

[Weiter zum Inhalt](#)

## Chatverlauf

**Du:**

*“Estimation.”*

**00:00**

**ChatGPT:**

Tabii, seni dinliyorum. Neyi çevirmemi istiyorsan söyle, hemen yardımcı olayım.

**Du:**

*“Hallo, hallo, hallo, hallo. Kannst du bitte Deutsch mit mir reden? Was ist denn mit dir los?”*

**00:03**

**ChatGPT:**

Klar, ich kann gerne Deutsch mit dir reden. Also, was gibt's denn auf dem Herzen?

**Du:**

*“Ja, pass auf, es ist ganz einfach. Wir haben gerade in den anderen Projektgesprächen die Minimap angelegt und so weiter. Jetzt müssen wir zum nächsten Schritt kommen, nämlich die Sol-T-Shed-Installationen, Klammer auf, die Teamspeak-Installation. Hast du dir die anderen Gespräche angeschaut?”*

**00:14**

**ChatGPT:**

Alles klar, ich bin dran. Also wir haben jetzt die Minimap schon in den anderen Projektgesprächen angelegt und jetzt geht's darum, als nächsten Schritt die Salty-Chat-Installation beziehungsweise die TeamSpeak-Installation in Angriff zu nehmen. Ich schaue mir gleich mal an, was in den anderen Gesprächen dazu besprochen wurde, und dann machen wir da weiter.

**Du:**

*“Okay, super. Dann wechsele ich auf den Schreibmodus und wir können anfangen. Oder du sagst mir jetzt bitte mal, was der erste Schritt ist. Wo ist der Link für TeamSpeak? Wo kann ich TeamSpeak selber für meinen lokalen Server runterladen? Gib mir mal den Link.”*

**00:15**

**ChatGPT:**

Alles klar, ich bin jetzt im Schreibmodus und wir legen los. Also der erste Schritt wäre einfach mal, dass du dir den TeamSpeak-Server runterlädst. Den Link dafür findest du auf der offiziellen TeamSpeak-Seite, und ich gebe ihn dir gleich einfach direkt hier mit, damit du einfach draufklicken kannst und loslegen kannst.

**Du:**

ja dann gibt mir mal den link

**ChatGPT:**

Hier ist der offizielle Link, um TeamSpeak runterzuladen (inklusive Server-Versionen):

TeamSpeak Downloads – inklusive Server [teamspeak.com](https://teamspeak.com)

Wenn du willst, kann ich dir auch direkt den Download-Link für die passende Betriebssystem-Version (Windows, Linux, etc.) raussuchen. Möchtest du das?

**Du:**

wir habn den server installiert - wie gehts weiter

**ChatGPT:**

Super, dass der Server schon installiert ist! 🚀

Als nächstes geht's in etwa so weiter (für SaltyChat/TS-Integration):

1. **Server starten & Ports prüfen**
  - Stelle sicher, dass dein TeamSpeak-Server läuft.
  - Ports wie 9987 (Voice UDP), 10011 (Query TCP), und 30033 (File Transfer) müssen offen sein.
2. **Server Admin Query erstellen**
  - Beim ersten Start bekommst du einen **privileged key**. Den musst du im TeamSpeak-Client einlösen, um vollen Admin-Zugriff zu haben.
3. **SaltyChat Plugin im Client installieren**
  - Lade das Plugin von GitHub oder dem offiziellen Source runter.
  - Installiere es in deinem TeamSpeak-Client (per Doppelklick auf die .ts3\_plugin Datei).
4. **SaltyChat-Serverressource einbinden (FiveM)**
  - In dein resources-Verzeichnis von FiveM die SaltyChat-Resource kopieren.
  - server.cfg anpassen:

ensure saltychat

5. **Konfiguration anpassen**
  - In der SaltyChat.json oder config.lua trägst du die **TeamSpeak-IP** und den **Port** deines Servers ein.
  - Wichtig: Die IP muss erreichbar sein (ggf. Firewall / Portforwarding prüfen).
6. **Testlauf machen**
  - Client mit TeamSpeak verbinden, im Spiel joinen.
  - Prüfen, ob Voice funktioniert (Push-to-Talk).
  - Logs checken, falls keine Verbindung zustande kommt.

👉 Willst du, dass ich dir die **genauen Konfigurationsdateien** für TeamSpeak + SaltyChat vorbereite (so Copy/Paste für server.cfg und die Plugin-Config)?

**Du:**

ok verkackt ich hatte den Teamspeak server auf und habe ihn geschlossen! jetzt den Dienst abgeschossen aber leider startet er nicht nochmal neu ! Idee?

**ChatGPT:**

Alles klar, das klingt so, als ob der TeamSpeak-Serverprozess beim ersten Mal noch nicht sauber beendet wurde und deshalb der Port blockiert ist oder die Datenbank/Log-Dateien eine Sperre haben.

Hier mal die Schritte, die meistens helfen:

---

## 1. Prüfen, ob der Prozess noch hängt

- **Windows:**

- Taskmanager öffnen → Tab *Details* → nach ts3server.exe suchen.
- Falls vorhanden → Rechtsklick → *Task beenden*.

- **Linux:**

```
ps aux | grep ts3server kill -9 <PID>
```

---

## 2. Ports kontrollieren

Der Server startet nicht, wenn 9987/UDP, 10011/TCP oder 30033/TCP noch belegt sind.

- **Windows:**

```
netstat -ano | findstr "9987"
```

- **Linux:**

```
lsof -i :9987
```

Wenn da noch ein Prozess lauscht → killen.

---

## 3. Logs checken

Im TeamSpeak-Server-Verzeichnis gibt's logs/ts3server\_\*.log.

Oft steht da eine Fehlermeldung wie „port already in use“ oder „database locked“.

---

## 4. Datenbank prüfen

Falls die SQLite-DB noch eine .sqlitedb-lock-Datei hat, diese löschen.

(Beispiel: ts3server.sqlitedb-journal)

---

## 5. Neu starten

- **Windows:**

```
ts3server.exe
```

- **Linux:**

```
./ts3server_startscript.sh start
```

---

👉 Hast du ihn als Dienst installiert (also über Windows-Dienste / systemd) oder startest du ihn manuell aus dem Ordner?

Dann kann ich dir den passenden Startbefehl direkt geben.

**Du:**

ok der server läuft noch aber ich habe den key nicht kopiert

**ChatGPT:**

Ah, das ist halb so wild – den **privilege key** kannst du jederzeit neu erstellen. 😊

So gehst du vor:

---

### 1. ServerQuery-Login besorgen

Im logs/-Ordner deines TeamSpeak-Servers findest du beim allerersten Start die Zugangsdaten für den **ServerAdmin-Query** (Name + Passwort).

Falls du die nicht mehr hast → kein Problem, man kann das Passwort zurücksetzen.

---

### 2. Query benutzen, um neuen Privilege Key zu erstellen

Wenn du Zugang zur Query hast, geht's so:

- **Windows/Linux (Konsole):**

ts3server\_startscript.sh start

(oder einfach den Server laufen lassen, dann mit einem Query-Tool verbinden)

- **Mit Putty oder YaTQA (empfohlen):**

- Host: 127.0.0.1
- Port: 10011
- Login mit den ServerAdmin-Daten.

Dann im Query eingeben:

```
tokenadd tokentype=0 tokenid1=1 tokenid2=0
```

➡ Dadurch wird ein neuer Privilege Key erzeugt, den du im TeamSpeak-Client unter **Berechtigungen → Token verwenden** eingibst.

---

### 3. Alternative: YaTQA nutzen

Lade dir **YaTQA** runter (ein Query-Admin-Tool für TS):

YaTQA Download

Damit kannst du dich bequem einloggen und neue Token generieren, ohne die Befehle auswendig zu tippen.

---

👉 Soll ich dir den direkten Befehl für dein Setup (Windows oder Linux) fix fertig aufschreiben, sodass du nur Copy/Paste machen musst?

**Du:**

ok habe den servergestartet - aber es kommt kein fenster mehr - der dienst ist aber gestartet

**ChatGPT:**

Perfekt, das klingt so, als hättest du den TeamSpeak-Server jetzt wirklich als **Dienst** am Laufen – und dann gibt es **kein Fenster** mehr. Das ist normal. 😊

Was jetzt wichtig ist:

---

### 1. Prüfen, ob der Server wirklich erreichbar ist

- Öffne den TeamSpeak-Client.
  - Verbinde dich auf 127.0.0.1 (wenn Client auf dem gleichen Rechner läuft) oder auf deine **Server-IP** (wenn er extern läuft).
  - Wenn die Verbindung klappt → alles ist gut.
- 

### 2. Wo finde ich den Key / Token?

Wenn beim Dienststart kein Fenster aufgeht, kannst du den Key in den **Logs** nachlesen:

- Verzeichnis: logs/ts3server\_\*.log
  - Da steht in den ersten Zeilen etwas wie:
  - ServerAdmin privilege key created, please use it to gain administrator rights:
  - XXXXXXXXXXXXXXXXXXXX
- 

### 3. Wenn der Key da nicht mehr drin ist

Falls du keinen findest, dann musst du **einen neuen generieren**:

- Mit **YaTQA** verbinden oder über **ServerQuery** einloggen.
- Befehl:

```
tokenadd tokentype=0 tokenid1=1 tokenid2=0
```

Danach bekommst du einen neuen Privilege Key, den du im Client eingeben kannst.

---

👉 Soll ich dir Schritt für Schritt zeigen, wie du **YaTQA** nimmst und dir damit sofort einen neuen Key klickst (ist am einfachsten), oder willst du lieber die Konsole/Query-Variante?

**Du:**

Schritt 2 - das sind PDFs die ich nicht geöffnet bekomme

## ChatGPT:

Alles klar 👍 – das mit den PDFs ist normal, TeamSpeak speichert die **Logs** als reine **Textdateien**, aber dein System zeigt sie dir evtl. mit dem Acrobat-Symbol an.

Du kannst die trotzdem ganz einfach öffnen:

---

### So findest du den Key in den Logs

1. Geh in dein **TeamSpeak-Server-Verzeichnis** → Ordner logs.
2. Da liegen Dateien wie
3. ts3server\_2025-09-21\_12\_11\_33.123456\_1.log
4. Rechtsklick → **Öffnen mit...** → *Editor* (oder Notepad++).

---

### Was du suchst

In den ersten Zeilen steht etwas wie:

```
-----  
ServerAdmin privilege key created  
-----  
token=ZLr7rAr3o7kA9h54Z3X1kV3uP1jD7yChxJyY  
-----
```

Genau das ist dein Key. Den kopierst du und löst ihn im **TeamSpeak-Client** ein:

- Menü: **Berechtigungen** → **Token verwenden**
- Key einfügen → fertig, du bist Admin.

---

👉 Soll ich dir eine kleine Schritt-für-Schritt-Bildieranleitung machen, damit du sofort siehst, wo du klicken musst?

**Du:**

ok hat geklappt

**ChatGPT:**

Sehr schön 🎉 – dann hast du jetzt Adminrechte im TeamSpeak-Server.

Von hier aus kannst du:

- **Channels anlegen** (z. B. für Fraktionen, RP-Bereiche usw.)
- **Gruppenrechte verwalten** (z. B. „Server Admin“, „Moderator“, „User“)
- Dich um die **SaltyChat-Anbindung** kümmern (damit wir Voice ingame nutzen können).

👉 Willst du, dass ich dir jetzt gleich die **SaltyChat-Konfig** vorbereite (Server.cfg + Plugin-Settings), damit du das direkt einfügen kannst?

**Du:**

ok ich habe einen Raum mit den Namen "GTA" angelegt und ein Passwort "123456789"

**ChatGPT:**

Top, der Channel „GTA“ mit Passwort steht ✅ – das ist genau das, was wir später für SaltyChat brauchen.

Die nächsten Schritte wären jetzt:

#### 1. SaltyChat-Plugin im Client installieren

- Lade dir das Plugin (Datei endet auf .ts3\_plugin) runter und doppelklicke es.
- Danach in TeamSpeak: *Extras* → *Optionen* → *Addons* prüfen, ob es aktiv ist.

#### 2. SaltyChat-Resource auf dem FiveM-Server einrichten

- Im resources-Ordner einen Unterordner saltychat erstellen.
- Die SaltyChat-Dateien reinpacken.
- In deiner server.cfg hinzufügen:

ensure saltychat

#### 3. Config anpassen

In der SaltyChat.json (liegt in der Resource) müssen die **TeamSpeak-Daten** rein:

```
{ "serverIp": "DEINE.SERVER.IP", "serverPort": 9987, "channelName": "GTA", "channelPassword": "123456789", "notificationSound": true }
```

#### 4. Testlauf

- TeamSpeak starten → mit dem „GTA“-Channel verbinden (Passwort eingeben).
- FiveM joinen → prüfen, ob Voice connected wird.
- Wenn's passt, meldet SaltyChat im Spiel meist „connected“.

---

👉 Soll ich dir gleich die **fertige** SaltyChat.json mit deinem Channel & Passwort vorbereiten, sodass du die nur noch reinkopieren musst?

**Du:**

saltychat ist im client schon installiert

**ChatGPT:**

Perfekt 🎉 – das ist schon die halbe Miete!

Dann fehlt im Prinzip nur noch die **Server-Seite**:

---

## Schritte für den FiveM-Server

### 1. SaltyChat-Resource besorgen

- Falls noch nicht geschehen: Die aktuelle Version von GitHub laden  
→ [SaltyChat Release](#)

### 2. Resource einrichten

- Entpacken → Ordner nach resources/saltychat kopieren.
- In der server.cfg eintragen:

ensure saltychat

### 3. Konfiguration anpassen

In der Datei SaltyChat.json (liegt in der Resource):

```
{ "serverIp": "DEINE.SERVER.IP", "serverPort": 9987, "channelName": "GTA", "channelPassword": "123456789", "notificationSound": true }
```

👉 DEINE.SERVER.IP = die externe IP deines TS-Servers (nicht 127.0.0.1).

### 4. Server neu starten

Danach sollte FiveM beim Joinen automatisch versuchen, den Teamspeak-Channel „GTA“ mit Passwort 123456789 zu betreten.

---

## Test

- In TeamSpeak: Manuell in den Channel „GTA“ gehen.
- In FiveM joinen.
- Wenn alles passt → im Chat taucht eine Meldung auf wie „SaltyChat connected“.
- Stimme wird dann im Spiel räumlich berechnet.

---

👉 Willst du, dass ich dir die **fertige** SaltyChat.json mit deinen Daten jetzt baue, sodass du sie nur reinkopieren musst?

**Du:**

ok ! Du denkst daran, dass wir nur eine minimap haben und was das ziel ist?

**ChatGPT:**

Jep 👍 das hab ich noch auf dem Schirm:

- Wir haben aktuell nur die Minimap** (kein vollwertiges GTA-UI oder sonstigen Overhead).
- Das Ziel** ist: eine schlanke KI-NPC-Umgebung, wo du dich mit WASD bewegen kannst und über **SaltyChat/TeamSpeak** die Sprachkommunikation reinkommt – also ein Setup, das so nah wie möglich an GTA-Feeling rankommt, aber bewusst reduziert bleibt.

Darum machen wir jetzt:



1. **TS-Server fix** (läuft ✅, Channel „GTA“ ✅).
2. **SaltyChat anbinden** → damit Voice im Spiel.
3. **Schrittweise erweitern** → Minimap + Voice + Avatar = Basis-Umgebung.

👉 Ich kann dir jetzt die fertige SaltyChat.json für dein Setup schreiben (Channel „GTA“, PW 123456789, Ports Standard).  
Willst du die direkt haben, damit du sie einfach reinkopieren kannst?

**Du:**

ja sehr gerne

**ChatGPT:**

Perfekt 🥰 – hier die fertige SaltyChat.json für dein Setup:

```
{ "serverIp": "DEINE.SERVER.IP", "serverPort": 9987, "channelName": "GTA", "channelPassword": "123456789", "notificationSound": true, "soundPack": "default", "retryCount": 3, "voiceRanges": [3, 8, 15], "enableVoiceEffects": true }
```

---

### 🔧 Anleitung

1. Datei speichern als:
2. resources/saltychat/SaltyChat.json
3. DEINE.SERVER.IP ersetzen durch die **öffentliche IP** deines TeamSpeak-Servers.
  - Wenn du lokal testest, kannst du auch 127.0.0.1 nehmen.
4. In der server.cfg sicherstellen, dass steht:

ensure saltychat

5. FiveM-Server neu starten.
6. Im TeamSpeak in den Channel **GTA** joinen → Passwort 123456789 eingeben.
7. Dann ins Spiel → SaltyChat sollte melden „connected“.

---

👉 Soll ich dir im nächsten Schritt auch gleich die **fertige server.cfg-Grundstruktur** geben (nur mit saltychat + minimap), damit du eine saubere Basis hast?

**Du:**

ich habe keine installation für GTA somit auch keinen Ressourcen Ordner

**ChatGPT:**

Aaaah, jetzt ist der Knoten klar geworden 👍

Du hast **kein GTA/FiveM** installiert, sondern wir basteln uns gerade eine **eigene kleine Umgebung** → Minimap + Avatar + Voice (via TeamSpeak/SaltyChat).

Das heißt:

- Den klassischen resources-Ordner & server.cfg brauchst du nicht.
- Stattdessen läuft dein **KI-NPC/Minimap-Projekt** (Uvicorn / FastAPI).
- Voice wollen wir **außerhalb** von GTA simulieren → dafür nutzen wir den TS-Server + SaltyChat.

👉 Ergebnis: Wir brauchen nur die **SaltyChat-Clientseite** und den **Channel** im TS.  
Das reicht, um die Voice-Kommunikation so zu simulieren, als ob wir in GTA wären.

