[Tutorials Basic - PyGame Tutorials (wikidot.com)](http://pygametutorials.wikidot.com/tutorials-basic)

DJOKOQUEST DOC

To integrate OpenAI's API into your CLI text-based adventure for dynamic character interactions, you can use the `langchain` library as it makes the process of using language models more straightforward for specific applications like yours. Given the requirements you've specified, here's how you can approach this:

1. \*\*Install Langchain\*\*: First, ensure you have `langchain` installed. If not, you can install it using pip:

```bash

pip install langchain

```

2. \*\*OpenAI API Key\*\*: Ensure you have an OpenAI API key. You can obtain this from the OpenAI website by creating an account and accessing the API section.

3. \*\*Implement Character Interaction\*\*: We'll create a scenario where the player can interact with characters defined through system prompts, utilizing langchain's seamless integration with OpenAI's models. This example will demonstrate how to set up a basic interaction.

### Example Implementation

First, you need to set up Langchain and the OpenAI API:

```python

from langchain.llms import LangChainWrapper, OpenAI

# Initialize the OpenAI LLM with your API key

openai\_api = OpenAI(api\_key="your\_openai\_api\_key")

# Initialize LangChain Wrapper

langchain\_api = LangChainWrapper(llm=openai\_api)

```

Next, you create a system where characters can be interacted with, incorporating memory functionality. For the sake of simplicity, we'll use a basic memory implementation here. Since you specified that keeping history persistently is not necessary, this example focuses on short-term memory during individual interactions.

```python

class Character:

def \_\_init\_\_(self, name, instructions):

self.name = name

self.instructions = instructions

self.memory = []

def interact(self, user\_input):

# Adding user input to memory for context

self.memory.append(user\_input)

# Using LangChain API to generate response

response = langchain\_api.llm.generate(

prompt=self.generate\_prompt(),

stop\_sequences=["\n"],

max\_tokens=250,

)

# Output the response

print(f"{self.name}: {response}")

# Keeping the last 3 interactions in memory

if len(self.memory) > 3:

self.memory.pop(0)

def generate\_prompt(self):

# Combine instructions with memory of past interactions to form prompt

prompt = self.instructions + " ".join(self.memory)

return prompt

# Example of creating a character and interacting

if \_\_name\_\_ == "\_\_main\_\_":

character\_instructions = "You are a wise wizard who provides guidance to adventurers."

wizard = Character(name="Merlin", instructions=character\_instructions)

while True:

player\_input = input("You: ")

if player\_input.lower() == "quit":

print("Ending conversation.")

break

wizard.interact(player\_input)

```

### Points to Note:

- Each `Character` object includes a memory list that stores user inputs for context in subsequent interactions, up to a limit (in the example, the last 3 interactions are kept).

- The `interact` method takes player input, adds it to memory, generates a response using Langchain coupled with OpenAI, and manages memory history.

- This simple memory system does not save interactions permanently; once the program ends, the history is lost, aligning with your requirement that it's not necessary to remember previous interactions.

\*\*Important\*\*: You must replace `"your\_openai\_api\_key"` with your actual OpenAI API key.

\*\*Security Note\*\*: When integrating AI models into games, especially ones that accept user input, consider implementing safety checks or content filters to ensure the responses are appropriate for your audience.