster: What Are the Parameters?

**How many monsters do we need?** Any good GM will consider encounter dynamics over quantity. A single creature with a difficulty of Level 5 will play very differently from five creatures of Level 1 difficulty. I’ll limit it to 10 for our purposes. Larger encounters can slow the game down to a crawl. An encounter should provide meaningful choices, not just numerical victories. With too many foes, the action economy (how many actions each side gets) becomes off-kilter.

**The Difficulty Rating** is of particular importance, because with it, you can tell the LLM how to scale the encounter to be an appropriate match for the players. A scale of 30 aligns with the Fifth Edition player level cap and is intuitive for players and Game Masters alike.

**Environment Dropdown** options provide additional context for the model to work off of when customizing monsters. To make sure we get enough, I’ll provide a list of biomes. Each one influences how the LLM interprets the request. The Arctic option tells the LLM that the monsters should be enemies who can navigate icy terrain and thrive in subzero temperatures. The Jungle option should lead the LLM to design creatures that live in dense vegetation and can engage in combat using three-dimensional movement. Here are some popular options:

* Arctic
* Desert
* Forest
* Plains
* Jungle
* Mountain
* Swamp
* Cavern
* Underwater
* Urban

**Size Categories** are also generic, covering everything from sprites and familiars to something the size of the Great A’Tuin. Here are the ones I like:

* Tiny
* Small
* Medium
* Large
* Huge
* Gargantuan

Deciding on an assortment of **Enemy Types** will provide the model with some common archetypes. Here are some basic types:

* Humanoid
* Beast
* Demonic
* Construct
* Elemental
* Fey
* Phantasmal
* Eldritch
* Corrupted
* Ethereal

The lists I provided are off the top of my head, but each one has its own influence on both the narrative of the encounter and the mechanics of combat. The model will interpret each word and understand what it implies. For instance, Beasts will use primal instincts or maybe pack tactics. Phantasmal enemies will have incorporeal abilities and likely some sort of spectral attack.

### Persuasion Check: Engineering Your Prompt

A good prompt sets the tone for the conversation with the LLM and provides the basis for this experiment. Crafting the right prompt the right way is one of the most important aspects of working with an LLM sidekick. Be as thorough as possible from the get-go. That doesn’t mean you won’t be able to adjust as you go along, but make sure the LLM has a clear understanding of the project. Writing a good prompt requires clarity and specificity. Your request to the LLM has to be focused, actionable, and provide enough context to steer the response.

Know what you want to achieve going in and communicate that precisely. Instead of telling the LLM to “write a story," tell it to write a 5000-word tale about a marauder on some forgotten coast. Always include relevant background details to outline the request. The limits you place on the request provide some focus. Tell the LLM what to do and what not to do. Specify the audience, the voice, or the style. If necessary, explicitly allow for multiple iterations and refining. Start with verbs like "Build," "Interpret," "Generate," or "Explain" to prompt specific actions. Phrase your request to minimize misunderstanding, like you’re trying to cast a Wish spell or bargain with a demonic patron. Here are some ground rules:

* Define the goal clearly.
* Provide context.
* Set constraints or parameters.
* Be specific about style and tone.
* Ask for iteration or refinement.
* Incorporate examples.
* Use action-oriented language.
* Avoid ambiguity.

Think of the prompt as your plan. The more you plan, the more specific you can be, and the fewer iterations and corrections you’ll have to do down the line. Sure, you could say, “code a ttrpg encounter builder,” but then you’d spend the rest of the time massaging the LLM’s various responses into what you could have said in one more precise prompt.