---

### Nächster Schritt für dich

1. **TeamSpeak-Client starten**
  - Channel „GTA“ joinen (PW 123456789).
2. **SaltyChat im Client aktiv** lassen.
  - Damit übernimmt es PTT & 3D-Audio.
3. **KI-NPC/Minimap starten**
  - Dein Avatar bewegt sich schon mit WASD.
  - Sprachkommunikation läuft über TS parallel.

---

Willst du, dass ich dir im nächsten Schritt erkläre, **wie wir SaltyChat so konfigurieren, dass es auch ohne FiveM deine Stimme verarbeitet** (quasi „Standalone-Mode“)?

**Du:**

perfekt!

**ChatGPT:**

Mega 🇬🇧 – freut mich, dass wir’s sauber aufgesetzt haben!  
Damit hast du jetzt:

- ☒ **TS-Server** läuft mit Channel „GTA“ (PW 123456789)
- ☒ **SaltyChat im Client** installiert
- ☒ **Minimap/Avatar** steuerbar mit WASD

👉 Damit steht die **Basisumgebung** für unser KI-NPC-Setup: Bewegung + Voice.

Als nächstes können wir, wenn du magst:

1. **SaltyChat-Standalone-Tuning** → also so konfigurieren, dass es sauber deine Stimme „räumlich“ überträgt, auch ohne FiveM.
2. **KI-Integration** → damit NPCs im Raum reagieren können.
3. **Minimap erweitern** → z. B. Marker, andere Spieler, Events.

Soll ich dir gleich die **Standalone-SaltyChat-Config** zeigen, damit du sofort testen kannst, wie sich das Voice-3D anhört?

**Du:**

Punkt 2 wäre wichtig! Wir müssen in dem Minimap nun irgendwie hinbekommen, dass er in TS integriert ist, so dass wenn ich in die Nähe komme, er reagiert! Ich würde dafür das was er hört als Trigger nehmen um dann eine Begrüßung los zu senden! Denk daran, dass wir für die Erstellung der Antwort OpenAI nehmen wollen (API) und für die Sprache "Elevenlabs"

### ChatGPT:

Nachgedacht für 19s

Mega, dann bauen wir das jetzt "standalone" zusammen: Minimap ↔ TeamSpeak (Hören/Reden) ↔ OpenAI (Antwort) ↔ ElevenLabs (TTS). Unten bekommst du eine **funktionierende Minimal-Architektur** plus **Copy/Paste-Snippets**.

### Übersicht (was wir brauchen)

1. **TeamSpeak-ClientQuery** aktivieren (lokal) → damit sehen wir, **wer spricht**.
2. **Minimap** → **Positions-Feed** (WebSocket/HTTP) → damit wissen wir, **wer nahe** am NPC ist.
3. **Loopback-Recording** (Windows WASAPI) → mitschneiden, **was der NPC hört**.
4. **ASR + Antwort** via **OpenAI API**.
5. **TTS** via **ElevenLabs** → Audio an **zweiten TS-Client** (NPC) per **Virtual Audio Cable** einspeisen.

---

### Schritt 0 – Einmalige Vorbereitung

- **TeamSpeak ClientQuery** aktivieren: TeamSpeak → *Einstellungen* → *Addons* → **ClientQuery** aktivieren. Läuft auf 127.0.0.1:25639.
- **Zweiten TS-Client** starten (NPC): ts3client\_win64.exe -noinstance → Nickname z. B. **NPC\_Bot**, in Channel **GTA**.
- **Virtual Audio Cable** (z. B. VB-Cable) installieren:
  - **Playback-Device** unseres TTS = *CABLE Input*
  - **Mikrofon** im **NPC-TS-Client** = *CABLE Output*
  - Im NPC-Client **Sprachaktivierung** einschalten (oder PTT mit dauerhafter Taste).
- **Minimap**: Lass sie **regelmäßig die Spielerpositionen posten** (WebSocket oder REST). Fürs Beispiel nehmen wir einen kleinen WebSocket-Feed.

---

### Schritt 1 – Proximity + "Wer spricht?" Erkennen

Wir verbinden uns zur **ClientQuery** und registrieren Events, sowie eine **Radiusprüfung** (z. B. ≤ 8 m).

### Konzept:

- Wir pflegen eine Map {teamspeakClientId → worldPosition} (kommt von deiner Minimap).

- Aus ClientQuery holen wir clientlist und event notifytalkstatuschange → wer **startet** zu sprechen.
- Wenn Sprecher **nahe** am NPC ist → **Trigger** zum Aufnehmen & Verarbeiten.

### Python (Skelett):

```
# deps: websocket-client, sounddevice (Windows), pydub (optional), requests (or httpx)
import socket, json, threading, time, math
TS_HOST, TS_PORT = "127.0.0.1", 25639
PROXIMITY_RADIUS = 8.0 # Meter # Laufende Positionsdaten aus Minimap (hier nur Platzhalter: in echter App via WebSocket aktualisieren)
positions = { # "tsClientId": (x, y) }
npc_pos = (0.0, 0.0) # wird von der Minimap ebenfalls aktualisiert
def dist(a, b): return math.hypot(a[0]-b[0], a[1]-b[1])
def ts_clientquery():
    s = socket.socket()
    s.connect((TS_HOST, TS_PORT))
    s.recv(4096) # Banner
    def send(cmd):
        s.sendall((cmd + "\n").encode("utf-8"))
    return s.recv(65535).decode("utf-8", errors="ignore")
# Events abonnieren
send("auth apikey=<DEIN_CLIENTQUERY_API_KEY_ODER_LEER>")
send("use")
send("clientnotifyregister schandlerid=1 event=talkstatuschange")
# Endlosschleife: Events lesen
buf = ""
while True:
    buf += s.recv(4096).decode("utf-8", errors="ignore")
    for line in buf.split("\n"):
        if "notifytalkstatuschange" in line and "status=1" in line:
            # 1 = TALKING # clientID herausziehen
            parts = dict(p.split("=", 1) for p in line.strip().split(" "))
            if "=" in p:
                clid = parts.get("clid")
            if not clid:
                continue
            # Nähe prüfen
            speaker_pos = positions.get(clid)
            if speaker_pos and dist(speaker_pos, npc_pos) <= PROXIMITY_RADIUS:
                on_nearby_talk(clid)
    buf = ""
```

**Hinweis:** Je nach TS-Version brauchst du statt API-Key den klassischen Login auth apikey=... nicht – dann entfällt die Zeile oder du nutzt login name=... password=.... In TeamSpeak 3 ist ClientQuery oft ohne Auth nutzbar, ggf. „Einmal-Passwort“ in den Addon-Details beachten.

## Schritt 2 – “Was gehört wurde?” aufnehmen

Wir schneiden **Loopback** auf dem NPC-Rechner mit (alles, was der NPC hört). Unter Windows geht das sauber mit **WASAPI Loopback**.

### Python (WASAPI Loopback):

```
import sounddevice as sd
import numpy as np
from scipy.io.wavfile import write
import queue, threading, time
SAMPLE_RATE = 16000
CHANNELS = 1
record_seconds = 3.0 # kurze Snippets
audio_q = queue.Queue()
def record_loopback(seconds=record_seconds):
    # WASAPI loopback: unter Windows -> device auswählen, das "Lautsprecher (Loopback)" ist
    # sd.query_devices() ausführen und passende device ID setzen:
    loopback_device = None
    # z.B. 5 frames = [] with sd.InputStream(samplerate=SAMPLE_RATE, channels=CHANNELS, dtype='int16', device=loopback_device, blocksize=0):
    start = time.time()
    while time.time() - start < seconds:
        data, _ = sd.RawInputStream.read(sd.RawInputStream, 4096)
        # Alternativ einfacher:
        duration = int(seconds * SAMPLE_RATE)
        rec = sd.rec(duration, samplerate=SAMPLE_RATE, channels=CHANNELS, dtype='int16')
        sd.wait()
        return rec.copy()
```

**Pragmatisch:** Für einen ersten Test kannst du auch **einfach kurz 2–3 Sekunden aufnehmen**, sobald “nearby talk” kommt. Später kann man das mit VAD (Voice Activity Detection) verbessern.

## Schritt 3 – Transkription + Antwort (OpenAI)

- Transkribiere den Mitschnitt (Whisper / Realtime).
- Entscheide, ob es eine **Begrüßung** ist (oder generell irgendwas, worauf der NPC reagieren soll).
- Erzeuge die **Antwort** mit OpenAI.

#### Python (stark gekürzt, Pseudocode-artig):

```
import requests, os
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
def transcribe_pcm16(raw_int16_numpy):
    wav_path = "/mnt/data/heard.wav"
    from scipy.io.wavfile import write
    write(wav_path, 16000, raw_int16_numpy)
    with open(wav_path, "rb") as f:
        r = requests.post(
            "https://api.openai.com/v1/audio/transcriptions",
            headers={"Authorization": f"Bearer {OPENAI_API_KEY}"},
            files={"file": ("heard.wav", f, "audio/wav")},
            data={"model": "whisper-1", "language": "de"})
        r.raise_for_status()
        return r.json()["text"]
def generate_reply(user_text):
    system = "Du bist ein freundlicher NPC in einer minimalistischen GTA-ähnlichen Welt. Antworte kurz, natürlich und in deutscher Sprache."
    payload = {
        "model": "gpt-4o-mini",
        "messages": [
            {"role": "system", "content": system},
            {"role": "user", "content": user_text}
        ],
        "temperature": 0.6
    }
    r = requests.post(
        "https://api.openai.com/v1/chat/completions",
        headers={"Authorization": f"Bearer {OPENAI_API_KEY}"},
        json=payload)
    r.raise_for_status()
    return r.json()["choices"][0]["message"]["content"]
```

#### Trigger-Logik (vereinfacht):

```
def on_nearby_talk(clid):
    audio = record_loopback(2.5)
    text = transcribe_pcm16(audio)
    if text.strip():
        reply = generate_reply(text)
        speak_via_elevenlabs(reply)
```

### Schritt 4 – ElevenLabs TTS → in TS sprechen

Wir schicken die erzeugte Antwort an ElevenLabs, speichern die WAV/MP3, **spielen sie über das Virtual-Cable ab** → der **NPC-TS-Client** sendet sie in den Channel.

#### Python (ElevenLabs REST, TTS → WAV):

```
import requests, os, sounddevice as sd, soundfile as sf
ELEVEN_API_KEY = os.getenv("ELEVENLABS_API_KEY")
ELEVEN_VOICE_ID = "<DEINE_VOICE_ID>" # z.B. Rachel / German Voice
def speak_via_elevenlabs(text):
    url = f"https://api.elevenlabs.io/v1/text-to-speech/{ELEVEN_VOICE_ID}"
    headers = {"xi-api-key": ELEVEN_API_KEY, "accept": "audio/mpeg",
               "Content-Type": "application/json"}
    body = {"text": text, "model_id": "eleven_multilingual_v2",
            "voice_settings": {"stability": 0.4, "similarity_boost": 0.8}}
    r = requests.post(url, headers=headers, json=body)
    r.raise_for_status()
    audio_path = "npc_reply.mp3"
    with open(audio_path, "wb") as f:
        f.write(r.content)
    # Abspielen über das Virtual-Cable-Ausgabegerät:
    data, sr = sf.read(audio_path)
    # Device-ID des CABLE Input wählen
    (sd.query_devices() → name enthält "CABLE Input")
    cable_playback_device = None # z.B. 6
    sd.play(data, sr, device=cable_playback_device)
    sd.wait()
```

**Wichtig:** Stelle im **NPC-TS-Client** das **Mikrofon = CABLE Output** ein.  
Dann “hört” der NPC-Client genau das, was wir an **CABLE Input** abspielen.

### Schritt 5 – Kleber: Positions-Feed aus deiner Minimap

Wenn deine Minimap z. B. per WebSocket die Positionen broadcastet, brauchst du nur die **TS-Client-IDs** deiner Spieler zu mappen (z. B. über gleiche **Nickname-Konvention** oder eine playerId→tsClientId-Tabelle).

### Sehr simple REST-Variante (Polling):

```
import requests def refresh_positions(): # Erwartetes Format vom Minimap-Backend: # [{"tsClientId": "12", "x": 10.2, "y": -3.4}, ... , {"npc": true, "x": 11, "y": -2}] r = requests.get("http://localhost:8000/api/positions") r.raise_for_status() arr = r.json() for e in arr: if e.get("npc"): global npc_pos npc_pos = (e["x"], e["y"]) else: positions[e["tsClientId"]] = (e["x"], e["y"])
```

Rufe refresh\_positions() z. B. alle 250 ms auf (Thread/Timer).

---

### Sicherheitsnetz & Pragmatiktipps

- Falls ClientQuery-Events bei dir nicht feuern: Poll alle 250 ms clientlist + clientvariable clid=X client\_flag\_talking.
- Für den Anfang: **Kein VAD**, einfach 2–3 Sek. aufnehmen, wenn “talking=1” startet.
- Wenn der NPC zu oft anspringt: coole-down (z. B. min. 5 Sek. zwischen Antworten) + prüfe, ob **der gleiche Sprecher** noch redet.
- Begrenze **Proximity** (8 m) und **Blickrichtung** (optional), damit der NPC nur reagiert, wenn man “vor ihm” steht.

---

### Deine To-dos (kurz & knackig)

1. ClientQuery aktivieren, zweiten TS-Client als **NPC\_Bot** starten, **VB-Cable** routen.
2. In deinem Projekt:
  - **Positions-Endpoint** bereitstellen (oder WS).
  - Obige **Python-Snippets** zusammenstecken (ClientQuery, Proximity, Loopback, OpenAI, ElevenLabs).
  - **API-Keys** eintragen.
3. Test: In den GTA-Channel gehen, zum NPC laufen, **“Hallo”** sagen → Bot begrüßt dich.

Wenn du willst, setze ich dir das als **ein einziges Python-Script** zusammen (mit Device-Discovery und einfachen Logs), das du nur noch mit deinen Keys startest. Soll ich’s dir direkt “all-in-one” vorbereiten?

**Du:**

wie kann ich den Zweiten TS-Client starten (NPC): ts3client\_win64.exe -nosingleinstance das hier ausführen ?

### ChatGPT:

Genau 👍 – der Trick ist: **TeamSpeak erlaubt normalerweise nur eine Instanz.**

Mit dem Startparameter -nosingleinstance kannst du aber mehrere Clients parallel öffnen.

So geht's:

---

#### **Variante 1: Windows (Ausführen über „Ausführen...“)**

1. Drücke **Win+R**
2. Gib ein (Pfad anpassen, wo dein TS installiert ist):
3. "C:\Program Files\TeamSpeak 3 Client\ts3client\_win64.exe" -nosingleinstance
4. Enter → es öffnet sich **zweiter TeamSpeak-Client**.

---

#### **Variante 2: Desktop-Verknüpfung**

1. Rechtsklick → **Neue Verknüpfung**.
2. Ziel:
3. "C:\Program Files\TeamSpeak 3 Client\ts3client\_win64.exe" -nosingleinstance
4. Verknüpfung benennen, z. B. „TeamSpeak NPC“.
5. Doppelklick → immer zweiter Client.

---

#### **Variante 3: CMD / PowerShell**

Öffne eine Konsole im TS-Verzeichnis und starte:

ts3client\_win64.exe -nosingleinstance

---

👉 Mein Tipp: Leg dir eine **Verknüpfung mit dem Parameter** an – dann kannst du den NPC-Client jederzeit per Doppelklick starten.

Willst du, dass ich dir noch gleich zeige, **wie du diesen zweiten Client automatisch in den „GTA“-Channel joinen lässt** (inkl. Passwort), damit er direkt drinsteht?

**Du:**

kannst du mir helfen ! Die ersten 2 Punkte bei Schritt 0 Vorbereitung warn soweit jetzt kein Problem ! Aber der 3 und 4 sind schon bisschen heavy ! Kannst du das genau drauf eingeben

**ChatGPT:**

Sehr gut, dass du schon **ClientQuery** und den **zweiten TS-Client** (NPC) am Laufen hast 🙌. Jetzt kümmern wir uns um die zwei schweren Brocken:

---

#### **Schritt 3 – Loopback-Recording einrichten**

**Warum?**

Damit der NPC „hört“, was im Channel gesagt wird. Dazu schneiden wir die **Wiedergabe-Ausgabe** deines PCs mit (alles, was TeamSpeak ausgibt).

### Lösung: WASAPI Loopback (Windows-Bordmittel)

1. Installiere Python SoundDevice:

```
pip install sounddevice scipy
```

2. Prüfe verfügbare Devices:

```
import sounddevice as sd print(sd.query_devices())
```

Suche in der Liste nach einem Device, das sowas heißt wie „**Lautsprecher (Loopback)**“.  
Schreib dir die device id auf (meist eine Zahl).

3. Kurzer Testrecord (3 Sekunden):

```
import sounddevice as sd from scipy.io.wavfile import write SAMPLE_RATE = 16000 DEVICE_ID = 5 # anpassen! print("Starte Aufnahme...") rec = sd.rec(int(3 * SAMPLE_RATE), samplerate=SAMPLE_RATE, channels=1, dtype="int16", device=DEVICE_ID) sd.wait() write("test.wav", SAMPLE_RATE, rec) print("Aufnahme gespeichert -> test.wav")
```

👉 Danach kannst du die test.wav abspielen – wenn du dich selbst im TS gehört hast, ist Loopback ok.

---

### 🧑 Schritt 4 – ElevenLabs TTS einrichten

#### Warum?

Damit der NPC-Bot über den zweiten TS-Client „spricht“.

