

# LLM per API nutzen



# Google AI Studio

- [https://aistudio.google.com/app/prompts/new\\_chat](https://aistudio.google.com/app/prompts/new_chat)

```
import google.generativeai as genai

genai.configure(api_key="YOUR_API_KEY")
model = genai.GenerativeModel("gemini-1.5-flash")
response = model.generate_content("Explain how AI works")
print(response.text)
```

# Schach mit Gemini-API

- Erste Idee
  - rein textbasiert → langweilig
- Besser: GUI
  - einfache Schach-GUI in GitHub suchen → aufwendig, nicht wirklich einfach
  - GUI von Gemini erzeugen lassen

Du bist ein Schachspieler. Spieler weiß beginnt mit dem Zug Bg2-g3. Was ist dein nächster Zug?



Als Schachspieler würde ich auf den Zug Bg2-g3 von Weiß wie folgt antworten:

Schwarz: ... Lg7

## Begründung

- **Entwicklung:** Schwarz entwickelt mit dem Zug ... Lg7 eine wichtige Figur und bringt seinen Königsläufer in eine aktive Position.
- **Fianchetto:** Der Zug ... Lg7 ist ein klassisches Fianchetto, bei dem der Läufer auf der langen Diagonalen positioniert wird und das Zentrum und den Königsflügel kontrolliert.
- **Flexibilität:** ... Lg7 ist ein flexibler Zug, der es Schwarz ermöglicht, verschiedene Pläne zu verfolgen und auf Weiß' Züge zu reagieren.

## Alternative Züge

- ... Sf6: Dieser Zug ist auch möglich und entwickelt einen Springer, der das Zentrum kontrolliert.

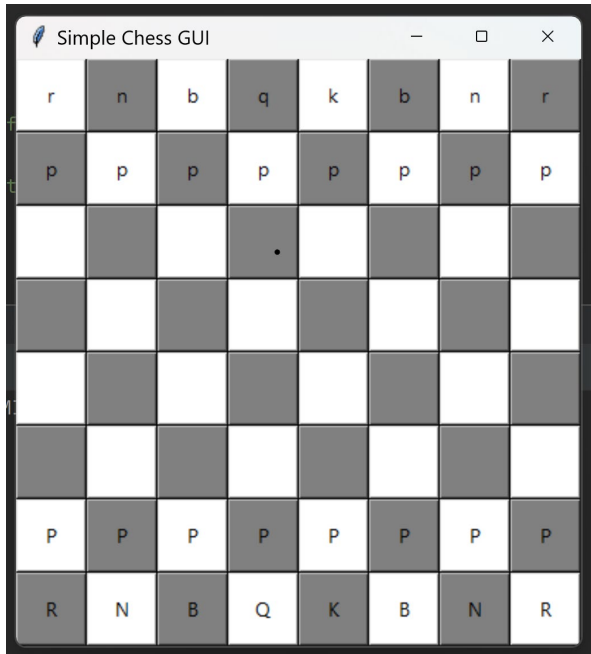


Gemini fragen





Generate code for a simple chess gui in python.



Python



```
import tkinter as tk
import tkinter.messagebox as messagebox

class ChessGUI:
    def __init__(self, master):
        self.master = master
        master.title("Simple Chess GUI")

        self.board = [
```



Gemini fragen





AI ruft einfach  
Zufallsgenerator  
auf.

add the AI opponent.



Absolutely! Let's add a basic AI opponent to your chess GUI. This AI will make random, legal moves. It's not very smart, but it's a start!

Here's the updated code:

Python



```
import tkinter as tk  
import random
```



Gemini fragen



```
from_coords, to_coords = random.choice(legal_moves)
```





- Verwendet JSON und baut Nachrichten selbst zusammen.

Change the AI opponent such that it asks Gemini through its API for the next move.



Python



```
import tkinter as tk
import requests
import json

# ... (Existing ChessGUI code)

class ChessGUI:
    # ... (Existing code)

    def ai_move(self):
        # 1. Get the current board state in a format Gemini understands
        board_state_string = self.convert_board_to_string()

        # 2. Construct the prompt for Gemini
        prompt = f"""
You are a chess AI. Given the following board state in algebraic notation,
determine the best move for Black. Respond with only the move in algebraic
```



Gemini fragen





change the code such that it uses the google.generativeai package



Python



```
import tkinter as tk
import random
import google.generativeai as palm # Updated import

# ... (Existing ChessGUI code)

class ChessGUI:
    def __init__(self, master, palm_client): # Add palm_client
        self.master = master
        # ... (Rest of the __init__ code)
        self.palm_client = palm_client # Store the palm client

    def ai_move(self):
        board_state_string = self.convert_board_to_string()

        prompt = f"""
```