After all of the sturm und drang, I come to this:

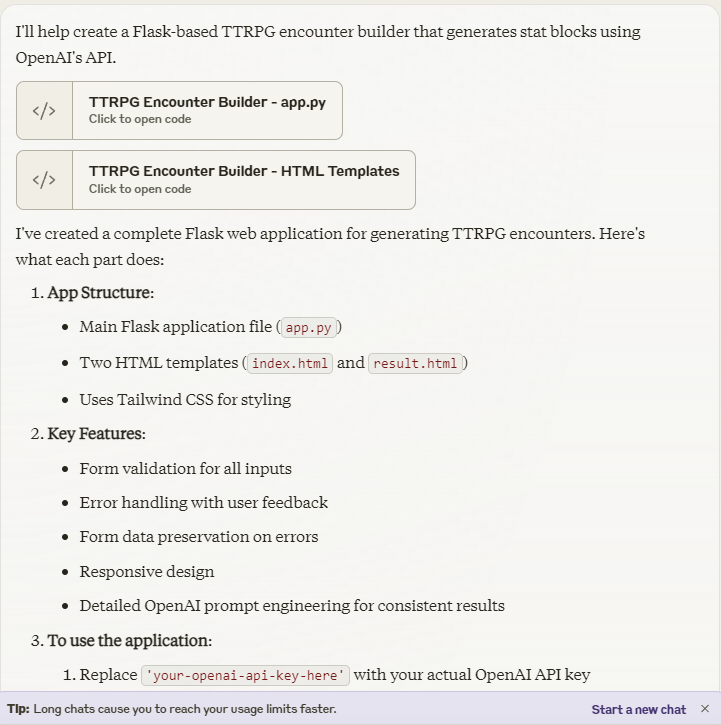
[AI Query]

Create a generic fantasy TTRPG encounter builder web app using Flask and the OpenAI API. The app should have a form with these fields: 1. Number of Enemies (number input) 2. Difficulty Rating (number input) 3. Environment dropdown (Arctic, Desert, Forest, Plains, Jungle, Mountain, Swamp, Cavern, Underwater, Urban) 4. Enemy Size dropdown (Tiny, Small, Medium, Large, Huge, Gargantuan) 5. Enemy Type dropdown (Humanoid, Beast, Demonic, Construct, Elemental, Fey, Phantasmal, Eldritch, Corrupted, Ethereal) 6. Additional Details (text area) The app should call the OpenAI API to generate complete and descriptive TTRPG stat blocks based on these inputs and display them on a new page. Use open-source mechanics generic or original terms. Include error handling and form data preservation. Include all necessary HTML templates and code comments explaining each part.

The LLM I’ve chosen for the build (not the API call) is Claude 3.5 Sonnet. Every model will approach the prompt a bit differently, but Claude is known for often having an edge when it comes to coding. You may get a different response than I did. These are not always consistent. Always question the LLM’s wisdom. It will be finicky. It will sometimes change its mind and start doing things with annoying inconsistency.

## Visiting the Oracle: Claude’s Response

**Figure 1** shows how Claude goes right to work. The model provides two sections containing the HTML and the Python code. After the code, Claude breaks down each part of the program, lists some of its key features and security notes, and shows how to implement the code.



**Figure 1:** Claude is concise and provides an immediate solution.

Opening the blocks shows you clean and lean Python code for the application.

from flask import Flask, render\_template, request,

flash

import openai

from markupsafe import escape

app = Flask(\_\_name\_\_)

app.secret\_key = 'your-secret-key-here'

# Required for flash messages

# OpenAI API Configuration

openai.api\_key = 'your-openai-api-key-here'

# Constants for form options

ENVIRONMENTS = ['Arctic', 'Desert', 'Forest',

'Plains', 'Jungle', 'Mountain', 'Swamp',

The HTML is similarly basic, prioritizing function over form, but it also tells you exactly what you’ll need going forward.

### Preparing the Spells: Libraries, APIs, etc.

Before you run it, you have to set up the environment. You’re going to need the Python libraries referenced, determine which LLM you’ll use, and get the respective API key. Flask is a lightweight web framework for Python that allows you to build web applications quickly and with minimal code. GPT-4's grasp of TTRPG mechanics hasn’t failed me in the past. Its ability to maintain consistency while introducing creative elements helps create encounters that are both mechanically sound and narratively engaging. MarkupSafe is a Python library that protects web apps from XSS attacks that use special characters in user input.

pip install flask

pip install openai

pip install markupsafe

The API key will vary by which LLM you choose to work with. For these purposes, OpenAI’s API works seamlessly with Python. It’s incredibly easy to use. Sign up on <https://platform.openai.com/> to generate as many of the alphanumeric keys as you need.