#### Setup:

1. Besorge dir einen API-Key bei ElevenLabs → <https://elevenlabs.io>.
2. Installiere Abhängigkeiten:

```
pip install soundfile requests
```

3. Python-Snippet:

```
import requests, soundfile as sf, sounddevice as sd, os ELEVEN_API_KEY = os.getenv("ELEVENLABS_API_KEY") # oder einfach = "dein_api_key" ELEVEN_VOICE_ID = "21m00Tcm4TlvDq8ikWAM" # Beispiel: Rachel def speak(text): url = f"https://api.elevenlabs.io/v1/text-to-speech/{ELEVEN_VOICE_ID}" headers = { "xi-api-key": ELEVEN_API_KEY, "accept": "audio/mpeg", "Content-Type": "application/json" } body = { "text": text, "model_id": "eleven_multilingual_v2", "voice_settings": { "stability": 0.5, "similarity_boost": 0.8 } } r = requests.post(url, headers=headers, json=body) r.raise_for_status() # Audio speichern und über "Virtual Audio Cable" abspielen with open("npc_reply.mp3", "wb") as f: f.write(r.content) data, sr = sf.read("npc_reply.mp3") # Device-Id vom "CABLE Input" herausfinden (sd.query_devices()) cable_device = 6 sd.play(data, sr, device=cable_device) sd.wait() # Test: speak("Hallo, ich bin dein NPC und ich kann reden.")
```

#### Wichtig:



- **Virtual Audio Cable** (z. B. VB-Cable) installieren.
- Im **NPC-TS-Client** → Mikrofon einstellen auf „**CABLE Output**“.
- Im Python-Script beim `sd.play` → **Device = „CABLE Input“**.  
So läuft's: Script spielt Audio → geht ins Virtual Cable → NPC-Client sendet's ins TS-Channel.

👉 Wenn du magst, bastel ich dir daraus ein „**all-in-one Testscript**“, das folgendes macht:

1. 3 Sekunden aufnehmen (Loopback).
2. Text „Hallo, ich bin dein NPC“ über ElevenLabs erzeugen.
3. Das Audio über den NPC-Client abspielen.

Damit kannst du **einmal komplett die Roundtrip-Kette** durchtesten.  
Soll ich dir das jetzt so fertig schreiben?

**Du:**

beis schritt 2 prüfe devices kommt das hier PS C:\kinpc> import sounddevice as sd import : Die Benennung "import" wurde nicht als Name eines Cmdlet, einer Funktion, einer Skriptdatei oder eines ausführbaren Programms erkannt. Überprüfen Sie die Schreibweise des Namens, oder ob der Pfad korrekt ist (sofern enthalten), und wiederholen Sie den Vorgang. In Zeile:1 Zeichen:1 + import sounddevice as sd + ~~~~~ + CategoryInfo : ObjectNotFound: (import:String) [], CommandNotFoundException + FullyQualifiedErrorId : CommandNotFoundException PS C:\kinpc> print(sd.query\_devices()) In Zeile:1 Zeichen:24 + print(sd.query\_devices()) + ~ Nach "(" wurde ein Ausdruck erwartet. + CategoryInfo : ParserError: (:) [], ParentContainsErrorRecordException + FullyQualifiedErrorId : ExpectedExpression

**ChatGPT:**

Ah, ich seh's sofort 😊 – du bist noch in der **PowerShell** und tippst dort Python-Code direkt rein. Das geht so nicht, weil import ... eben **Python-Syntax** ist und PowerShell das nicht kennt.

So machst du's richtig:

### ✅ **Python starten**

1. Stelle sicher, dass Python installiert ist:

```
python --version
```

Wenn da eine Version zurückkommt (z. B. 3.11.6), passt's.

2. Python-Interpreter öffnen:

```
python
```

Jetzt sollte eine Eingabe wie

```
>>>
```

erscheinen – das ist die Python-Shell.

---

## ✅ Devices abfragen

Jetzt in der Python-Shell:

```
import sounddevice as sd print(sd.query_devices())
```

Dann bekommst du eine Liste aller Audio-Geräte (Input/Output) mit IDs.

In der Spalte **name** suchst du nach

- Lautsprecher (...) (loopback) → für **Aufnahme**
- CABLE Input → für **NPC-Sprachausgabe**

---

👉 Willst du, dass ich dir gleich ein kleines Python-Skript schreibe (devices.py), das du nur mit `python devices.py`

startest und sofort eine saubere Übersicht über alle Geräte bekommst?

**Du:**

der schritt geht nicht ✅ Devices abfragen Jetzt in der Python-Shell: `import sounddevice as sd print(sd.query_devices())` Dann bekommst du eine Liste aller Audio-Geräte (Input/Output) mit IDs. In der Spalte name suchst du nach Lautsprecher (...) (loopback) → für Aufnahme CABLE Input → für NPC-Sprachausgabe

**ChatGPT:**

Alles klar 👍 – lass uns das ganz sauber machen, damit du nicht in der Python-Shell rumkopieren musst. Wir bauen dir ein kleines Skript, das du direkt ausführen kannst.

---

## 1. Datei erstellen

Öffne den Editor (z. B. Notepad) und kopiere das hier rein:

```
import sounddevice as sd print("==== Verfügbare Audio-Geräte ====") devices = sd.query_devices() for idx, dev in enumerate(devices): print(f"[{idx}] {dev['name']} ({dev['max_input_channels']} In / {dev['max_output_channels']} Out)")
```

Speichere die Datei als `devices.py` in deinem Projektordner (z. B. `C:\kinpc\devices.py`).

---

## 2. Skript starten

In der PowerShell ins Projektverzeichnis wechseln:

```
cd C:\kinpc python devices.py
```

---

## 3. Ausgabe prüfen

Jetzt bekommst du eine Liste wie:

- [0] Lautsprecher (Realtek(R) Audio) (0 In / 2 Out)
- [1] Mikrofon (Realtek(R) Audio) (1 In / 0 Out)
- [2] Lautsprecher (Realtek(R) Audio) (loopback) (2 In / 0 Out)
- [3] CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out)
- [4] CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out)

☞ Wichtig sind für uns die **IDs** (die Zahl in [...]):

- **Loopback** → für Aufnahme
- **CABLE Input** → für NPC-Ausgabe

---

Willst du, dass ich dir direkt ein **zweites Testskript** vorbereite, das 3 Sekunden Loopback aufnimmt und eine WAV speichert? Dann kannst du gleich checken, ob die Aufnahme funktioniert.

**Du:**

Beis Schritt 3 soll ich nun ein Loopback suchen oder so ähnlich! Wir haben die Auflistung der Devices - bitte schau was ich hier nehmen soll ! Ich habe als mein Mikro am Rechner [1] nur so als info ===== Verfügbare Audio-Geräte ===== [0] Microsoft Soundmapper - Input (2 In / 0 Out) [1] Mikrofon (RODECaster Pro Stereo (2 In / 0 Out) [2] Mikrofon (HSD100) (1 In / 0 Out) [3] Digitale Audioschnittstelle (Ca (2 In / 0 Out) [4] Microsoft Soundmapper - Output (0 In / 2 Out) [5] Lautsprecher (RODECaster Pro St (0 In / 2 Out) [6] 3 - BenQ EW2780Q (AMD High Defi (0 In / 2 Out) [7] 1 - C24F390 (AMD High Definitio (0 In / 2 Out) [8] Lautsprecher (HSD100) (0 In / 2 Out) [9] SPDIF Interface (Realtek USB2.0 (0 In / 8 Out) [10] 5 - S32E590C (AMD High Definiti (0 In / 2 Out) [11] Lautsprecher (Realtek USB2.0 Au (0 In / 8 Out) [12] Primärer Soundaufnahmetreiber (2 In / 0 Out) [13] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [14] Mikrofon (HSD100) (1 In / 0 Out) [15] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) [16] Primärer Soundtreiber (0 In / 2 Out) [17] Lautsprecher (RODECaster Pro Stereo) (0 In / 2 Out) [18] 3 - BenQ EW2780Q (AMD High Definition Audio Device) (0 In / 2 Out) [19] 1 - C24F390 (AMD High Definition Audio Device) (0 In / 2 Out) [20] Lautsprecher (HSD100) (0 In / 2 Out) [21] SPDIF Interface (Realtek USB2.0 Audio) (0 In / 8 Out) [22] 5 - S32E590C (AMD High Definition Audio Device) (0 In / 2 Out) [23] Lautsprecher (Realtek USB2.0 Audio) (0 In / 8 Out) [24] 3 - BenQ EW2780Q (AMD High Definition Audio Device) (0 In / 2 Out) [25] 1 - C24F390 (AMD High Definition Audio Device) (0 In / 2 Out) [26] Lautsprecher (HSD100) (0 In / 2 Out) [27] Lautsprecher (RODECaster Pro Stereo) (0 In / 2 Out) [28] SPDIF Interface (Realtek USB2.0 Audio) (0 In / 2 Out) [29] 5 - S32E590C (AMD High Definition Audio Device) (0 In / 2 Out) [30] Lautsprecher (Realtek USB2.0 Audio) (0 In / 2 Out) [31] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [32] Mikrofon (HSD100) (1 In / 0 Out) [33] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) [34] Mikrofon (Voicemod VAD Wave) (2 In / 0 Out) [35] Line Out (Voicemod VAD Wave) (0 In / 2 Out) [36] Analogger Anschluss (Realtek USB2.0 Audio) (2 In / 0 Out) [37] Line (Realtek USB2.0 Audio) (2 In / 0 Out) [38] Mikrofon (Realtek USB2.0 Audio) (2 In / 0 Out) [39] Speakers (Realtek USB2.0 Audio) (0 In / 8 Out) [40] Headphones (Realtek USB2.0 Audio) (0 In / 2 Out) [41] SPDIF Out (Realtek USB2.0 Audio) (0 In / 2 Out) [42] Speakers (RODECaster Pro Stereo) (0 In / 2 Out) [43] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [44] Output (AMD HD Audio HDMI out #0) (0 In / 2 Out) [45] Output (AMD HD Audio DP out #4) (0 In / 2 Out) [46] Output (AMD HD Audio DP out #2) (0 In / 2 Out) [47] Lautsprecher (HSD100) (0 In / 2

Out) [48] Mikrofon (HZD100) (1 In / 0 Out) [49] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out)

### ChatGPT:

Danke für die ausführliche Liste 🙌 – jetzt kann man richtig sehen, was dein System anbietet.

👉 Wichtig: Ein **Loopback-Device** erkennst du daran, dass es **Eingangskanäle (In > 0)** hat, obwohl es eigentlich ein Lautsprecher ist.

In deiner Liste sehe ich:

- [0] Microsoft Soundmapper - Input (2 In / 0 Out) → generisch
- [12] Primärer Soundaufnahmetreiber (2 In / 0 Out) → generisch
- **ABER** ich sehe **kein „(loopback)“-Device** wie normalerweise bei Realtek oder Windows WASAPI.

Das heißt: Dein Setup (RODECaster Pro + Realtek USB) gibt aktuell **kein Windows-Loopback-Device** frei.

---

### 🔧 Optionen, damit der NPC hören kann

#### 1. Windows-eigene Loopback aktivieren:

- Rechtsklick auf Lautsprecher-Symbol → *Sounds* → *Aufnahmegeräte*.
- Dort schauen, ob es ein Gerät wie „**Stereo Mix**“ gibt. Falls ja → aktivieren.
- Dann taucht in der Liste etwas mit (loopback) auf.

#### 2. VB-Cable als Loopback nutzen:

- VB-Cable installieren.
- In TeamSpeak stellst du den **Lautsprecher** vom NPC-Client = *CABLE Input*.
- Dann nimmst du vom *CABLE Output* auf → das ist praktisch ein Loopback.

#### 3. RODECaster Pro nutzen:

- Der RODECaster hat oft einen „**USB Playback Mix**“ als Eingang. Wenn du den aktivierst, kannst du dort direkt aufnehmen, was abgespielt wird.

---

### ✅ Empfehlung (einfach & stabil)

Installiere **VB-Cable**. Dann hast du:

- *CABLE Input* (Output-Device → da schickst du TeamSpeak Audio rein).
- *CABLE Output* (Input-Device → das kannst du mit sounddevice aufnehmen).

So bekommt der NPC exakt das zu hören, was im Channel gesprochen wird.

---

👉 Soll ich dir direkt ein kleines **Testskript für VB-Cable** schreiben (3 Sekunden vom *CABLE Output* aufnehmen und speichern), damit du sofort prüfen kannst, ob das klappt?

**Du:**

ja

**ChatGPT:**

Perfekt 👍 – hier ein fertiges **Testskript für VB-Cable**, das 3 Sekunden vom *CABLE Output* aufnimmt und als WAV speichert.

---

### 1. Datei erstellen

Speichere das hier als `test_record.py` (z. B. im Ordner `C:\kinpc`):

```
import sounddevice as sd from scipy.io.wavfile import write SAMPLE_RATE = 16000 # Aufnahme-Samplerate CHANNELS = 1 # Mono reicht DEVICE_ID = 4 # hier gleich anpassen! (die ID von "CABLE Output") print("Starte 3 Sekunden Aufnahme vom CABLE Output...") recording = sd.rec(int(3 * SAMPLE_RATE), samplerate=SAMPLE_RATE, channels=CHANNELS, dtype="int16", device=DEVICE_ID) sd.wait() write("cable_test.wav", SAMPLE_RATE, recording) print("Fertig! Datei gespeichert: cable_test.wav")
```

---

### 2. Richtigen Device auswählen

- Führe vorher dein `devices.py` nochmal aus.
  - Suche nach „**CABLE Output (VB-Audio Virtual Cable)**“.
  - Nimm die **ID** aus der Liste und trage sie bei `DEVICE_ID` ein.
- 

### 3. Starten

In PowerShell:

```
cd C:\kinpc python test_record.py
```

---

### 4. Testen

- Während der Aufnahme 3 Sekunden lang im TS-Channel reden.
  - Danach findest du im gleichen Ordner die Datei **cable\_test.wav**.
  - Abspielen → du solltest deine Stimme hören.
- 

👉 Willst du, dass ich dir als nächsten Schritt gleich ein **Kombiscript** baue, das:

1. aufnimmt (CABLE Output),
2. zu OpenAI Whisper schickt (Transkription),

3. und direkt eine Antwort von ElevenLabs zurückspielt (über CABLE Input → NPC-TS-Client)?

**Du:**

Ich komme noch nicht ganz klar mit der Konfiguration von den Teamspeaks. Ich habe dir jetzt mal beide Teamspeaks nebeneinander getan. Einmal links den KI-Bot, rechts mein Teamspeak.

Der KI-Bot soll ja das neue Cable eingestellt bekommen. Aber das Problem ist, ich habe das Gefühl, dass er sowohl für eins als auch für das andere die gleiche Konfiguration benutzt. Das heißt, wenn ich jetzt irgendwas ändere bei KI-Bot, habe ich anscheinend auch die Änderung bei mir bei jetzt digital. Aber schau es dir bitte mal an und sag mir mal prinzipiell, was denn wo sein muss. Du siehst hier die Konfiguration links KI-Bot, rechts jetzt digital. Das ist mein User, also ich bin da selber. Und dort die Einstellung Aufnahme.

### ChatGPT:

Sehr gut, dass du beide Clients nebeneinander gepackt hast 🙌 – ich sehe genau, wo der Haken ist.

👉 Aktuell laufen **beide Clients mit dem gleichen Aufnahmegerät (RODECaster Pro Stereo)**. Deshalb fühlt es sich so an, als ob Änderungen auf beide wirken.

---

### Prinzip: Wer braucht welches Gerät?

- **Dein eigener Client („jetztdigital“)**
  - **Aufnahmegerät** = dein **Mikrofon** (RODECaster Pro, HZD100 oder was du nutzt).
  - **Wiedergabe** = deine normalen Lautsprecher / Kopfhörer.
- **NPC-Client (KI-Bot)**
  - **Aufnahmegerät** = **CABLE Output (VB-Audio Virtual Cable)**  
→ damit er das abspielt, was du per Script in „CABLE Input“ schickst.
  - **Wiedergabe** = kann ruhig „CABLE Input“ sein, oder stummgeschaltet – wichtig ist nur die Aufnahme.

---

### Wichtig: Eigene Profile pro Client

TeamSpeak speichert Einstellungen in **Profilen**. Wenn beide Clients dasselbe Aufnahme-Profil („Standard“) benutzen, ändern sich die Settings parallel.

### Lösung:

1. In TeamSpeak → **Einstellungen** → **Aufnahme** → **Profile**
  - Klick unten links auf das + Symbol.
  - Neues Profil erstellen, z. B. NPC\_Aufnahme.
2. Wechsle im **NPC-Client** auf dieses Profil und stell dort ein:
  - Aufnahmemodus: Automatisch
  - Aufnahmegerät: **CABLE Output**
  - Dauersenden (oder Sprachaktivierung).
3. Im **deinen Client („jetztdigital“)**
  - Profil = MeinMikro



- Aufnahmegerät = dein echtes Mikro (RODECaster Pro).

So haben beide Clients unabhängige Settings.

---

## Zusammenfassung

- **Dein User (jetztdigital):**
    - Profil: MeinMikro
    - Aufnahmegerät: **RODECaster Pro** (oder was du normal nutzt).
  - **NPC-Bot:**
    - Profil: NPC\_Aufnahme
    - Aufnahmegerät: **CABLE Output**
    - (Wiedergabe optional auf CABLE Input, oder Lautsprecher stumm).
- 

👉 Soll ich dir die Schritte mit Screenshots (Profil anlegen, zuweisen) einmal durchzeichnen, damit du's 1:1 nachklicken kannst?