Gemini fragen





Verwendet  
(anscheinend?)  
nicht die  
aktuelle API

```
def ai_move(self):
    board_state_string = self.convert_board_to_string()

    prompt = f"""
    You are a chess AI. Given the following board state in algebraic notation,
    determine the best move for Black. Respond with only the move in algebraic notation.

    Board State:
    {board_state_string}

    Black to move.
    """

    try:
        response = self.palm_client.generate_text(
            model="models/gemini-pro", # Updated model name (or another Gemini model)
            prompt=prompt,
            temperature=0, # Set temperature. 0 means deterministic.
        )
        ai_move = response.result.strip() # Updated response access

    if self.is_valid_algebraic_move(ai_move):
        from_coords, to_coords = self.convert_algebraic_to_coords(ai_move)
        self.move_piece(from_coords, to_coords)
```





- Händisch auf aktuelle API umgestellt.
- Statt Spielfeld wird nun Folge der Züge übertragen.

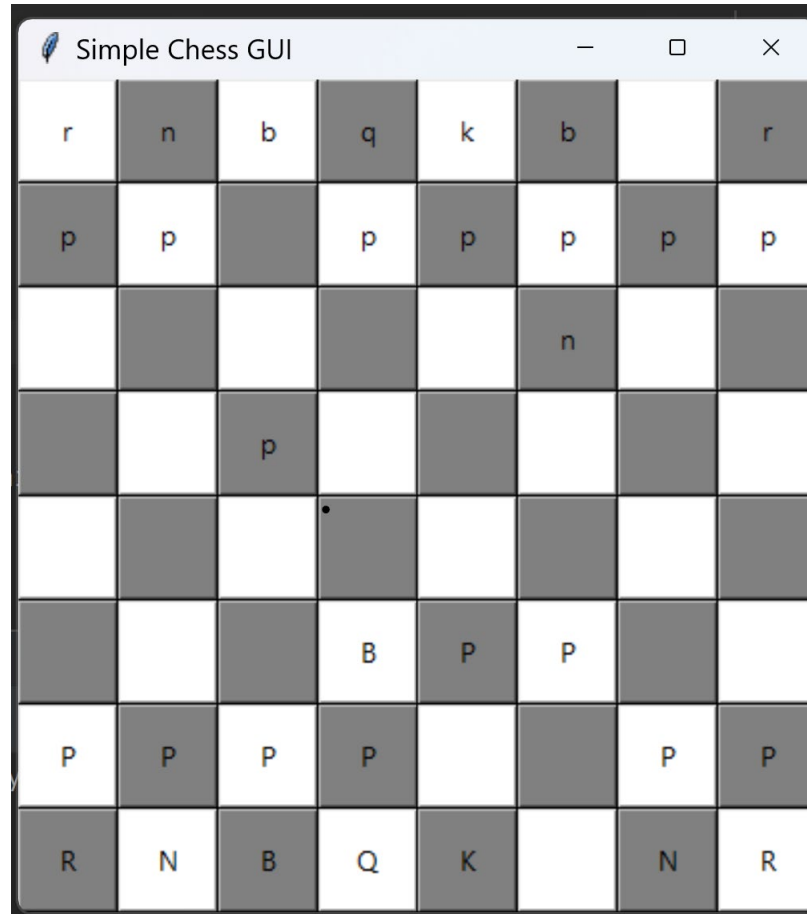
```
You are a chess AI. The following moves have been done:  
e2e3 c7c5 d2d3  
Respond with only the move in long algebraic notation.  
Black to move.
```

```
def ai_move(self):  
    firstPrompt = f"""  
    You are a chess AI. White has made the following first move:  
    {self.last_move}  
    Respond with only the move in long algebraic notation.  
    Black to move.  
    """"  
  
    nextPrompt = f"""  
    You are a chess AI. The following moves have been done:  
    {self.moves}  
    Respond with only the move in long algebraic notation.  
    Black to move.  
    """"  
  
    try:  
        if self.is_first_move:  
            prompt = firstPrompt  
            self.is_first_move = False  
        else:  
            prompt = nextPrompt  
  
        response = self.genai_client.generate_content(prompt)  
        ai_move = response.text.strip()  
  
        if self.is_valid_algebraic_move(ai_move):  
            from_coords, to_coords = self.convert_algebraic_to_coords(ai_move)  
            self.move_piece(from_coords, to_coords)  
        else:  
            print("Invalid move from Gemini:", ai_move)
```