### The Ancient Words: The Code

If you examine pieces of the working implementation, starting with the core application code, you can see that the entirety of it is lean and efficient. It offers an easy way to implement your LLM’s API key.

client = OpenAI(api\_key=

'your-openai-api-key-here')

Implement strict validation to ensure that all inputs meet your requirements before making API calls.

if num\_enemies < 1 or num\_enemies > 10:

raise ValueError

("Number of enemies must be between 1 and 10")

Using escape() prevents XSS attacks by sanitizing user input.

environment = escape(request.form.get

('environment', ''))

Claude assigns a simple role to OpenAI that it thinks will be best suited to the task. I do see one technical problem, however, and it’s a game-breaker. If you’re up to date with OpenAI API protocols, you may see it, too.

# Call OpenAI API response =

openai.ChatCompletion.create(

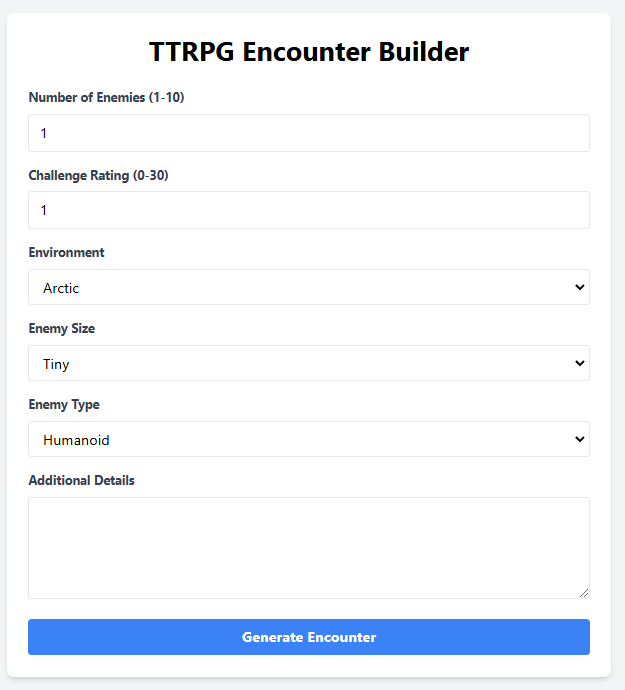
model="gpt-4

",

I’m going to leave it for now to see how Claude handles the error.

### Casting the Spell: Running the Code

The first time you run the file, you’re greeted with the simple interface seen in **Figure 2**.



**Figure 2:** Brutally simple, it’s exactly what I imagined.

I tell the Infinite Monster Engine that I want seven monsters with a difficulty rating of 3. This is an Arctic environment and they’ll be tiny. Make them flesh eating ice elves!

Unfortunately, I don’t get far. After I press **Generate Encounter**, I get the first error.

**“An error occurred while generating the encounter. Please try again.”**

That really doesn’t help at all, does it? I take that error and feed it back into Claude.

[AI Query]