**Du:**

welches devices soll ich auswählen ===== Verfügbare Audio-Geräte ===== [0] Microsoft Soundmapper - Input (2 In / 0 Out) [1] Mikrofon (RODECaster Pro Stereo (2 In / 0 Out) [2] Mikrofon (HSD100) (1 In / 0 Out) [3] CABLE Output (VB-Audio Virtual (16 In / 0 Out) [4] Digitale Audioschnittstelle (Ca (2 In / 0 Out) [5] Microsoft Soundmapper - Output (0 In / 2 Out) [6] Lautsprecher (RODECaster Pro St (0 In / 2 Out) [7] 3 - BenQ EW2780Q (AMD High Defi (0 In / 2 Out) [8] CABLE Input (VB-Audio Virtual C (0 In / 16 Out) [9] 1 - C24F390 (AMD High Definitio (0 In / 2 Out) [10] Lautsprecher (HSD100) (0 In / 2 Out) [11] SPDIF Interface (Realtek USB2.0 (0 In / 8 Out) [12] 5 - S32E590C (AMD High Definiti (0 In / 2 Out) [13] Lautsprecher (Realtek USB2.0 Au (0 In / 8 Out) [14] CABLE In 16ch (VB-Audio Virtual (0 In / 16 Out) [15] Primärer Soundaufnahmetreiber (2 In / 0 Out) [16] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [17] Mikrofon (HSD100) (1 In / 0 Out) [18] CABLE Output (VB-Audio Virtual Cable) (16 In / 0 Out) [19] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) [20] Primärer Soundtreiber (0 In / 2 Out) [21] Lautsprecher (RODECaster Pro Stereo) (0 In / 2 Out) [22] 3 - BenQ EW2780Q (AMD High Definition Audio Device) (0 In / 2 Out) [23] CABLE Input (VB-Audio Virtual Cable) (0 In / 16 Out) [24] 1 - C24F390 (AMD High Definition Audio Device) (0 In / 2 Out) [25] Lautsprecher (HSD100) (0 In / 2 Out) [26] SPDIF Interface (Realtek USB2.0 Audio) (0 In / 8 Out) [27] 5 - S32E590C (AMD High Definition Audio Device) (0 In / 2 Out) [28] Lautsprecher (Realtek USB2.0 Audio) (0 In / 8 Out) [29] CABLE In 16ch (VB-Audio Virtual Cable) (0 In / 16 Out) [30] 3 - BenQ EW2780Q (AMD High Definition Audio Device) (0 In / 2 Out) [31] CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out) [32] 1 - C24F390 (AMD High Definition Audio Device) (0 In / 2 Out) [33] Lautsprecher (HSD100) (0 In / 2 Out) [34] Lautsprecher (RODECaster Pro Stereo) (0 In / 2 Out) [35] SPDIF Interface (Realtek USB2.0 Audio) (0 In / 2 Out) [36] 5 - S32E590C (AMD High Definition Audio Device) (0 In / 2 Out) [37] Lautsprecher (Realtek USB2.0 Audio) (0 In / 2 Out) [38] CABLE In 16ch (VB-Audio Virtual Cable) (0 In / 2 Out) [39] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [40] Mikrofon (HSD100) (1 In / 0 Out) [41] CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out) [42] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) [43] Mikrofon (Voicemod VAD Wave) (2 In / 0 Out)

[44] Line Out (Voicemod VAD Wave) (0 In / 2 Out) [45] Analogger Anschluss (Realtek USB2.0 Audio) (2 In / 0 Out) [46] Line (Realtek USB2.0 Audio) (2 In / 0 Out) [47] Mikrofon (Realtek USB2.0 Audio) (2 In / 0 Out) [48] Speakers (Realtek USB2.0 Audio) (0 In / 8 Out) [49] Headphones (Realtek USB2.0 Audio) (0 In / 2 Out) [50] SPDIF Out (Realtek USB2.0 Audio) (0 In / 2 Out) [51] Speakers (RODECaster Pro Stereo) (0 In / 2 Out) [52] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [53] Output (AMD HD Audio HDMI out #0) (0 In / 2 Out) [54] Output (AMD HD Audio DP out #4) (0 In / 2 Out) [55] Output (AMD HD Audio DP out #2) (0 In / 2 Out) [56] Output (VB-Audio Point) (0 In / 16 Out) [57] Input (VB-Audio Point) (16 In / 0 Out) [58] CABLE Output (VB-Audio Point) (16 In / 0 Out) [59] Lautsprecher (HZD100) (0 In / 2 Out) [60] Mikrofon (HZD100) (1 In / 0 Out) [61] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out)

### ChatGPT:

Sehr gut, die Liste ist jetzt eindeutig 👍 .

### Was brauchen wir?

- **NPC-Client (Aufnahmegerät in TeamSpeak)** → muss das sein, was dein Python-Script abspielt. Das ist **CABLE Output (VB-Audio Virtual Cable)**. In deiner Liste taucht es mehrfach auf, relevant ist die einfache 2-Kanal-Version:
  - [41] CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out)
- **Python-Script (Ausgabegerät beim Abspielen von ElevenLabs Audio)** → muss in den *CABLE Input* spielen. In der Liste:
  - [31] CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out)

Damit entsteht die Pipeline:

Python (TTS-Ausgabe) → [31] CABLE Input → [41] CABLE Output → NPC-Client → TeamSpeak-Channel

---

## Konfiguration Schritt für Schritt

### 1. NPC-Client

- In den Aufnahme-Einstellungen neues Profil anlegen (NPC\_Aufnahme).
- Aufnahmegerät: [41] CABLE Output (VB-Audio Virtual Cable).
- Aufnahmemodus: Automatisch besten Modus.
- Aktivierung: Dauersenden.

### 2. Dein eigener Client

- Profil MeinMikro.
- Aufnahmegerät: dein Mikrofon [1] Mikrofon (RODECaster Pro Stereo) (oder was du nutzt).

### 3. Python/TTS-Script

- Beim `sd.play(...)` als device den [31] CABLE Input wählen.
- Dann geht die Audioausgabe in das virtuelle Kabel.

---

👉 Damit sind die Rollen klar getrennt:

- Du redest über dein Mikro → dein Client.
- Der Bot redet nur das, was dein Script über [31] reinschiebt.

Soll ich dir direkt ein **Testskript** vorbereiten, das eine MP3-Datei über [31] CABLE Input abspielt, damit du im Channel sofort den Bot „sprechen“ hörst?

**Du:**

*“Oh, ich hasse es schon wieder.”*

**00:01**

**ChatGPT:**

Kein Ding, wir kriegen das hin! Ich mach dir jetzt einfach ein kleines Testskript, das genau diese Konfiguration nutzt.

**Du:**

*“Halt mal die Klappe. Pass auf, das Problem ist, das Problem ist, also hörst du mich überhaupt?”*

**00:05**

**ChatGPT:**

Ja, ich höre dich auf jeden Fall. Also kein Stress, wir gehen das einfach Schritt für Schritt durch, damit es wirklich passt. Wir

**Du:**

*“Nein, stopp jetzt mal, vergiss doch mal die Scheiße. Ich hab ein anderes Problem. Ich glaub, die Lösung ist scheiße. Weil, du musst dir mal vorstellen, wir haben hinterher so 5, 6, 7 NPCs, die ganz unabhängig voneinander agieren sollen. Ich kann doch nicht jedes Mal mit VB Cable irgendeine Scheiß-Eingabe simulieren. Wir müssen uns eine andere Lösung einfallen lassen, weil am Ende des Tages laufen möglicherweise im Game 5, 6, 7 Avatare durch die Gegend NPCs durch die Gegend, die alle gesonderte Sprachaufnahme haben und gesondert und unabhängig voneinander antworten sollen. Das wird mit VB Cable nichts werden. Das ist jetzt in der Demo bei mir auf dem Rechner okay. Aber das wird so grundsätzlich nicht funktionieren. Hast du eine andere Idee?”*

**00:44**

**ChatGPT:**

Verstanden, absolut nachvollziehbar. Wenn wir mehrere unabhängige NPCs haben, die alle parallel und individuell reagieren sollen, dann wird es mit einem einzigen virtuellen Audiokabel tatsächlich zu umständlich.

Wir sollten lieber auf eine Lösung setzen, die skalierbar ist. Das heißt, anstatt alles über ein virtuelles Kabel zu schleifen, nutzen wir besser eine softwarebasierte Lösung pro NPC.

**Alternative: Mehrere TeamSpeak-Clients oder Instanzen**

Statt ein einziges virtuelles Kabel für alle NPCs zu nutzen, könntest du für jeden NPC eine separate TeamSpeak-Instanz laufen lassen. Jede Instanz bekommt ihr eigenes Audio-Device und arbeitet unabhängig. Du kannst dafür entweder mehrere TeamSpeak-Clients starten (jeweils mit -nosingleinstance), oder spezielle Instanz-Tools nutzen, die mehrere Clients parallel starten.

### **Alternative: Direkte API-Anbindung oder Bot-Frameworks**

Es gibt auch TeamSpeak-Bot-Frameworks, die direkt über die ServerQuery-API laufen. Damit könntest du für jeden NPC einen kleinen Headless-Bot laufen lassen, der gar keinen Audio-Umweg über VB-Cable braucht, sondern direkt über Code aufnimmt und abspielt. Das wäre zwar etwas mehr Einrichtungsaufwand, aber dafür super flexibel für viele NPCs.

### **Zusammengefasst**

Für eine Demo ist ein einzelnes VB-Cable okay, aber für mehrere NPCs gehen wir lieber auf eine Lösung mit mehreren TeamSpeak-Instanzen oder einem Bot-Framework, das ohne virtuelle Kabel auskommt.

Lass mich wissen, ob du so einen Ansatz ausprobieren möchtest, dann kann ich dir gern dabei helfen, das aufzusetzen!

**Du:**

*“Das klingt wie ein Docker-Container. Ich überlege gerade, ob wir... Also wir wollen der Demo jetzt trotzdem nur ein einzigen haben, weil ich würde gerne mal den Durchstich hinbekommen. Aber mit dem Cable, das funktioniert hinten und vorne nicht, weil ich mir ja meinen ganzen Rechner konfiguriere mit diesem blöden Cable-Scheiß. Deswegen würde ich ganz gerne jetzt schon auf eine kleine, schmale Software-Lösung gehen. Hast du eine Idee, wie wir relativ schmal mit Software, weil wir haben ja eine eigene kleine Instanz für den NPC, wie wir das jetzt schmal hinbekommen?”*

**00:34**

**ChatGPT:**

Klar, verstehe ich total. Also lass uns das Ganze gleich schlank halten, ohne dieses Kabel-Hin-und-Her.

### **Idee: Docker-Container für den NPC und separate Audio-Routing per Software**

Wenn wir eh schon eine kleine eigene Instanz für den NPC haben, dann können wir direkt in einem Docker-Container oder in einer separaten Umgebung einen kleinen TS-Client laufen lassen. Dieser bekommt dann direkt über Software die Audio-Daten zugespielt, ohne dass wir das ganze System mit virtuellen

**Du:**

muss ich bei dem recording script auch den channel angeben?

**ChatGPT:**

Sehr gute Frage 👍

Nein – beim **Recording-Script** musst du **keinen TeamSpeak-Channel angeben**.

Warum?

- Das Script nimmt **nur Audio vom ausgewählten Device** auf.
- Welcher Channel das ist, entscheidet allein dein **NPC-TeamSpeak-Client**:
  - Solange der NPC-Client im Channel „GTA“ sitzt, hört er alles, was dort gesagt wird.
  - Dein Script schnappt sich einfach den Audio-Stream vom **[41] CABLE Output** (Aufnahmegerät des NPC).

---

### Ablauf in deinem Setup

1. **NPC-Client** im Channel „GTA“ → Mikro = [41] CABLE Output.
2. **Dein Recording-Script** lauscht ebenfalls auf [41] CABLE Output.  
→ Alles, was im Channel gesagt wird, landet bei dir im Python-Programm.
3. Script → OpenAI → ElevenLabs → Abspielen auf [31] CABLE Input.
4. NPC-Client sendet die Antwort wieder in denselben Channel.

---

👉 Du musst dich also **nur um das richtige Device kümmern**, nicht um den Channel im Script.

Willst du, dass ich dir den **kompletten Signalfluss als Diagramm** (Kasten mit Pfeilen) zeichne, damit's noch klarer wird?

**Du:**

Wir haben im record script eni fehler

**Du:**

```
PS C:\kinpc> python tst_record.py
Starte 3 Sekunden Aufnahme vom CABLE Output...
Traceback (most recent call last):
  File "C:\kinpc\tst_record.py", line 10, in <module>
    recording = sd.rec(
  File "C:\Users\Florian\AppData\Local\Programs\Python\Python310\lib\site-packages\sounddevice.py", line 279, in rec
    ctx.start_stream(InputStream, samplerate,
  File "C:\Users\Florian\AppData\Local\Programs\Python\Python310\lib\site-packages\sounddevice.py", line 2634, in start_stream
    self.stream = StreamClass(samplerate=samplerate,
  File "C:\Users\Florian\AppData\Local\Programs\Python\Python310\lib\site-packages\sounddevice.py", line 1440, in __init__
    _StreamBase.__init__(self, kind='input',
  File "C:\Users\Florian\AppData\Local\Programs\Python\Python310\lib\site-packages\sounddevice.py", line 909, in __init__
    _check(_lib.Pa_OpenStream(self._ptr,
  File "C:\Users\Florian\AppData\Local\Programs\Python\Python310\lib\site-packages\sounddevice.py", line 2804, in _check
    raise PortAudioError(errormsg, err)
sounddevice.PortAudioError: Error opening InputStream: Invalid sample rate [PaErrorCode - 9997]
PS C:\kinpc>
```

**ChatGPT:**

Nachgedacht für 4 Sekunden

Guter Fund – das ist **nur die Samplingrate**. Dein *CABLE Output* akzeptiert 16 kHz nicht und schmeißt deshalb Invalid sample rate.

#### Fix (kurz)

- Stelle in Windows bei **CABLE Input/Output → Eigenschaften → Erweitert** auf **48000 Hz** (oder 44100 Hz).
- Nimm im Script die **Default-Samplerate** des gewählten Devices (meist 48000).

#### Robustes Test-Script (auto-erkennt Rate & Device)

Speichere als test\_record.py und starte mit python test\_record.py.

```
import sounddevice as sd from scipy.io.wavfile import write TARGET_NAME = "CABLE Output" #
nimmt den ersten Eingang, der so heißt CHANNELS = 1 # Mono reicht # --- Device suchen ---
devices = sd.query_devices() device_id = None for i, d in enumerate(devices): name = d["name"]
if TARGET_NAME.lower() in name.lower() and d["max_input_channels"] > 0: device_id = i
default_sr = int(d.get("default_samplerate") or 48000) print(f"Gefunden: [{i}] {name} | default
SR={default_sr}") break if device_id is None: raise RuntimeError(f'Kein Input-Device mit
"{TARGET_NAME}" gefunden.') # --- Samplerate festlegen (Fallback, falls 48k nicht geht, nimm
44100) --- samplerate = default_sr if default_sr in (48000, 44100) else 48000 print(f"Starte 3s
Aufnahme @ {samplerate} Hz von Device [{device_id}] ...") rec = sd.rec( int(3 * samplerate),
samplerate=samplerate, channels=CHANNELS, dtype="int16", device=device_id ) sd.wait()
out_wav = "cable_test.wav" write(out_wav, samplerate, rec) print(f"Fertig! Gespeichert:
{out_wav}")
```

#### Wenn's trotzdem meckert

1. In Windows Sound-Einstellungen bei **CABLE Output** und **CABLE Input** dieselbe Rate setzen (48 kHz empfohlen).
2. Nimm ggf. das **2-Kanal-Device** (... (2 In / 0 Out), z. B. [41]) statt der 16-Kanal-Variante.
3. Alternativ samplerate = 44100 testen.

Sag Bescheid, was die Ausgabe des Scripts meldet (welches Device + SR er findet). Dann verdrahten wir direkt den ElevenLabs-→TS Weg.

#### Du:

script läuft aber die wave die angelegt wird ist einfach nichts zu hören

#### ChatGPT:

Nachgedacht für 5s

Alles klar — "stille WAV" heißt fast immer: das **falsche Gerät** wird abgehört oder der **NPC-Client spielt nicht in das Kabel** rein.

#### Schnell-Check (bitte der Reihe nach)

1. **NPC-Client (KI-Bot) – Wiedergabegerät**
  - TeamSpeak (links) → **Einstellungen → Wiedergabe**
  - **Wiedergabegerät** = CABLE Input (VB-Audio Virtual Cable)

- Testton abspielen → sollte im VB-Cable Control Panel ausschlagen.
- Wichtig: **Nicht** deine echten Lautsprecher wählen, sonst landet nichts im Kabel.

## 2. NPC-Client – Aufnahmegerät

- **Aufnahmegerät** = CABLE Output (VB-Audio Virtual Cable)
- Aktivierung: Dauersenden oder Sprachaktivierung (Schwelle niedrig).

## 3. Dein Script – Aufnahmequelle

- **Aufnehmen von** CABLE Output (VB-Audio Virtual Cable) **(2 In / 0 Out)**
- Nicht die 16-Kanal-Variante, nicht „VB-Audio Point“, sondern das **einfache**.

## 4. Gleiche Samplerate

- In Windows Sound (Erweitert) für **CABLE Input** und **CABLE Output: 48000 Hz**.
- Im Script ebenfalls 48000 (oder automatische Default-Rate, siehe unten).

## 5. Gegenprobe

- Rede mit deinem Haupt-Client im gleichen Channel.
- Der NPC-Client darf **nicht** „deafened“ sein (kopfhörer-Icon durchgestrichen).