- White: d2d3
- Black: d7d5
- White: c1e3
- Invalid move from Gemini: ...Nf6

**Format “Springer nach f6” kann das Programm noch nicht parsen.**



# Gemini AI als Schachspieler

- White: e2e3
- Black: c7c5
- White: f1d3
- Black: g8f6
- White: f2f3
- Invalid move from Gemini: Nf6

**Der schwarze Springer steht schon auf f6.**

# Codererzeugung und – bewertung mit der API

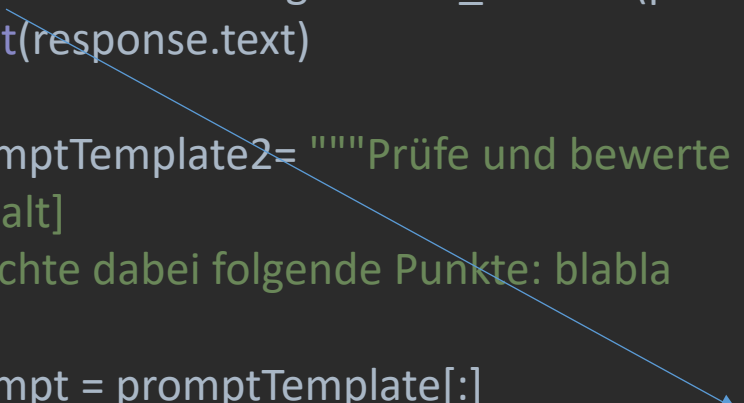
```
promptTemplate1 = promptTemplate= """Hier könnte Ihre Frage stehen.  
[Schlüssel1]  
blabla  
Blabla [Schlüssel2] blabla"""
```

```
genai.configure(api_key=mine.geminiAPIKey)  
model = genai.GenerativeModel("gemini-1.5-flash")
```

```
prompt = promptTemplate[:]  
prompt = prompt.replace("[Schlüssel1]", wert1)  
prompt = prompt.replace("[Schlüssel2]", wert2)  
response = model.generate_content(prompt)  
print(response.text)
```

```
promptTemplate2= """Prüfe und bewerte den folgenden Inhalt:  
[Inhalt]  
Beachte dabei folgende Punkte: blabla  
"""
```

```
prompt = promptTemplate[:]  
prompt = prompt.replace("[Code]", response.text)  
response = model.generate_content(prompt)  
print(response.text)
```



```
x = 1
```

# Iterative Kodeverbesserung

```
# Counter variable, initialized to 1  
counter = 1
```

```
# Counter for tracking the number of iterations  
counter: int = 1 #Python example, for other languages adjust type accordingly
```

```
def my_function(data):  
    """Example function demonstrating improved counter usage."""  
    iteration_count = 0 # More descriptive name  
    for item in data:  
        iteration_count += 1  
        # Process item  
        print(f"Processing item {item}, iteration {iteration_count}")  
    return iteration_count  
  
my_data = [1,2,3,4,5]  
total_iterations = my_function(my_data)  
print(f"Total iterations: {total_iterations}")
```



```
x = 1
```

# Iterative Kodeverbesserung

```
initial_inventory_count = 1 # More descriptive variable name
```

```
initial_widget_inventory_count = 10 # More specific, reflects likely scenario where initial count is not 1
```

# Or better still, using a dictionary for more flexible and maintainable inventory:

```
initial_inventory = {  
    "widgets": 10,  
    "gadgets": 5,  
    "sprockets": 25  
}
```

# Iterative Kodeverbesserung

```
from typing import Dict
```

```
[[[
```

```
initial_inventory: Dict[str, int] = {
```

```
    "widgets": 10,
```

```
    "gadgets": 5,
```

```
    "sprockets": 25
```

```
}
```

```
def remove_item(inventory: Dict[str, int], item: str, quantity: int) -> Dict[str, int]:
```

```
    """Removes a specified quantity of an item from the inventory.
```

```
    Args:
```

```
        inventory: The current inventory.
```

```
        item: The name of the item to remove.
```

```
        quantity: The quantity to remove.
```

```
    Returns:
```

```
        The updated inventory. Raises ValueError if insufficient quantity.
```

```
    """
```

```
    if item not in inventory:
```

```
        raise ValueError(f"Item '{item}' not found in inventory.")
```

```
    if inventory[item] < quantity:
```

```
        raise ValueError(f"Insufficient quantity of '{item}' in inventory.")
```

```
    inventory[item] -= quantity
```

```
    return inventory
```

```
#Example Usage
```

```
updated_inventory = remove_item(initial_inventory.copy(), "widgets", 3) #Using copy to avoid modifying original
```

```
print(updated_inventory)
```

```
updated_inventory = remove_item(updated_inventory, "widgets", 10) #this will raise an error
```

```
print(updated_inventory)
```