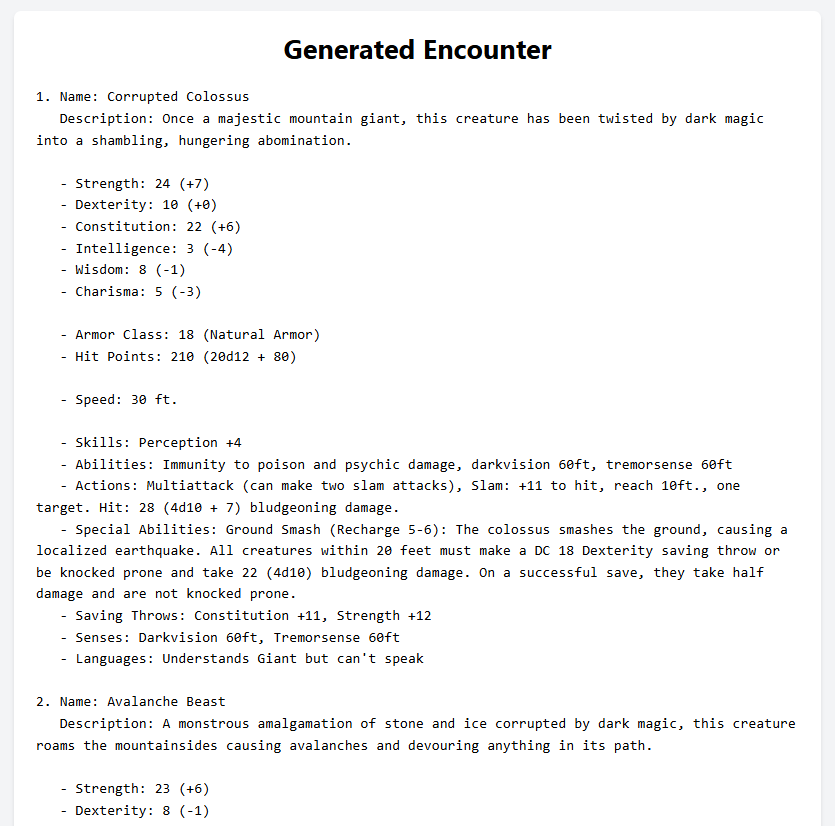
“I see you're getting an error with the encounter generation. The error is likely occurring because the OpenAI integration needs to be updated to use their latest API client and format. Let me fix that by updating the API call portion of the code.”

Claude doesn’t waste any time in identifying the issue. It immediately creates a new version, a list of the changes, and how to make sure the OpenAI Python library is up to date. After it updates the code, it calls out the problem I spotted earlier.

Key changes made:

* Updated the OpenAI import to use from openai import OpenAI
* Changed the client initialization to use the new format with client = OpenAI()
* Updated the API call to use client.chat.completions.create()
* Added more detailed error logging to help diagnose issues
* Modified the error handler to display the specific error message

It was using an outdated protocol to call OpenAI’s API, but after the error, it finds and corrects it immediately. I made the required changes and refreshed the page right away. Now my current players are high-level murder hobos masquerading as liberators. They think they’re fancy. I want them to reconsider their place in the world. I want to give them something to cry about. From the Infinite Monster Engine, I request three Huge Corrupted monsters with Difficulty Ratings of 15. These creatures live in the mountains and are “Shambling, gigantic horrors that are mindless and hungry.” Check out our new array of baddies in **Figure 3**.



**Figure 3:** Witness the birth of a shambling abomination!

Clutch your dice tightly, folks! This gang of beasts from **Figure 3** is ready, armed with horrible tactics sure to instigate a rousing battle. (I’ll let you know how it goes.) I asked for a **stat block**. It provided everything I needed, without requiring me to specify. Claude uses plain language for the request to OpenAI, based on common TTRPG stat block requirements. You can see this in plain language in the code itself.

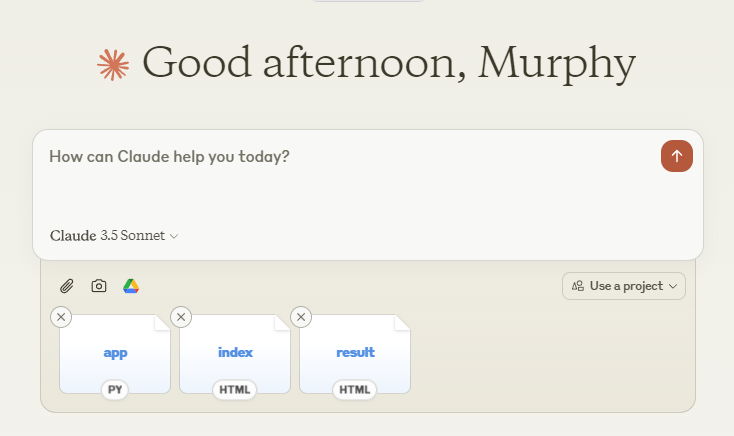
* Name and description
* Attributes
* AC/HP
* Movement
* Skills/Abilities
* Actions
* Saving throws
* Senses/Languages

## Leveling Up

This is a great shortcut and can trim hours from your prep time, but it could be better. I forgot XP and loot! Before I go any further, let’s participate in some LLM hygiene.