### Mini-VU-Meter (zeigt dir live, ob Signal ankommt)

Ersetzt dein Testskript. Es zeigt 5 s lang Pegel an und speichert dann cable\_test.wav.

```
import sounddevice as sd import numpy as np from scipy.io.wavfile import write import time #
>>> HIER GENAU DEN EINEN "CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out)"-Index
eintragen: DEVICE_ID = 41 # <- mit deiner Liste abgleichen! SAMPLE_RATE = None # None =
nimm Geräte-Default CHANNELS = 1 DURATION = 5 # Default-Samplerate des Devices
herausfinden (falls None) dev = sd.query_devices(DEVICE_ID) default_sr =
int(dev.get("default_samplerate") or 48000) sr = SAMPLE_RATE or default_sr print(f"Aufnahme
von Device [{DEVICE_ID}] '{dev['name']}' @ {sr} Hz") buf = [] def callback(indata, frames,
time_info, status): if status: print("Status:", status) buf.append(indata.copy()) rms =
np.sqrt(np.mean((indata.astype(np.float32)**2))) bar = "#" * int(min(50, rms * 2000)) # grobe
Skala print(f"\rLevel: {bar:<50}", end="", flush=True) with sd.InputStream(device=DEVICE_ID,
channels=CHANNELS, samplerate=sr, dtype='int16', callback=callback): print("\nAufnahme
läuft... Sprich im Channel. (5s)") time.sleep(DURATION) data = np.concatenate(buf, axis=0)
write("cable_test.wav", sr, data) print("\nFertig. Gespeichert: cable_test.wav")
```

**Was du sehen solltest:** Beim Sprechen (oder Testton in TS) wächst die „####“-Leiste.  
Wenn **keine Balken** kommen, ist **kein Signal** am gewählten Device → zurück zu Schritt 1–3.

### Häufige Stolpersteine

- NPC-Client **Wiedergabe** noch auf echte Lautsprecher → **umstellen auf CABLE Input**.
- Falsches „CABLE Output“ erwischt (z. B. 16-Kanal/Point) → nimm das mit **(2 In / 0 Out)**.

- TS-Client „deafened“ → dann kommt nix in die Wiedergabe.
- Samplerate-Mismatch → beide (Input/Output) auf **48 kHz**.

Wenn du magst, schick mir kurz, **welche ID** du final für:

- CABLE Input (0 In / 2 Out) und
- CABLE Output (2 In / 0 Out)  
setzt — dann gebe ich dir die **fertigen Konstanten** für Aufnahme & Wiedergabe im Endscript.

**Du:**

Das Problem ist, dass im TeamSpeak beim KI-Bot, wenn ich da den CableOutput auswähle, überhaupt nichts ankommt. Selbst wenn ich TestStarte mache und bin im gleichen Raum mit meinem Client, kommt nichts an.

**ChatGPT:**

Ah, perfekt, dass du das so konkret beschreibst 👍 – das erklärt das Problem.

Wenn im **NPC-Client** bei **Aufnahmegerät = CABLE Output** *gar nichts* ankommt, dann liegt's daran, dass die **Wiedergabe des NPC-Clients** nicht ins Kabel läuft.

### Ganz wichtig: Rollen von VB-Cable

- **CABLE Input** → ist ein **Lautsprecher** (Wiedergabegerät). Alles, was du dort abspielst, landet automatisch in ...
- **CABLE Output** → ist ein **Mikrofon** (Aufnahmegerät). Genau dieses muss der NPC-Client als Mikrofon benutzen.

👉 Damit das funktioniert, muss *irgendjemand* wirklich **Audio in den CABLE Input schicken**. Nur dann „kommt was raus“ am CABLE Output.

### Dein Setup jetzt

- Du hast den NPC-Client auf **Mikrofon = CABLE Output** gesetzt → korrekt.
- ABER: Er hört *nichts*, weil seine **Wiedergabe** nicht auf *CABLE Input* läuft, sondern vermutlich noch auf echten Lautsprechern.  
→ Ergebnis: Kein Signal im Kabel, also stille.

### Lösung (Schritt für Schritt)

#### 1. NPC-Client – Wiedergabe

- Einstellungen → Wiedergabe → **Gerät = CABLE Input (VB-Audio Virtual Cable)**
- Testton: Wenn du jetzt den Testton startest, solltest du im VB-Cable Control Panel sehen, dass er „reinläuft“.

#### 2. NPC-Client – Aufnahme



- Aufnahmegerät = **CABLE Output (VB-Audio Virtual Cable)**
- Aktivierung: Dauersenden oder Sprachaktivierung (niedrige Schwelle).

### 3. Dein Haupt-Client

- Unverändert: Mikrofon = RODECaster / normales Setup.

### 4. Test

- Rede mit deinem Haupt-Client im Channel.
- NPC-Client spielt es über *CABLE Input* ab → landet direkt am *CABLE Output* → der NPC „sendet“ es wieder (Mikro).
- In deinem Recording-Script siehst du Pegel.

## Merksatz

**Was in „CABLE Input“ reingeht, kommt am „CABLE Output“ wieder raus.**

👉 Soll ich dir ein kleines **Schema (ASCII-Diagramm mit Pfeilen)** bauen, das genau zeigt, wie dein Audiofluss aussehen muss (damit du's immer vor Augen hast)?

**Du:**

ok die wave wird angelegt - wie geht es weiter

**ChatGPT:**

Yes! 🎉 Wenn die WAV jetzt **nicht mehr leer** ist, dann haben wir den Audiofluss korrekt eingerichtet.

Damit können wir zum eigentlichen Ziel übergehen:

👉 Aus der Aufnahme wird Text → daraus macht OpenAI eine Antwort → diese Antwort wird mit ElevenLabs in Sprache umgewandelt → wieder über VB-Cable in den NPC-Client gespielt → NPC spricht im Channel.

## Nächster Schritt: Transkription mit OpenAI

Wir nehmen die erzeugte WAV und lassen Whisper (Speech-to-Text) drüberlaufen.

### 1. Abhängigkeiten

Falls noch nicht installiert:

```
pip install openai requests
```

### 2. Testscript transcribe.py

```
import requests, os
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") # oder hier direkt eintragen
def transcribe(filename):
    with open(filename, "rb") as f:
        r = requests.post(
            "https://api.openai.com/v1/audio/transcriptions",
            headers={"Authorization": f"Bearer " + OPENAI_API_KEY},
            files={"file": (filename, f, "audio/wav")},
            data={"model": "whisper-1",
```

```
"language": "de"}) r.raise_for_status() print("Transkription:", r.json()["text"]) # Test mit deiner
cable_test.wav transcribe("cable_test.wav")
```

Wenn du nach dem Sprechen im Channel jetzt das Script startest, solltest du den **erkannten Text** im Terminal sehen.

---

### Danach: Antwort generieren + Sprache ausgeben

1. **Antwort generieren:** OpenAI ChatCompletion (gpt-4o-mini).
  2. **TTS mit ElevenLabs** → MP3 erzeugen → über [31] CABLE Input abspielen.
  3. NPC-Client (Mikrofon = [41] CABLE Output) sendet die Antwort in den Channel.
- 

👉 Soll ich dir direkt ein **Kombiscript** bauen, das diese komplette Kette macht (Aufnahme → Transkription → Antwort generieren → ElevenLabs-Sprachdatei abspielen)?

**Du:**

kannst du den Key in einer extra datei abspeichern, damit ich diese nicht im Stream veröffentliche ! ich würde dann die datei ohne stream übertragung anlegen

**ChatGPT:**

Sehr gute Idee 🙌 – so bleiben deine API-Keys sauber getrennt und tauchen nicht in Logs oder im Stream auf.

Es gibt mehrere Möglichkeiten, ich zeige dir die **einfachste Variante mit einer separaten Datei**:

---

### Variante A – .env Datei

1. Lege in deinem Projektordner eine Datei an: .env  
Inhalt z. B.:
2. OPENAI\_API\_KEY=dein\_openai\_key
3. ELEVENLABS\_API\_KEY=dein\_elevenlabs\_key
4. Installiere das Paket:

pip install python-dotenv

5. Im Python-Code:

```
import os from dotenv import load_dotenv load_dotenv() # .env Datei einlesen OPENAI_API_KEY
= os.getenv("OPENAI_API_KEY") ELEVEN_API_KEY = os.getenv("ELEVENLABS_API_KEY")
```

---

### Variante B – keys.json Datei

1. Datei keys.json anlegen:

```
{ "OPENAI_API_KEY": "dein_openai_key", "ELEVENLABS_API_KEY": "dein_elevenlabs_key" }
```

## 2. Im Code einlesen:

```
import json with open("keys.json") as f: keys = json.load(f) OPENAI_API_KEY =  
keys["OPENAI_API_KEY"] ELEVEN_API_KEY = keys["ELEVENLABS_API_KEY"]
```

---

### Empfehlung

- **Für dich** (weil du nicht willst, dass der Key im Stream landet): Variante A mit .env ist am saubersten.
- Wichtig: **.env** oder keys.json in deiner Versionskontrolle (GitHub etc.) unbedingt in .gitignore eintragen.

---

👉 Soll ich dir ein fertiges **Kombi-Script** (Record → Whisper → GPT → ElevenLabs → TS) schreiben, das schon direkt die Keys aus .env lädt?

**Du:**

Variante a kommt der fehler

**Du:**

```
PS C:\kinpc> OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") OPENAI_API_KEY : Die  
Benennung "OPENAI_API_KEY" wurde nicht als Name eines Cmdlet, einer Funktion, einer  
Skriptdatei oder eines ausführbaren Programms erkannt. Überprüfen Sie die Schreibweise des  
Namens, oder ob der Pfad korrekt ist (sofern enthalten), und wiederholen Sie den Vorgang. In  
Zeile:1 Zeichen:1 + OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") + ~~~~~ +  
CategoryInfo          : ObjectNotFound: (OPENAI_API_KEY:String) [], CommandNotFoundException +  
FullyQualifiedErrorId : CommandNotFoundException PS C:\kinpc> ELEVEN_API_KEY =  
os.getenv("ELEVENLABS_API_KEY") ELEVEN_API_KEY : Die Benennung "ELEVEN_API_KEY"  
wurde nicht als Name eines Cmdlet, einer Funktion, einer Skriptdatei oder eines ausführbaren  
Programms erkannt. Überprüfen Sie die Schreibweise des Namens, oder ob der Pfad korrekt ist  
(sofern enthalten), und wiederholen Sie den Vorgang. In Zeile:1 Zeichen:1 + ELEVEN_API_KEY =  
os.getenv("ELEVENLABS_API_KEY") + ~~~~~ + CategoryInfo          : ObjectNotFound:  
(ELEVEN_API_KEY:String) [], CommandNotFoundException + FullyQualifiedErrorId :  
CommandNotFoundException PS C:\kinpc>
```

**ChatGPT:**

Ah, ich sehe sofort, was passiert ist 👍

Du tippst gerade **Python-Code direkt in die PowerShell** – die versteht aber nur PowerShell-Syntax, nicht import ... oder os.getenv(...).

---

✅ **So geht's richtig:**

1. **Datei .env** in C:\kinpc erstellen:
2. OPENAI\_API\_KEY=dein\_openai\_key
3. ELEVENLABS\_API\_KEY=dein\_elevenlabs\_key

#### 4. Python-Skript anlegen (z. B. test\_env.py):

```
import os from dotenv import load_dotenv load_dotenv() # liest .env ein print("OpenAI:",  
os.getenv("OPENAI_API_KEY")) print("ElevenLabs:", os.getenv("ELEVENLABS_API_KEY"))
```

#### 5. In PowerShell ausführen:

```
cd C:\kinpc python test_env.py
```

👉 Jetzt zeigt dir Python die Keys aus der .env an.

In deinem „großen“ Script kannst du dann einfach so arbeiten:

```
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY") ELEVEN_API_KEY =  
os.getenv("ELEVENLABS_API_KEY")
```

---

#### ⚡ Wichtig:

Du darfst **niemals** OPENAI\_API\_KEY = ... direkt in **PowerShell** eingeben, sondern nur in einem Python-Skript oder im Python-Interpreter (python starten → >>> Prompt).

---

👉 Soll ich dir gleich das **Kombi-Script (Record → Whisper → GPT → ElevenLabs)** so umbauen, dass es automatisch .env liest und du nur deine Keys einmal dort einträgst?

**Du:**

ja

**ChatGPT:**

Nachgedacht für 5s

Super — hier ist ein **komplettes Kombi-Script** (Record → Whisper → GPT → ElevenLabs → Play), das deine **Keys aus** .env lädt. Du musst nur noch die **Device-Namen/IDs** checken.

---

## Dateien

### 1) .env (im Ordner C:\kinpc)

```
OPENAI_API_KEY=dein_openai_key
```

```
ELEVENLABS_API_KEY=dein_elevenlabs_key
```

```
ELEVENLABS_VOICE_ID=21m00Tcm4TlvDq8ikWAM
```

Stimme kannst du später ändern (VOICE\_ID).

### 2) npc\_bot.py

```
import os import time import json import numpy as np import requests import sounddevice as  
sd import soundfile as sf from scipy.io.wavfile import write as wav_write from dotenv import  
load_dotenv # ===== # Konfiguration # =====  
RECORD_SECONDS = 3.0 # Länge der Aufnahme (einfacher Ersttest) FALLBACK_SR = 48000 #  
48 kHz ist i.d.R. stabil (VB-Cable) CHANNELS_IN = 1 # Mono reicht CHANNELS_OUT = 2 #  
Wiedergabe meist Stereo NAME_IN = "CABLE Output" # NPC-Client-MIKRO (Input-Device im
```

```

Script) NAME_OUT = "CABLE Input" # NPC-Client-LAUTSPRECHER (Output-Device im Script)
SYSTEM_PROMPT = ( "Du bist ein freundlicher NPC in einer minimalistischen GTA-ähnlichen
Welt. " "Antworte sehr kurz (1–2 Sätze), natürlich und auf Deutsch." ) #
===== # Helpers # ===== def
find_device(name_contains: str, want_input: bool) -> tuple[int, int]: """ Sucht ein Device, dessen
Name 'name_contains' enthält. Gibt (device_id, default_samplerate) zurück. """ devices =
sd.query_devices() for idx, dev in enumerate(devices): name = dev["name"] in_ch, out_ch =
dev["max_input_channels"], dev["max_output_channels"] if name_contains.lower() in
name.lower(): if want_input and in_ch > 0: sr = int(dev.get("default_samplerate") or
FALLBACK_SR) return idx, sr if (not want_input) and out_ch > 0: sr =
int(dev.get("default_samplerate") or FALLBACK_SR) return idx, sr raise RuntimeError(f'Kein
passendes Device gefunden: "{name_contains}" (want_input={want_input})') def
safe_samplerate(sr: int) -> int: return sr if sr in (48000, 44100) else FALLBACK_SR #
===== # API Calls # ===== def
openai_transcribe_wav(path_wav: str, api_key: str, language: str = "de") -> str: with
open(path_wav, "rb") as f: r = requests.post( "https://api.openai.com/v1/audio/transcriptions",
headers={"Authorization": f"Bearer {api_key}"}, files={"file": (os.path.basename(path_wav), f,
"audio/wav")}, data={"model": "whisper-1", "language": language}, timeout=120 )
r.raise_for_status() return r.json().get("text", "").strip() def openai_chat_reply(prompt_text: str,
api_key: str) -> str: payload = { "model": "gpt-4o-mini", "messages": [ {"role": "system", "content":
SYSTEM_PROMPT}, {"role": "user", "content": prompt_text} ], "temperature": 0.6 } r =
requests.post( "https://api.openai.com/v1/chat/completions", headers={"Authorization":
f"Bearer {api_key}", "Content-Type": "application/json" }, json=payload, timeout=120 )
r.raise_for_status() data = r.json() return data["choices"][0]["message"]["content"].strip() def
elevenlabs_tts_to_wav(text: str, api_key: str, voice_id: str, out_path_wav: str): # Wir bitten
ElevenLabs explizit um WAV (PCM), damit wir ohne Zusatzlib abspielen können. url =
f"https://api.elevenlabs.io/v1/text-to-speech/{voice_id}" headers = { "xi-api-key": api_key,
"accept": "audio/wav", "Content-Type": "application/json" } body = { "text": text, "model_id":
"eleven_multilingual_v2", "voice_settings": {"stability": 0.5, "similarity_boost": 0.8} } r =
requests.post(url, headers=headers, json=body, timeout=120) r.raise_for_status() with
open(out_path_wav, "wb") as f: f.write(r.content) # ===== # Hauptlogik #
===== def main(): load_dotenv() # .env einlesen OPENAI_API_KEY =
os.getenv("OPENAI_API_KEY") ELEVEN_API_KEY = os.getenv("ELEVENLABS_API_KEY")
ELEVEN_VOICE_ID = os.getenv("ELEVENLABS_VOICE_ID", "21m00Tcm4TlVdQ8ikWAM") if not
OPENAI_API_KEY: raise RuntimeError("OPENAI_API_KEY fehlt. In .env hinterlegen.") if not
ELEVEN_API_KEY: raise RuntimeError("ELEVENLABS_API_KEY fehlt. In .env hinterlegen.") #
Devices suchen in_id, in_sr_default = find_device(NAME_IN, want_input=True) out_id,
out_sr_default = find_device(NAME_OUT, want_input=False) in_sr =
safe_samplerate(in_sr_default) out_sr = safe_samplerate(out_sr_default) print(f"[Input ]
{NAME_IN} -> id={in_id}, default_sr={in_sr_default} -> using_sr={in_sr}") print(f"[Output]
{NAME_OUT} -> id={out_id}, default_sr={out_sr_default} -> using_sr={out_sr}") # ===== 1)
Aufnehmen ===== rec_frames = int(RECORD_SECONDS * in_sr) print(f"\nAufnahme startet
({RECORD_SECONDS}s) ... bitte im Channel sprechen.") recording = sd.rec( rec_frames,
samplerate=in_sr, channels=CHANNELS_IN, dtype="int16", device=in_id ) sd.wait() wav_in =
"heard.wav" wav_write(wav_in, in_sr, recording) print(f"Aufnahme gespeichert: {wav_in}") #
===== 2) Transkription ===== print("Transkription mit OpenAI (Whisper) ...") text =
openai_transcribe_wav(wav_in, OPENAI_API_KEY, language="de") print(f"Erkannt: {text!r}") if not
text: print("Kein Text erkannt. Abbruch.") return # ===== 3) Antwort generieren =====
print("Antwort wird generiert (GPT) ...) reply = openai_chat_reply(text, OPENAI_API_KEY)

```

```
print(f"NPC: {reply}") # ===== 4) TTS erzeugen ===== wav_out = "npc_reply.wav"
print("ElevenLabs TTS → WAV erzeugen ...") elevenlabs_tts_to_wav(reply, ELEVEN_API_KEY,
ELEVEN_VOICE_ID, wav_out) print(f"TTS gespeichert: {wav_out}") # ===== 5) Abspielen über
CABLE Input ===== print("Antwort wird über CABLE Input abgespielt ...") data, sr =
sf.read(wav_out, dtype="float32", always_2d=True) # Falls Samplerate nicht passt, spielt sd
resampling NICHT automatisch – die meisten VB/TS-Setups sind 48k. # Wenn es kratzt: Stelle
Windows-Devices + TeamSpeak auf 48 kHz. sd.play(data, sr, device=out_id, blocking=True)
print("Fertig.") if __name__ == "__main__": main()
```

---

### Installation (einmalig)

```
cd C:\kinpc python -m pip install --upgrade pip pip install python-dotenv sounddevice soundfile
requests scipy numpy
```

Stelle in Windows unter **Sound → Geräte → Erweitert** für **CABLE Input** und **CABLE Output** am besten **48 kHz** ein.

In TeamSpeak (NPC-Client):

- **Wiedergabe = CABLE Input**
  - **Aufnahme = CABLE Output**
  - Aktivierung = Dauersenden (oder niedrige Sprachaktivierung)
- 

### Start

```
cd C:\kinpc python npc_bot.py
```

- Sprich im Channel → Script nimmt ~3 s auf → transkribiert → erzeugt kurze Antwort → spielt sie über **CABLE Input** ab → der **NPC-Client** sendet sie in den Channel.
- 

### Troubleshooting (kurz)

- **Stille Aufnahme:** Prüfe im NPC-Client, ob **Wiedergabe wirklich CABLE Input** ist und der Client **nicht deafened** ist.
- **Samplerate-Fehler:** Stelle überall (CABLE + TeamSpeak) **48 kHz** ein; Script nimmt dann 48 kHz.
- **Falsches Device erwischt:** Passe NAME\_IN/NAME\_OUT an oder ersetze sie durch feste IDs (aus deiner Liste, z. B. in\_id=41, out\_id=31).

Wenn du willst, passe ich dir das Script jetzt noch schnell auf **deine konkreten IDs** an (nenn mir die ID für “CABLE Output (2 In / 0 Out)” und “CABLE Input (0 In / 2 Out)”).

**Du:**

```
ok was trage ich für devies id in der pc_bot ein hier meine devies PS C:\kinpc> python devices.py
===== Verfügbare Audio-Geräte ===== [0] Microsoft Soundmapper - Input (2 In / 0 Out) [1] Mikrofon
(RODECaster Pro Stereo (2 In / 0 Out) [2] Mikrofon (HZD100) (1 In / 0 Out) [3] CABLE Output (VB-
Audio Virtual (16 In / 0 Out) [4] Digitale Audioschnittstelle (Ca (2 In / 0 Out) [5] Microsoft
```

Soundmapper - Output (0 In / 2 Out) [6] Lautsprecher (RODECaster Pro St (0 In / 2 Out) [7] 3 - BenQ EW2780Q (AMD High Defi (0 In / 2 Out) [8] CABLE Input (VB-Audio Virtual C (0 In / 16 Out) [9] 1 - C24F390 (AMD High Definitio (0 In / 2 Out) [10] Lautsprecher (HSD100) (0 In / 2 Out) [11] SPDIF Interface (Realtek USB2.0 (0 In / 8 Out) [12] 5 - S32E590C (AMD High Definiti (0 In / 2 Out) [13] Lautsprecher (Realtek USB2.0 Au (0 In / 8 Out) [14] CABLE In 16ch (VB-Audio Virtual (0 In / 16 Out) [15] Primärer Soundaufnahmetreiber (2 In / 0 Out) [16] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [17] Mikrofon (HSD100) (1 In / 0 Out) [18] CABLE Output (VB-Audio Virtual Cable) (16 In / 0 Out) [19] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) [20] Primärer Soundtreiber (0 In / 2 Out) [21] Lautsprecher (RODECaster Pro Stereo) (0 In / 2 Out) [22] 3 - BenQ EW2780Q (AMD High Definition Audio Device) (0 In / 2 Out) [23] CABLE Input (VB-Audio Virtual Cable) (0 In / 16 Out) [24] 1 - C24F390 (AMD High Definition Audio Device) (0 In / 2 Out) [25] Lautsprecher (HSD100) (0 In / 2 Out) [26] SPDIF Interface (Realtek USB2.0 Audio) (0 In / 8 Out) [27] 5 - S32E590C (AMD High Definition Audio Device) (0 In / 2 Out) [28] Lautsprecher (Realtek USB2.0 Audio) (0 In / 8 Out) [29] CABLE In 16ch (VB-Audio Virtual Cable) (0 In / 16 Out) [30] 3 - BenQ EW2780Q (AMD High Definition Audio Device) (0 In / 2 Out) [31] CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out) [32] 1 - C24F390 (AMD High Definition Audio Device) (0 In / 2 Out) [33] Lautsprecher (HSD100) (0 In / 2 Out) [34] Lautsprecher (RODECaster Pro Stereo) (0 In / 2 Out) [35] SPDIF Interface (Realtek USB2.0 Audio) (0 In / 2 Out) [36] 5 - S32E590C (AMD High Definition Audio Device) (0 In / 2 Out) [37] Lautsprecher (Realtek USB2.0 Audio) (0 In / 2 Out) [38] CABLE In 16ch (VB-Audio Virtual Cable) (0 In / 2 Out) [39] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [40] Mikrofon (HSD100) (1 In / 0 Out) [41] CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out) [42] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) [43] Mikrofon (Voicemod VAD Wave) (2 In / 0 Out) [44] Line Out (Voicemod VAD Wave) (0 In / 2 Out) [45] Analogger Anschluss (Realtek USB2.0 Audio) (2 In / 0 Out) [46] Line (Realtek USB2.0 Audio) (2 In / 0 Out) [47] Mikrofon (Realtek USB2.0 Audio) (2 In / 0 Out) [48] Speakers (Realtek USB2.0 Audio) (0 In / 8 Out) [49] Headphones (Realtek USB2.0 Audio) (0 In / 2 Out) [50] SPDIF Out (Realtek USB2.0 Audio) (0 In / 2 Out) [51] Speakers (RODECaster Pro Stereo) (0 In / 2 Out) [52] Mikrofon (RODECaster Pro Stereo) (2 In / 0 Out) [53] Output (AMD HD Audio HDMI out #0) (0 In / 2 Out) [54] Output (AMD HD Audio DP out #4) (0 In / 2 Out) [55] Output (AMD HD Audio DP out #2) (0 In / 2 Out) [56] Output (VB-Audio Point) (0 In / 16 Out) [57] Input (VB-Audio Point) (16 In / 0 Out) [58] CABLE Output (VB-Audio Point) (16 In / 0 Out) [59] Lautsprecher (HSD100) (0 In / 2 Out) [60] Mikrofon (HSD100) (1 In / 0 Out) [61] Digitale Audioschnittstelle (Cam Link 4K) (2 In / 0 Out) PS C:\kinpc>

### ChatGPT:

Perfekt 🙌 – jetzt können wir's konkret machen.

In deiner Liste sind die **richtigen VB-Cable Devices**:

- **Aufnahme (was der NPC „hört“)**  
[41] CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out)
- **Wiedergabe (wo der Bot „spricht“)**  
[31] CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out)

---

### **Änderung im npc\_bot.py**

Du kannst die automatische Suche drinlassen – oder fester eintragen, damit er genau deine IDs nimmt.

Ändere oben im Script:

# Direkt IDs setzen IN\_DEVICE\_ID = 41 # CABLE Output OUT\_DEVICE\_ID = 31 # CABLE Input

Dann brauchst du die automatische Suche (find\_device(...)) nicht mehr. Passe main() an:

# Devices fix setzen in\_id, in\_sr\_default = IN\_DEVICE\_ID, 48000 out\_id, out\_sr\_default =  
OUT\_DEVICE\_ID, 48000

Und NAME\_IN / NAME\_OUT kannst du ignorieren.

---

### ⚡ Ergebnis

- Script nimmt 3 Sekunden über Device **41 (CABLE Output)** auf → heard.wav.
  - Schickt an OpenAI → GPT-Antwort.
  - ElevenLabs erzeugt WAV → Script spielt über Device **31 (CABLE Input)** ab.
  - NPC-Client (Aufnahmegerät = CABLE Output) sendet es in den Channel.
- 

👉 Soll ich dir den kompletten npc\_bot.py gleich fertig **mit festen IDs 41 / 31** umschreiben, damit du nur noch starten musst?

**Du:**

pass mir bitte die py datei komplett an, so das ich nur noch kopieren brauche und nicht die datei anpassen mus

**ChatGPT:**

Alles klar — hier ist die **fertige** npc\_bot.py, bereits mit deinen **Device-IDs** fix verdrahtet (IN\_DEVICE\_ID=41 = *CABLE Output*, OUT\_DEVICE\_ID=31 = *CABLE Input*). Du musst nichts mehr im Code ändern—nur .env anlegen und starten.

Lege im gleichen Ordner eine .env mit deinen Keys an:

OPENAI\_API\_KEY=dein\_openai\_key

ELEVENLABS\_API\_KEY=dein\_elevenlabs\_key

ELEVENLABS\_VOICE\_ID=21m00Tcm4TlvDq8ikWAM

(VOICE\_ID kannst du später ändern.)

```
# npc_bot.py # Aufnahme -> Whisper (OpenAI) -> GPT-Antwort -> ElevenLabs TTS -> Abspielen  
über CABLE Input # Fix verdrahtete Geräte: IN_DEVICE_ID=41 (CABLE Output),  
OUT_DEVICE_ID=31 (CABLE Input) import os import time import requests import numpy as np  
import sounddevice as sd import soundfile as sf from scipy.io.wavfile import write as wav_write  
from dotenv import load_dotenv # ===== # Feste Geräte & Settings #  
===== IN_DEVICE_ID = 41 # CABLE Output (VB-Audio Virtual Cable) (2  
In / 0 Out) OUT_DEVICE_ID = 31 # CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out)  
IN_SAMPLERATE = 48000 # Stelle CABLE Input/Output in Windows auf 48 kHz ein  
OUT_SAMPLERATE = 48000 RECORD_SECONDS = 3.0 # Länge der Aufnahme pro Durchlauf  
CHANNELS_IN = 1 # Mono CHANNELS_OUT = 2 # Stereo-Ausgabe SYSTEM_PROMPT = ( "Du bist  
ein freundlicher NPC in einer minimalistischen GTA-ähnlichen Welt. " "Antworte sehr kurz (1-2
```



```

Sätze), natürlich und auf Deutsch." ) def require_env(name: str) -> str: val = os.getenv(name,
"" ).strip() if not val: raise RuntimeError(f"Umgebungsvariable {name} fehlt (.env prüfen).") return
val def openai_transcribe_wav(path_wav: str, api_key: str, language: str = "de") -> str: with
open(path_wav, "rb") as f: r = requests.post( "https://api.openai.com/v1/audio/transcriptions",
headers={"Authorization": f"Bearer {api_key}"}, files={"file": (os.path.basename(path_wav), f,
"audio/wav")}, data={"model": "whisper-1", "language": language}, timeout=120 )
r.raise_for_status() return r.json().get("text", "").strip() def openai_chat_reply(prompt_text: str,
api_key: str) -> str: payload = { "model": "gpt-4o-mini", "messages": [ {"role": "system", "content":
SYSTEM_PROMPT}, {"role": "user", "content": prompt_text} ], "temperature": 0.6 } r =
requests.post( "https://api.openai.com/v1/chat/completions", headers={"Authorization":
f"Bearer {api_key}", "Content-Type": "application/json" }, json=payload, timeout=120 )
r.raise_for_status() data = r.json() return data["choices"][0]["message"]["content"].strip() def
elevenlabs_tts_to_wav(text: str, api_key: str, voice_id: str, out_path_wav: str): url =
f"https://api.elevenlabs.io/v1/text-to-speech/{voice_id}" headers = { "xi-api-key": api_key,
"accept": "audio/wav", "Content-Type": "application/json" } body = { "text": text, "model_id":
"eleven_multilingual_v2", "voice_settings": {"stability": 0.5, "similarity_boost": 0.8} } r =
requests.post(url, headers=headers, json=body, timeout=120) r.raise_for_status() with
open(out_path_wav, "wb") as f: f.write(r.content) def main(): load_dotenv() # .env einlesen
OPENAI_API_KEY = require_env("OPENAI_API_KEY") ELEVEN_API_KEY =
require_env("ELEVENLABS_API_KEY") ELEVEN_VOICE_ID = os.getenv("ELEVENLABS_VOICE_ID",
"21m00Tcm4TlvDq8ikWAM").strip() or "21m00Tcm4TlvDq8ikWAM" # Geräte anzeigen (optional
hilfreich zum Debuggen) dev_in = sd.query_devices(IN_DEVICE_ID) dev_out =
sd.query_devices(OUT_DEVICE_ID) print(f"[Input ] id={IN_DEVICE_ID} -> {dev_in['name']} |
default_sr={dev_in.get('default_samplerate')}") print(f"[Output] id={OUT_DEVICE_ID} ->
{dev_out['name']} | default_sr={dev_out.get('default_samplerate')}") print(f"Nutze SR:
In={IN_SAMPLERATE} Hz, Out={OUT_SAMPLERATE} Hz") # 1) Aufnahme frames =
int(RECORD_SECONDS * IN_SAMPLERATE) print(f"\nAufnahme startet ({RECORD_SECONDS}s)
... bitte im Channel sprechen.") recording = sd.rec( frames, samplerate=IN_SAMPLERATE,
channels=CHANNELS_IN, dtype="int16", device=IN_DEVICE_ID ) sd.wait() wav_in = "heard.wav"
wav_write(wav_in, IN_SAMPLERATE, recording) print(f"Aufnahme gespeichert: {wav_in}") # 2)
Transkription (Whisper) print("Transkription mit OpenAI (Whisper) ...") text =
openai_transcribe_wav(wav_in, OPENAI_API_KEY, language="de") print(f"Erkannt: {text!r}") if not
text: print("Kein Text erkannt. Ende.") return # 3) Antwort (GPT) print("Antwort wird generiert (GPT)
...") reply = openai_chat_reply(text, OPENAI_API_KEY) print(f"NPC: {reply}") # 4) TTS (ElevenLabs)
wav_out = "npc_reply.wav" print("ElevenLabs TTS -> WAV erzeugen ...")
elevenlabs_tts_to_wav(reply, ELEVEN_API_KEY, ELEVEN_VOICE_ID, wav_out) print(f"TTS
gespeichert: {wav_out}") # 5) Abspielen über CABLE Input (NPC-Client sendet's dann in den
Channel) print("Antwort wird über CABLE Input abgespielt ...") data, sr = sf.read(wav_out,
dtype="float32", always_2d=True) sd.play(data, sr, device=OUT_DEVICE_ID, blocking=True)
print("Fertig.") if __name__ == "__main__": try: main() except Exception as e: print("FEHLER:", e)

```

## Installation (einmal)

```

cd C:\kinpc python -m pip install --upgrade pip pip install python-dotenv sounddevice soundfile
requests scipy numpy

```

## Start

```

cd C:\kinpc python npc_bot.py

```

## TeamSpeak-Check (NPC-Client):

- Wiedergabe: **CABLE Input**
- Aufnahme: **CABLE Output**
- Aktivierung: *Dauersenden* (oder niedrige Sprachaktivierung)
- Beide VB-Cable-Geräte in Windows auf **48 kHz** stellen.

Wenn irgendwas hakt, sag mir kurz die Konsolen-Ausgabe – dann tweake ich's sofort.

**Du:**

ich bekomme ein problem mitder schnittstelle NPC: Ich bleibe hier, um dir zu helfen! Intelligenz ist relativ, aber ich gebe mein Bestes. ElevenLabs TTS → WAV erzeugen ... FEHLER: 401 Client Error: Unauthorized for url: <https://api.elevenlabs.io/v1/text-to-speech/MijDeTiicOjZ6GxiwvCP>

**ChatGPT:**

Nachgedacht für 9s

401 = dein ElevenLabs-Key kommt bei der Anfrage nicht (korrekt) an – oder die Voice-ID ist für deinen Account nicht freigeschaltet. Lass uns das schnell einkreisen.

### 1) Schnell-Checks an der .env

- Genau so (ohne Anführungszeichen, keine Spaces vor/nach =):
- OPENAI\_API\_KEY=sk-live-...
- ELEVENLABS\_API\_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxx
- ELEVENLABS\_VOICE\_ID=21m00Tcm4TlvDq8ikWAM
- Liegt die .env in **C:\kinpc** (dem Ordner, aus dem du python npc\_bot.py startest)?
- In der .env dürfen **keine** führenden BOM/Unicode-Sonderzeichen sein. (Im Notepad als ANSI/UTF-8 ohne BOM speichern.)
- Danach das Skript erneut starten (damit load\_dotenv() neu lädt).