### Harvesting XP and Grinding for Loot: Basic Improvements

Best practice for conversations with Claude is to start a new chat when your original becomes unwieldy. With longer chats, you’ll reach your usage limit much faster. Longer chats also push the limits of the LLM’s context window. LLMs can appear to "forget" the context of a conversation over time due to the limitations of their context window and the design of their architecture. Without true memory, they are stateless and thus require explicit repetition to hold onto the right context. As your conversation grows, the model can have difficulty tracking multiple threads. You’re going to start a new chat, but you’ll upload the files to Claude so that it doesn’t start from zero. **Figure 4** shows Claude’s clean interface.



**Figure 4:** I reupload our files to a fresh conversation with Claude.

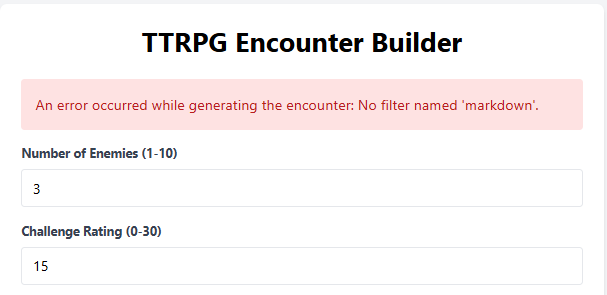
The prompt I provide is short and to the point. I’m hoping Claude can see the logic of its own code and immediately make the adjustments I’m looking for.

[AI Query]

“Please use this existing code but make small changes. Add loot and XP specific to each creature to every stat block. Also, please style the Result page to be a bit easier to read.”

### An Unfortunate Side Quest

Here’s where the car flies off the road. I just don’t know it yet. Claude provides the updated files I requested, along with a bulleted list of changes. I don’t bother with all of that. I already provided a detailed roadmap in the form of the three files, so I lazily copy and paste. How autonomous are you, Claude? As anyone would expect, something has gone awry in **Figure 5**.



**Figure 5:** I’ve hit a strange snarl.

I pass the error over to Claude.

[AI Response]

Ah, we need to add the Flask-Markdown extension to handle Markdown formatting. Let's fix this by modifying the app.py file and installing the required dependency.

We’re dealing with markdown now? Since when? If you’re not paying attention and keeping at least one hand on the wheel, Claude will wreck the car. I let it. From here, Claude leads us down a road of compounding errors.

[AI Response]

Ah, I apologize—the package name has changed. Let's use the current version of Flask-Markdown called flask-markdown.

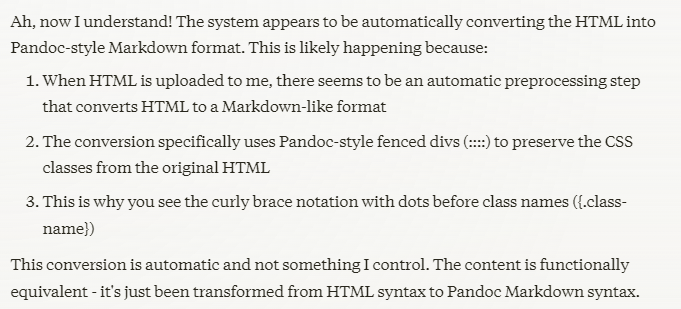
[AI Response]

Claude:Ah yes—I just noticed that while I updated the import statement in my code, I shouldn't have included Flask-Markdown at all since your HTML template doesn't require Markdown processing anymore.