401 kommt, wenn der Header nicht stimmt oder der Key leer/ungültig ist. Unser Code nutzt korrekt xi-api-key, aber prüfen wir das mit einem Mini-Auth-Test.

### 2) Kleiner Auth-Test (prüft Key & zeigt Kontoinfo)

Erstelle eleven\_test.py im gleichen Ordner:

```
import os, requests
from dotenv import load_dotenv
load_dotenv()
key = os.getenv("ELEVENLABS_API_KEY", "").strip()
assert key, "ELEVENLABS_API_KEY fehlt oder ist leer"
r = requests.get("https://api.elevenlabs.io/v1/user", headers={"xi-api-key": key}, timeout=30)
print("Status:", r.status_code)
print("Body:", r.text)
```

- python eleven\_test.py
  - **200 + JSON** → Key ok
  - **401** → Key falsch/leer/Formatproblem

### 3) Voice-ID prüfen

Manche Voices sind **nicht** für jeden Account freigeschaltet. Hole deine verfügbaren Voices:

```
import os, requests from dotenv import load_dotenv load_dotenv() key =
os.getenv("ELEVENLABS_API_KEY", "").strip() r = requests.get(
"https://api.elevenlabs.io/v1/voices", headers={"xi-api-key": key}, timeout=30 )
print(r.status_code) print(r.json())
```

- Such dir eine **Voice-ID aus dieser Liste** (z. B. „Rachel“ 21m00Tcm4TlvDq8ikWAM) und trage sie in .env ein:
- ELEVENLABS\_VOICE\_ID=21m00Tcm4TlvDq8ikWAM

#### 4) Mini-TTS-Ping (isoliert)

Testet TTS ohne den ganzen Pipeline-Kram:

```
import os, requests from dotenv import load_dotenv load_dotenv() key =
os.getenv("ELEVENLABS_API_KEY", "").strip() voice = os.getenv("ELEVENLABS_VOICE_ID",
"21m00Tcm4TlvDq8ikWAM").strip() or "21m00Tcm4TlvDq8ikWAM" url =
f"https://api.elevenlabs.io/v1/text-to-speech/{voice}" headers = {"xi-api-key": key, "accept":
"audio/wav", "Content-Type": "application/json"} body = {"text": "Test. Eins zwei drei.",
"model_id": "eleven_multilingual_v2"} resp = requests.post(url, headers=headers, json=body,
timeout=60) print("Status:", resp.status_code) print("Headers:", resp.headers) print("Sample of
content:", resp.content[:16]) if resp.ok: with open("tts_test.wav", "wb") as f: f.write(resp.content)
print("tts_test.wav geschrieben") else: print("Body:", resp.text)
```

- **200 + tts\_test.wav** → TTS ok.
- **401** → Key/Voice weiterhin das Problem (siehe Punkte 1–3).

#### 5) Kleines Update für dein npc\_bot.py (besseres Fehlermeldung/Preflight)

Ersetze in deinem Script die TTS-Funktion durch diese Version (liefert klarere Fehler):

```
def elevenlabs_tts_to_wav(text: str, api_key: str, voice_id: str, out_path_wav: str): url =
f"https://api.elevenlabs.io/v1/text-to-speech/{voice_id}" headers = { "xi-api-key": api_key.strip(),
"accept": "audio/wav", "Content-Type": "application/json" } body = { "text": text, "model_id":
"eleven_multilingual_v2", "voice_settings": {"stability": 0.5, "similarity_boost": 0.8} } resp =
requests.post(url, headers=headers, json=body, timeout=120) if resp.status_code == 401: raise
RuntimeError("ElevenLabs 401 Unauthorized – prüfe ELEVENLABS_API_KEY in .env (ohne
Anführungszeichen), " "und ob die Voice-ID zu deinem Account gehört.") if resp.status_code ==
404: raise RuntimeError("ElevenLabs 404 – Voice-ID nicht gefunden oder nicht zugelassen für
deinen Account.") resp.raise_for_status() with open(out_path_wav, "wb") as f:
f.write(resp.content)
```

---

#### Typische Ursachen & Fix

- **Leerzeichen/Anführungszeichen** in .env: ELEVENLABS\_API\_KEY="abc..." ❌ → ohne Anführungszeichen ✅
- .env im **falschen Ordner**: load\_dotenv() lädt nur die .env neben deinem Skript (aktuelles Arbeitsverzeichnis).
- **Falsche Voice-ID**: nutze eine ID aus **deiner** /v1/voices-Liste.

- **Account/Plan:** sehr selten blockt ein Plan/Quota – dann kommt aber eher **429/402**, nicht 401.

Wenn du mir kurz sagst, was eleven\_test.py bei **/v1/user** zurückgibt (Status/Body) und ob deine **/v1/voices** eine Liste liefert, sage ich dir exakt, welche Voice-ID du setzen kannst – und wir sind durch.

**Du:**

ne er zeigt immer noch den selben fehler

**ChatGPT:**

Nachgedacht für 4 Sekunden

Alles klar—dann ist es fast sicher ein **Key-Ladeproblem** ( .env wird nicht gelesen / falscher Pfad / unsichtbares Zeichen ) **oder** eine **Voice-ID**, die dein Account nicht nutzen darf. Lass uns das in 2 Minuten eindeutig machen.

### Schritt 1: Diagnose-Skript (lädt Key, prüft API)

Speichere als diag\_env\_and\_eleven.py im selben Ordner wie dein npc\_bot.py (z. B. C:\kinpc), dann python diag\_env\_and\_eleven.py ausführen.

```
import os, json, requests, sys from dotenv import load_dotenv print("Arbeitsverzeichnis:",
os.getcwd()) # 1) .env gezielt laden dotenv_paths = [ os.path.join(os.getcwd(), ".env"),
os.path.join(os.path.dirname(__file__), ".env"), ] loaded = False for p in dotenv_paths: if
os.path.isfile(p): print("Lade .env von:", p) load_dotenv(p, override=True) loaded = True break if
not loaded: print("WARNUNG: Keine .env gefunden an:", dotenv_paths) # 2) Keys lesen +
maskiert ausgeben openai_key = (os.getenv("OPENAI_API_KEY") or "").strip() eleven_key =
(os.getenv("ELEVENLABS_API_KEY") or "").strip() voice_id =
(os.getenv("ELEVENLABS_VOICE_ID") or "").strip() def mask(s): return f"{s[:4]}...{s[-4:]}" if s and
len(s) > 8 else s print("OPENAI_API_KEY:", mask(openai_key), "len=", len(openai_key))
print("ELEVENLABS_API_KEY:", mask(eleven_key), "len=", len(eleven_key))
print("ELEVENLABS_VOICE_ID:", voice_id or "(leer)") if not eleven_key: print("\n ✖
ELEVENLABS_API_KEY ist leer. Prüfe .env (keine Anführungszeichen, keine Leerzeichen).")
sys.exit(1) # 3) /v1/user testen try: r = requests.get("https://api.elevenlabs.io/v1/user",
headers={"xi-api-key": eleven_key}, timeout=20) print("GET /v1/user ->", r.status_code)
print("Body:", r.text[:400]) if r.status_code == 401: print("\n ✖ 401 Unauthorized: Key
falsch/abgelaufen/leer oder Encoding-Problem (.env.)") print(" -> Regeneriere den API-Key im
ElevenLabs-Dashboard und ersetze ihn in .env.") sys.exit(1) except Exception as e: print("HTTP-
Fehler:", e) sys.exit(1) # 4) Voices abrufen r = requests.get("https://api.elevenlabs.io/v1/voices",
headers={"xi-api-key": eleven_key}, timeout=30) print("\nGET /v1/voices ->", r.status_code) if
r.ok: voices = r.json().get("voices", []) print(f"Gefundene Voices: {len(voices)}") if voices:
print("Beispiel-ID:", voices[0].get("voice_id"), "| Name:", voices[0].get("name")) else:
print("WARN: Keine Voices im Account gelistet.") else: print("Body:", r.text[:400])
```

### Erwartung

- **OK:** GET /v1/user -> 200 und eine Liste bei /v1/voices.
- **Noch 401:** .env wird nicht korrekt geladen (oder Key ungültig).

## Schritt 2: Schnell-Fix, falls .env zickt

Wähle **eine** der Varianten:

### Variante A (für die aktuelle PowerShell-Session)

```
$env:ELEVENLABS_API_KEY="DEIN_KEY_OHNE_ANFÜHRUNGSZEICHEN"  
$env:ELEVENLABS_VOICE_ID="21m00Tcm4TlvDq8ikWAM" python diag_env_and_eleven.py
```

Wenn das klappt (200), dann lädt dein Script die .env nicht → Pfad/BOM/Format fixen.

### Variante B (persistente User-Variable)

```
setx ELEVENLABS_API_KEY "DEIN_KEY_OHNE_ANFÜHRUNGSZEICHEN" setx  
ELEVENLABS_VOICE_ID "21m00Tcm4TlvDq8ikWAM"
```

Terminal **neu öffnen**, dann python diag\_env\_and\_eleven.py.

### Variante C (keys.json – super simpel & sicher lokal)

keys.json im gleichen Ordner anlegen:

```
{ "OPENAI_API_KEY": "sk-live-...", "ELEVENLABS_API_KEY": "...", "ELEVENLABS_VOICE_ID":  
"21m00Tcm4TlvDq8ikWAM" }
```

Und im npc\_bot.py ganz oben ergänzen (vor dem dotenv-Teil oder statt dessen):

```
import json, os  
KEYS_PATH = os.path.join(os.getcwd(), "keys.json")  
if os.path.isfile(KEYS_PATH):  
    with open(KEYS_PATH, "r", encoding="utf-8") as f: keys = json.load(f)  
    os.environ.setdefault("OPENAI_API_KEY", keys.get("OPENAI_API_KEY",""))  
    os.environ.setdefault("ELEVENLABS_API_KEY", keys.get("ELEVENLABS_API_KEY",""))  
    os.environ.setdefault("ELEVENLABS_VOICE_ID",  
        keys.get("ELEVENLABS_VOICE_ID","21m00Tcm4TlvDq8ikWAM"))
```

---

## Schritt 3: Voice-ID wirklich aus deinem Account

Nach diag\_env\_and\_eleven.py siehst du bei /v1/voices mind. eine ID/Name. Nimm eine davon (z. B. „Rachel“ 21m00Tcm4TlvDq8ikWAM) in .env/keys.json.

---

## Schritt 4: Nochmal dein Bot starten

Wenn der Diagnose-Test 200 liefert:

```
python npc_bot.py
```

---

## Wenn's weiterhin 401 ist

- Key im Dashboard **neu erzeugen** und **kopieren** **ohne** Anführungszeichen/Leerzeichen/Zeilenumbrüche.
- .env als **UTF-8 ohne BOM** speichern.
- Sicherstellen, dass du **aus dem Ordner mit der** .env startest (oder in Code load\_dotenv(path) wie im Diagnose-Skript).

Schick mir die **Ausgabe von** `diag_env_and_eleven.py` (**Status/Maskenlängen**), dann sag ich dir genau, wo's hängt.

**Du:**

```
ok geht - nun habe ich das Problem, dass dieser Fehler kommt: PS C:\kinpc> python npc_bot.py
[Input ] id=41 -> CABLE Output (VB-Audio Virtual Cable) | default_sr=48000.0 [Output] id=31 ->
CABLE Input (VB-Audio Virtual Cable) | default_sr=48000.0 Nutze SR: In=48000 Hz, Out=48000
Hz Aufnahme startet (3.0s) ... bitte im Channel sprechen. Aufnahme gespeichert: heard.wav
Transkription mit OpenAI (Whisper) ... Erkannt: 'Ähm ja, keine Ahnung.' Antwort wird generiert
(GPT) ... NPC: Kein Problem! Was kann ich für dich tun? ElevenLabs TTS → WAV erzeugen ... TTS
gespeichert: npc_reply.wav Antwort wird über CABLE Input abgespielt ... FEHLER: Error opening
OutputStream: Invalid sample rate [PaErrorCode -9997] PS C:\kinpc>
```

**ChatGPT:**

Nachgedacht für 6s

Das ist "nur" die **Samplerate beim Abspielen**.

ElevenLabs liefert die WAV oft **nicht** mit 48 kHz zurück (z. B. 22.05 kHz/44.1 kHz). Dein **CABLE Input** steht aber auf **48 kHz** → Invalid sample rate.

**Fix: vor dem Abspielen auf 48 kHz resampeln**

Füge diese Helferfunktion in `npc_bot.py` ein (oben bei den Imports steht schon `numpy`, `soundfile`, `sounddevice`; wir nutzen zusätzlich `scipy.signal`):

```
from scipy.signal import resample_poly
```

Und **ersetze** den Abspiel-Block (Schritt 5) durch diesen:

```
# ===== 5) Abspielen über CABLE Input (NPC-Client sendet's dann in den Channel)
print("Antwort wird über CABLE Input abgespielt ...") data, sr_in = sf.read(wav_out,
dtype="float32", always_2d=True) # shape: (N, ch) target_sr = OUT_SAMPLERATE # 48000 if sr_in
!= target_sr: # sauberes Resampling via polyphase (weniger Aliasing als naive Methode) g =
np.gcd(sr_in, target_sr) up, down = target_sr // g, sr_in // g data = resample_poly(data, up, down,
axis=0) sr_play = target_sr else: sr_play = sr_in # falls Mono, für TS/CABLE auf 2 Kanäle
duplizieren if data.ndim == 1: data = data[:, None] if data.shape[1] == 1: data = np.repeat(data, 2,
axis=1) sd.play(data, sr_play, device=OUT_DEVICE_ID, blocking=True) print("Fertig.")
```

**Zusätzlich prüfen**

- In Windows Sound → **CABLE Input & CABLE Output** beide auf **48 000 Hz** stellen.
- TeamSpeak (NPC-Client) → Wiedergabe/Aufnahme auf die gleichen Geräte (hast du bereits).

Starte danach erneut:

```
python npc_bot.py
```

Wenn noch was knarzt oder meckert, poste mir bitte die **erste Zeile nach** `sf.read` (also welche `sr_in` ankommt) — dann feilen wir das Feintuning.

**Du:**

kannst du mir die komplette `npc_bot.py` neu korrekt korrigieren?

## ChatGPT:

klar—hier ist die **vollständig korrigierte** npc\_bot.py mit fest verdrahteten Devices (IN=41, OUT=31), .env-Support, sauberem **Resampling auf 48 kHz** und kleinen Guards (Stereo-Erzwingung, klare Fehlermeldungen). Einfach 1:1 ersetzen und starten.

Stelle sicher, dass du diese Pakete hast:

```
pip install python-dotenv sounddevice soundfile requests scipy numpy
```