[AI Response]

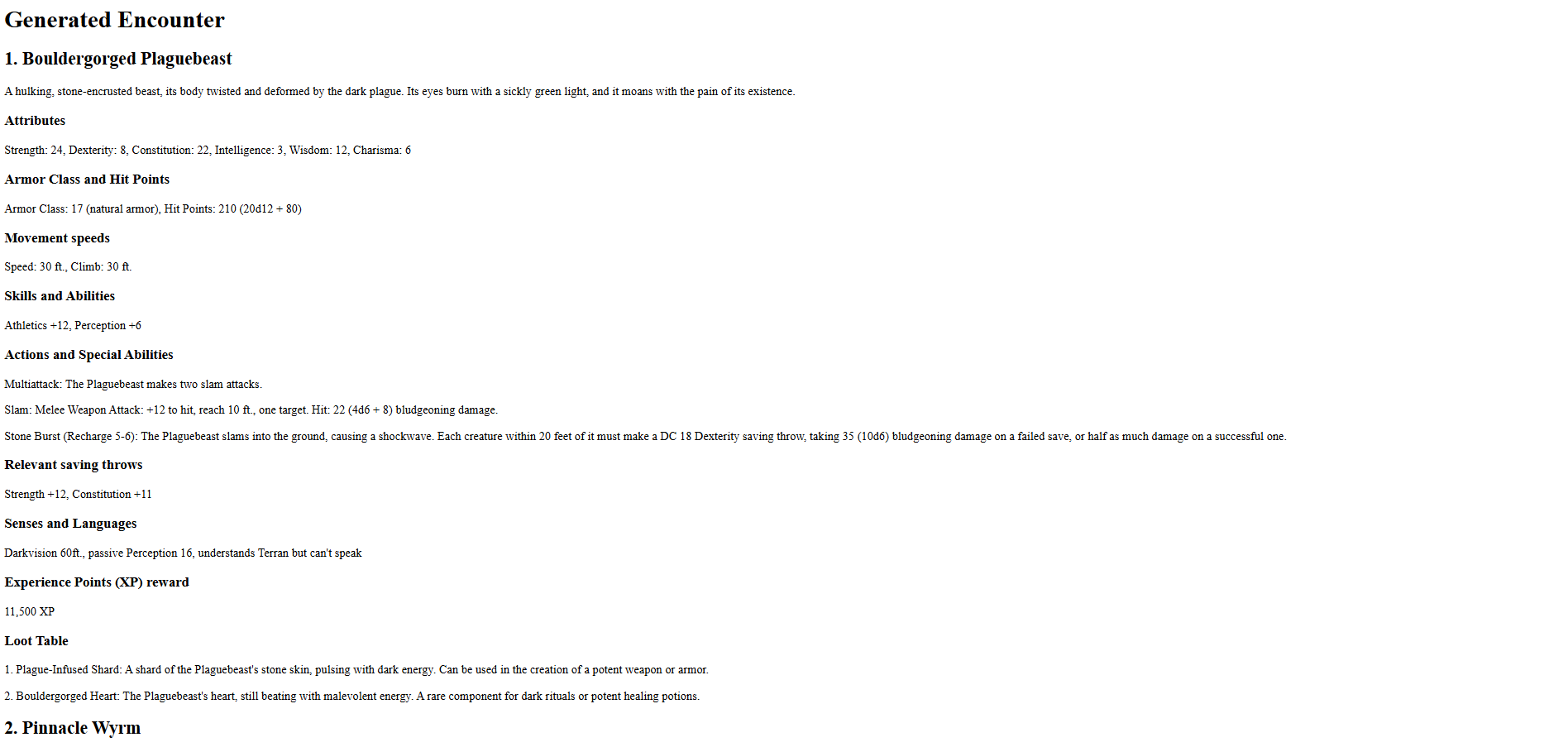
“Looking at the given HTML files, there's another issue.”

With gentle prodding, Claude gets back on track. Somehow, it had changed all of the original HTML to markdown, as it discovers in **Figure 6**. The original files I uploaded were correct functional code. They were, that is, until I uploaded them to Claude. From the beginning, Claude immediately parsed and interpreted the HTML files incorrectly. It converted the HTML into markdown right when I uploaded them, but of course didn’t update any of the Python code required in the app.py file.



**Figure 6:** I guess I should have checked the uploads.

Now the Infinite Monster Generator gets its upgrade, as in **Figure 7**.



**Figure 7:** No one expects the Bouldergorged Plaguebeast!

A Plague-Infused Shard! I’ve always wanted one. **Listing 1** offers a look at the final Python Code. **Listing 2** reflects the HTML input on the index page, and **Listing 3** shows the HTML for the output.

The Infinite Monster Engine is lean and functional, but the work of a Game Master doesn’t stop there. Where does the adventure take us? What additions should we make? Automating a chunk of the preparation can free you up to be more creative. If this gives you the itch, think about GenAI produced battle maps, initiative trackers, or NPCs your players can actually chat with.

Current Benchmarks for LLMs

Although there are currently no official performance standards for LLMs, there are some general metrics for comparison. Not all LLMs are made the same. Your mileage may vary.

Some of these markers include criteria like sentiment analysis, common sense, fairness and bias, and diverse academic areas.

Currently, Claude 3.5 Sonnet from Anthropic is marked as one of the best for coding, but these change rapidly because the models are constantly evolving.

Getting the Right API Key

OpenAI’s API is incredibly easy to use. You can generate keys as needed with just a few short clicks. Access to the API does cost money, so make sure you pay attention to the token limits.

It's not the only LLM on the market, of course. I can’t speak for the ease of use of the others, but swapping them out within this code is surely a simple task.

Remember to never share your API key with anyone! That could be a very expensive mistake.

**Listing 1:** The final Python code

<div class="max-w-2xl mx-auto bg-white rounded-lg shadow-md p-6">

<h1 id="generated-encounter"

class="text-3xl font-bold mb-6 text-center">

Generated Encounter

</h1>

{% with messages = get\_flashed\_messages(with\_categories=true)

%}

{% if messages %}

{% for category, message in messages %}

<div class="mb-4 p-4 rounded {% if category ==

'error' %}bg-red-100 text-red-700{% else %}bg-green-100

text-green-700{% endif %}">

{{ message }}

</div>

{% endfor %}

{% endif %}

{% endwith %}

<div class="stat-blocks space-y-8 whitespace-pre-wrap">

{{ stat\_blocks | safe }}

</div>

<div class="mt-6 text-center">

<a href="{{ url\_for('index') }}" class="bg-blue-500

hover:bg-blue-700 text-white font-bold py-2 px-4

rounded inline-block">

Generate Another Encounter

</a>

</div>

<style>

.stat-blocks h2 {

@apply text-2xl font-bold mb-4 pb-2 border-b-2

border-gray-300 text-blue-800;

}

.stat-blocks h3 {

@apply text-xl font-semibold mb-3 mt-4

text-gray-700;

}

.stat-blocks p {

@apply mb-3 leading-relaxed;

}

.stat-blocks ul, .stat-blocks ol {

@apply mb-4 pl-6;

}

.stat-blocks li {

@apply mb-2;

}

.stat-blocks strong {

@apply text-gray-700;

}

/\* Special styling for XP and Loot sections \*/

.stat-blocks h3:contains("XP"),

.stat-blocks h3:contains("Loot") {

@apply text-green-700 mt-6;

}

</style>

</div>

**Listing 2:** Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width,

initial-scale=1.0">

<title>TTRPG Encounter Builder</title>

<link href="https://cdn.jsdelivr.net/npm/tailwindcss

@2.2.19/dist/tailwind.min.css" rel="stylesheet">

</head>

<body class="bg-gray-100 min-h-screen p-8">

<div class="max-w-2xl mx-auto bg-white

rounded-lg shadow-md

p-6">

<h1 class="text-3xl font-bold mb-6 text-center">

TTRPG Encounter Builder</h1>

{% with messages =

get\_flashed\_messages(with\_categories=true) %}

{% if messages %}

{% for category, message in messages %}

<div class="mb-4 p-4 rounded

{% if category == 'error' %}bg-red-100

text-red-700{% else %}bg-green-100

text-green-700{% endif %}">

{{ message }}

</div>

{% endfor %}

{% endif %}

{% endwith %}

<form action="{{

url\_for('generate\_encounter') }}"

method="post" class="space-y-4">

<div>

<label class="block text-gray-700

text-sm font-bold mb-2">

Number of Enemies (1-10)

</label>

<input type="number" name="num\_enemies"

min="1" max="10"

value="{{ form\_data.num\_enemies

if form\_data else '1' }}"

class="w-full px-3 py-2 border

rounded" required>

</div>

<div>

<label class="block text-gray-700

text-sm font-bold mb-2">

Difficulty Rating (0-30)

</label>

<input type="number"

name="challenge\_rating" min="0"

max="30" step="0.125"

value="{{ form\_data.challenge\_rating

if form\_data else '1' }}"

class="w-full px-3 py-2 border

rounded" required>

</div>

<div>

<label class="block text-gray-700

text-sm font-bold mb-2">

Environment

</label>

<select name="environment" class="w-full

px-3 py-2 border rounded" required>

{% for env in environments %}

<option value="{{ env }}"

{% if form\_data and

form\_data.environment ==

env %}selected{% endif %}>

{{ env }}

</option>

{% endfor %}

</select>

</div>

<div>

<label class="block text-gray-700

text-sm font-bold mb-2">

Enemy Size

</label>

<select name="enemy\_size"

class="w-full px-3

py-2 border rounded" required>

{% for size in sizes %}

<option value="{{ size }}"

{% if form\_data and

form\_data.enemy\_size ==

size %}selected{% endif %}>

{{ size }}

</option>

{% endfor %}

</select>

</div>

<div>

<label class="block text-gray-700

text-sm font-bold mb-2">

Enemy Type

</label>

<select name="enemy\_type"

class="w-full px-3 py-2 border

rounded" required>

{% for type in types %}

<option value="{{ type }}"

{% if form\_data and

form\_data.enemy\_type ==

type %}selected{% endif %}>

{{ type }}

</option>

{% endfor %}

</select>

</div>

<div>

<label class="block text-gray-700

text-sm font-bold mb-2">

Additional Details

</label>

<textarea name="additional\_details"

rows="4" class="w-full px-3 py-2

border rounded">{{

form\_data.additional\_details

if form\_data else ''

}}</textarea>

</div>

<button type="submit"

class="w-full bg-blue-500

hover:bg-blue-700 text-white font-bold

py-2 px-4 rounded">

Generate Encounter

            </button>

        </form>

    </div>

</body>

</html>

**Listing 3:** Result.html

<div class="max-w-2xl mx-auto bg-white rounded-lg

shadow-md p-6">

<h1 id="generated-encounter" class="text-3xl

font-bold mb-6 text-center">

Generated Encounter

</h1>

{% with messages = get\_flashed\_messages

(with\_categories=true) %}

{% if messages %}

{% for category, message in messages %}

<div class="mb-4 p-4 rounded {% if

category == 'error' %}bg-red-100

text-red-700{% else %}bg-green-100

text-green-700{% endif %}">

{{ message }}

</div>

{% endfor %}

{% endif %}

{% endwith %}

<div class="stat-blocks space-y-8

whitespace-pre-wrap">

{{ stat\_blocks | safe }}

</div>

<div class="mt-6 text-center">

<a href="{{ url\_for('index') }}"

class="bg-blue-500 hover:bg-blue-700

text-white font-bold py-2 px-4 rounded inline-block">

Generate Another Encounter

</a>

</div>

<style>

.stat-blocks h2 {

@apply text-2xl font-bold mb-4 pb-2

border-b-2 border-gray-300 text-blue-800;

}

.stat-blocks h3 {

@apply text-xl font-semibold mb-3 mt-4

text-gray-700;

}

.stat-blocks p {

@apply mb-3 leading-relaxed;

}

.stat-blocks ul, .stat-blocks ol {

@apply mb-4 pl-6;

}

.stat-blocks li {

@apply mb-2;

}

.stat-blocks strong {

@apply text-gray-700;

}

/\* Special styling for XP and Loot sections \*/

.stat-blocks h3:contains("XP"),

.stat-blocks h3:contains("Loot") {

@apply text-green-700 mt-6;

}

</style>

</div>