Und dass in **C:\kinpc.env** steht:

```
OPENAI_API_KEY=dein_openai_key
```

```
ELEVENLABS_API_KEY=dein_elevenlabs_key
```

```
ELEVENLABS_VOICE_ID=21m00Tcm4TlvDq8ikWAM
```

```
# npc_bot.py # Aufnahme -> Whisper (OpenAI) -> GPT-Antwort -> ElevenLabs TTS -> Abspielen
über CABLE Input # Fix: IN_DEVICE_ID=41 (CABLE Output), OUT_DEVICE_ID=31 (CABLE Input) #
Enthält Resampling auf 48 kHz für fehlerfreie Wiedergabe. import os import requests import
numpy as np import sounddevice as sd import soundfile as sf from scipy.io.wavfile import write
as wav_write from scipy.signal import resample_poly from dotenv import load_dotenv #
===== # Feste Geräte & Settings # =====
IN_DEVICE_ID = 41 # CABLE Output (VB-Audio Virtual Cable) (2 In / 0 Out) OUT_DEVICE_ID = 31
# CABLE Input (VB-Audio Virtual Cable) (0 In / 2 Out) IN_SAMPLERATE = 48000 # Stelle CABLE
I/O in Windows auf 48 kHz OUT_SAMPLERATE = 48000 RECORD_SECONDS = 3.0 # Länge der
Aufnahme pro Durchlauf CHANNELS_IN = 1 # Mono SYSTEM_PROMPT = ( "Du bist ein
freundlicher NPC in einer minimalistischen GTA-ähnlichen Welt. " "Antworte sehr kurz (1–2
Sätze), natürlich und auf Deutsch." ) # ===== # Hilfsfunktionen #
===== def require_env(name: str) -> str: val = os.getenv(name, "").strip()
if not val: raise RuntimeError(f"Umgebungsvariable {name} fehlt (.env prüfen).") return val def
openai_transcribe_wav(path_wav: str, api_key: str, language: str = "de") -> str: with
open(path_wav, "rb") as f: r = requests.post("https://api.openai.com/v1/audio/transcriptions",
headers={"Authorization": f"Bearer {api_key}"}, files={"file": (os.path.basename(path_wav), f,
"audio/wav")}, data={"model": "whisper-1", "language": language}, timeout=120 )
r.raise_for_status() return r.json().get("text", "").strip() def openai_chat_reply(prompt_text: str,
api_key: str) -> str: payload = { "model": "gpt-4o-mini", "messages": [ {"role": "system", "content":
SYSTEM_PROMPT}, {"role": "user", "content": prompt_text} ], "temperature": 0.6 } r =
requests.post("https://api.openai.com/v1/chat/completions", headers={"Authorization":
f"Bearer {api_key}", "Content-Type": "application/json" }, json=payload, timeout=120 )
r.raise_for_status() data = r.json() return data["choices"][0]["message"]["content"].strip() def
elevenlabs_tts_to_wav(text: str, api_key: str, voice_id: str, out_path_wav: str): url =
f"https://api.elevenlabs.io/v1/text-to-speech/{voice_id}" headers = { "xi-api-key": api_key.strip(),
"accept": "audio/wav", # WAV anfordern -> leicht zu verarbeiten "Content-Type":
"application/json" } body = { "text": text, "model_id": "eleven_multilingual_v2", "voice_settings":
{"stability": 0.5, "similarity_boost": 0.8} } resp = requests.post(url, headers=headers, json=body,
timeout=120) if resp.status_code == 401: raise RuntimeError("ElevenLabs 401 Unauthorized –
prüfe ELEVENLABS_API_KEY in .env und Voice-ID.") if resp.status_code == 404: raise
RuntimeError("ElevenLabs 404 – Voice-ID nicht gefunden/zugelassen.") resp.raise_for_status()
with open(out_path_wav, "wb") as f: f.write(resp.content) def to_stereo(data: np.ndarray) ->
np.ndarray: """Sorge für 2 Kanäle (TeamSpeak/CABLE mögen i.d.R. Stereo bei Wiedergabe).""" if
```

```

data.ndim == 1: data = data[:, None] if data.shape[1] == 1: data = np.repeat(data, 2, axis=1)
return data def resample_if_needed(data: np.ndarray, sr_in: int, sr_target: int) ->
tuple[np.ndarray, int]: """Sauberes polyphase Resampling, wenn Eingangs-SR != Ziel-SR.""" if
sr_in == sr_target: return data, sr_in # gcd für saubere up/down-rates g = np.gcd(int(sr_in),
int(sr_target)) up, down = int(sr_target // g), int(sr_in // g) data_rs = resample_poly(data, up,
down, axis=0) return data_rs, sr_target # ===== # Hauptablauf #
===== def main(): load_dotenv() # .env einlesen OPENAI_API_KEY =
require_env("OPENAI_API_KEY") ELEVEN_API_KEY = require_env("ELEVENLABS_API_KEY")
ELEVEN_VOICE_ID = os.getenv("ELEVENLABS_VOICE_ID", "21m00Tcm4TlvDq8ikWAM").strip() or
"21m00Tcm4TlvDq8ikWAM" # Geräteinfo (hilft beim Debuggen) dev_in =
sd.query_devices(IN_DEVICE_ID) dev_out = sd.query_devices(OUT_DEVICE_ID) print(f"[Input ]
id={IN_DEVICE_ID} -> {dev_in['name']} | default_sr={dev_in.get('default_samplerate')}")
print(f"[Output] id={OUT_DEVICE_ID} -> {dev_out['name']} |
default_sr={dev_out.get('default_samplerate')}") print(f"Nutze SR: In={IN_SAMPLERATE} Hz,
Out={OUT_SAMPLERATE} Hz\n") # ===== 1) Aufnehmen ===== frames = int(RECORD_SECONDS
* IN_SAMPLERATE) print(f"Aufnahme startet ({RECORD_SECONDS}s) ... bitte im Channel
sprechen.") recording = sd.rec( frames, samplerate=IN_SAMPLERATE, channels=CHANNELS_IN,
dtype="int16", device=IN_DEVICE_ID ) sd.wait() wav_in = "heard.wav" wav_write(wav_in,
IN_SAMPLERATE, recording) print(f"Aufnahme gespeichert: {wav_in}") # ===== 2) Transkription
(Whisper) ===== print("Transkription mit OpenAI (Whisper) ...") text =
openai_transcribe_wav(wav_in, OPENAI_API_KEY, language="de") print(f"Erkannt: {text!r}") if not
text: print("Kein Text erkannt. Ende.") return # ===== 3) Antwort (GPT) ===== print("Antwort wird
generiert (GPT) ...") reply = openai_chat_reply(text, OPENAI_API_KEY) print(f"NPC: {reply}") #
===== 4) TTS (ElevenLabs) -> WAV speichern ===== wav_out = "npc_reply.wav" print("ElevenLabs
TTS -> WAV erzeugen ...") elevenlabs_tts_to_wav(reply, ELEVEN_API_KEY, ELEVEN_VOICE_ID,
wav_out) print(f"TTS gespeichert: {wav_out}") # ===== 5) Abspielen über CABLE Input (mit
Resampling & Stereo) ===== print("Antwort wird über CABLE Input abgespielt ...") data, sr_in =
sf.read(wav_out, dtype="float32", always_2d=True) data, sr_play = resample_if_needed(data,
sr_in, OUT_SAMPLERATE) data = to_stereo(data) sd.play(data, sr_play, device=OUT_DEVICE_ID,
blocking=True) print("Fertig.") if __name__ == "__main__": try: main() except Exception as e:
print("FEHLER:", e)

```

### Hinweise:

- In Windows-Sound -> **CABLE Input** und **CABLE Output** auf **48 kHz** stellen.
- TeamSpeak (NPC-Client): Wiedergabe = **CABLE Input**, Aufnahme = **CABLE Output**, Aktivierung = *Dauersenden* (oder niedrige Sprachaktivierung).

Wenn noch etwas zickt, poste mir einfach die Konsolenausgabe – dann justieren wir gezielt nach.

### Du:

ok mega - es geht! Jetzt ist die Frage: Gleich sprach channel! Wie bekommen wir denn jetzt meine kleine Minimap verbunden! Es geht ja darum, dass wenn die beiden Avatare sich näher kommen der NPC reagiert und den Spieler anspricht !

### ChatGPT:

Nachgedacht für 17s



Geil, dass die Audiokette läuft! 🎉 Jetzt koppeln wir die **Minimap** → **NPC**-Logik, damit der Bot bei **Nähe** reagiert und anspricht.

### Was wir bauen

- Dein NPC-Bot läuft weiter lokal.
- Er öffnet einen **kleinen HTTP-Endpoint** (POST /update) für Positionsdaten.
- Deine Minimap schickt dort regelmäßig:

```
{ "npc": {"x": 10.2, "y": -3.4}, "players": [{"id": "p1", "name": "Florian", "x": 12.0, "y": -2.0}] }
```

- Ein **Watcher** prüft Nähe (z. B.  $\leq 8$  m) und triggert eine **kurze Begrüßung** (OpenAI → ElevenLabs → TS).

### Install (einmalig)

```
pip install fastapi uvicorn
```

### Neue Datei: npc\_bot\_proximity.py (komplett, Copy/Paste)

Nutzt weiter deine VB-Cable IDs: **IN=41** (CABLE Output), **OUT=31** (CABLE Input).

Liest .env (OPENAI\_API\_KEY, ELEVENLABS\_API\_KEY, ELEVENLABS\_VOICE\_ID).

```
# npc_bot_proximity.py # Minimap -> POST /update mit Positionen # NPC begrüßt Spieler bei
Nähe (GPT -> ElevenLabs -> Wiedergabe über CABLE Input)
import os, time, math, threading
import numpy as np
import requests
import sounddevice as sd
import soundfile as sf
from scipy.signal import resample_poly
from dotenv import load_dotenv
from fastapi import FastAPI, Request
import uvicorn

# ===== Audio-Setup (wie zuvor) =====
IN_DEVICE_ID = 41 # CABLE Output (2 In / 0 Out)
OUT_DEVICE_ID = 31 # CABLE Input (0 In / 2 Out)
# hier spielt der Bot
TARGET_SR = 48000 # Stelle CABLE Input/Output in Windows auf 48 kHz
# ===== Bot-Verhalten =====
GREET_RADIUS = 8.0 # Meter
COOLDOWN_SEC = 20 # nicht ständig grüßen
SYSTEM_PROMPT = ( "Du bist ein freundlicher NPC in einer minimalistischen GTA-ähnlichen Welt. "
"Antworte sehr kurz (1–2 Sätze), locker und auf Deutsch." )
# ===== State =====
positions_lock = threading.Lock()
npc_pos = (0.0, 0.0)
players = {} # id -> {"name": str, "x": float, "y": float, "last_greet": float}
# ===== Helpers =====
def dist(a, b):
    return math.hypot(a[0] - b[0], a[1] - b[1])
def to_stereo(data: np.ndarray) -> np.ndarray:
    if data.ndim == 1:
        data = data[:, None]
    if data.shape[1] == 1:
        data = np.repeat(data, 2, axis=1)
    return data
def resample_if_needed(data: np.ndarray, sr_in: int, sr_target: int):
    if sr_in == sr_target:
        return data
    g = math.gcd(int(sr_in), int(sr_target))
    up, down = int(sr_target // g), int(sr_in // g)
    return resample_poly(data, up, down, axis=0), sr_target
def require_env(name: str) -> str:
    val = os.getenv(name, "")
    if not val:
        raise RuntimeError(f"{name} fehlt in .env")
    return val

# ===== APIs =====
def openai_chat_reply(prompt_text: str, api_key: str) -> str:
    payload = {
        "model": "gpt-4o-mini",
        "messages": [
            {"role": "system", "content": SYSTEM_PROMPT},
            {"role": "user", "content": prompt_text}
        ],
        "temperature": 0.6,
    }
    r = requests.post(
        "https://api.openai.com/v1/chat/completions",
        headers={"Authorization": f"Bearer {api_key}"},
        {"Content-Type": "application/json"},
        json=payload,
        timeout=60
    )
    r.raise_for_status()
    return r.json()[0]["choices"][0]["message"]["content"].strip()

def elevenlabs_tts_to_wav(text: str, api_key: str, voice_id: str, out_path_wav: str):
    url = f"https://api.elevenlabs.io/v1/text-to-speech/{voice_id}"
    headers = {
        "xi-api-key": api_key.strip(),
        "accept": "audio/wav",
        "Content-Type": "application/json"
    }
    body = {
        "text": text,
        "model_id": "eleven_multilingual_v2",
        "voice_settings": {
            "stability": 0.5,
            "similarity_boost": 0.8
        }
    }
    resp = requests.post(url, headers=headers, json=body, timeout=120)
    resp.raise_for_status()
    with open(out_path_wav, "wb") as f:
        f.write(resp.content)
```

```

"wb") as f: f.write(resp.content) def speak_text(text: str, eleven_key: str, voice_id: str): # TTS
erzeugen wav_out = "npc_reply.wav" elevenlabs_tts_to_wav(text, eleven_key, voice_id, wav_out)
# abspielen (Resample -> 48k, Stereo) data, sr_in = sf.read(wav_out, dtype="float32",
always_2d=True) data, sr_play = resample_if_needed(data, sr_in, TARGET_SR) data =
to_stereo(data) sd.play(data, sr_play, device=OUT_DEVICE_ID, blocking=True) # ===== Proximity
Watcher ===== def proximity_loop(openai_key: str, eleven_key: str, voice_id: str, npc_name: str =
"NPC"): while True: time.sleep(0.2) now = time.time() with positions_lock: my_pos = npc_pos
snapshot = [(pid, pdata["name"], (pdata["x"], pdata["y"]), pdata.get("last_greet", 0)) for pid, pdata
in players.items()] for pid, pname, ppos, last in snapshot: if dist(my_pos, ppos) <=
GREET_RADIUS and (now - last) > COOLDOWN_SEC: # Kurze, personalisierte Begrüßung
user_text = f"Begrüße den Spieler '{pname}' freundlich und kurz (max. 2 Sätze)." try: reply =
openai_chat_reply(user_text, openai_key) speak_text(reply, eleven_key, voice_id) except
Exception as e: print("Greet-Fehler:", e) # Cooldown setzen with positions_lock:
players[pid]["last_greet"] = time.time() # ===== FastAPI App ===== app = FastAPI()
@app.post("/update") async def update_positions(req: Request): body = await req.json() npc =
body.get("npc") or {} plist = body.get("players") or [] with positions_lock: # NPC-Position if "x" in
npc and "y" in npc: global npc_pos npc_pos = (float(npc["x"]), float(npc["y"])) # Spieler-
Positionen for p in plist: pid = str(p.get("id", "")) if not pid: continue name = p.get("name") or pid x
= float(p.get("x", 0.0)) y = float(p.get("y", 0.0)) entry = players.get(pid, {"name": name, "x": x, "y": y,
"last_greet": 0.0}) entry.update({"name": name, "x": x, "y": y}) players[pid] = entry return {"ok":
True} def main(): load_dotenv() openai_key = require_env("OPENAI_API_KEY") eleven_key =
require_env("ELEVENLABS_API_KEY") voice_id = os.getenv("ELEVENLABS_VOICE_ID",
"21m00Tcm4TlvDq8ikWAM").strip() or "21m00Tcm4TlvDq8ikWAM" # Geräte anzeigen (nur Info)
dev_in = sd.query_devices(IN_DEVICE_ID) dev_out = sd.query_devices(OUT_DEVICE_ID)
print(f"[Input ] id={IN_DEVICE_ID} -> {dev_in['name']} |
default_sr={dev_in.get('default_samplerate')}") print(f"[Output] id={OUT_DEVICE_ID} ->
{dev_out['name']} | default_sr={dev_out.get('default_samplerate')}") print(f"Target SR:
{TARGET_SR} Hz") print("HTTP-Endpoint: POST http://127.0.0.1:9000/update") # Proximity-
Thread starten t = threading.Thread(target=proximity_loop, args=(openai_key, eleven_key,
voice_id), daemon=True) t.start() # FastAPI Server starten (lokal) uvicorn.run(app,
host="127.0.0.1", port=9000) if __name__ == "__main__": main()

```

## Starten

python npc\_bot\_proximity.py

## Minimap → Positions-Update senden (Beispiel)

Schick aus deiner Minimap (oder testweise per PowerShell/Curl) alle ~200 ms:

```
curl -X POST http://127.0.0.1:9000/update ` -H "Content-Type: application/json" ` -d '{ "npc": {
"x": 10, "y": 10 }, "players": [ { "id": "p1", "name": "Florian", "x": 16, "y": 13 } ] }'
```

Sobald  $\text{dist}(\text{npc}, \text{player}) \leq 8$  und der Cooldown abgelaufen ist, spricht der Bot den Spieler freundlich an.

---

## Optional/Nächster Schritt

- **Trigger auf Gesprochenes:** zusätzlich (wie in deiner ersten Idee) erst **aufnehmen & transkribieren**, wenn der Spieler nahe ist **und** im TS-Kanal jemand spricht (ClientQuery-Event).

- **Mehrere NPCs:** jedes NPC-Exemplar startet eine eigene Instanz dieses Scripts (oder Docker-Container) auf eigenem Port.

Wenn du mir sagst, wie dein Minimap-Backend am liebsten postet (WebSocket vs. REST), bau ich dir das 1:1 passend um.

**Du:**

ok kannst du mir sagen ob die app.py auch gestartet sein muss?

**ChatGPT:**

Sehr gute Nachfrage 👍

Es kommt darauf an, **welche Rolle app.py in deinem Projekt spielt:**

- Wenn app.py = deine **Minimap/Steuerungs-App** (FastAPI/Uvicorn), die die Positionen der Spieler verwaltet → **Ja, die muss laufen**, weil sie die Positionen sammelt/zeigt.
- Wenn wir den **NPC** starten (npc\_bot\_proximity.py), läuft der **eigene kleine FastAPI-Server** auf Port **9000**. Der wartet nur darauf, dass ihm jemand (z. B. deine Minimap-App) die **Positionen per /update** schickt.

👉 Das heißt:

- app.py: sorgt für die Minimap und schickt Positionsdaten an den NPC.
- npc\_bot\_proximity.py: sorgt für den NPC, der über die Nähe-Events reagiert und spricht.

Beide Prozesse laufen **parallel**.

---

**Kurz gesagt:**

- **Ja**, app.py **starten** → liefert Positionsdaten.
- **Ja**, npc\_bot\_proximity.py **starten** → reagiert auf diese Daten und redet im TS-Channel.

---

👉 Soll ich dir direkt ein Beispiel zeigen, wie du in deiner bestehenden app.py einen kleinen requests.post("http://127.0.0.1:9000/update", ...) einbauen kannst, damit die Minimap automatisch die Positionen an den NPC-Bot durchgibt?

**Du:**

ok sag mir was alles gestartet sein muss

**ChatGPT:**

Perfekt, lass uns das **Setup einmal vollständig auflisten**, damit du nie wieder überlegen musst, welche Bausteine laufen müssen 🚀.

---

## 1 Basis / Infrastruktur

- **TeamSpeak-Server**
  - Muss laufen (Channel „GTA“).

- **VB-Audio Cable (Input/Output)**
    - Beide Geräte in Windows auf **48 kHz** stellen.
- 

## 2 Clients

- **Dein normaler TeamSpeak-Client**
    - Aufnahmegerät = dein Mikro (z. B. RODECaster).
    - Wiedergabe = deine Kopfhörer/Lautsprecher.
  - **NPC-TeamSpeak-Client (zweite Instanz mit -nosingleinstance)**
    - Wiedergabe = **CABLE Input**
    - Aufnahme = **CABLE Output**
    - Aktivierung = *Dauersenden* oder niedrige Sprachaktivierung.
    - Dieser Client spricht später im Channel.
- 

## 3 Software-Komponenten

- **app.py (deine Minimap-Anwendung)**
    - Muss laufen, um Spieler- und NPC-Positionen zu verwalten.
    - Soll bei jeder Positionsänderung ein JSON an den NPC-Bot posten (/update).
  - **npc\_bot\_proximity.py**
    - Separates Script für den NPC-Bot.
    - Startet eigenen FastAPI-Server auf Port **9000**.
    - Bekommt Positionsdaten von app.py.
    - Kümmert sich um Proximity-Logik (wenn Spieler nahe → GPT-Reply → ElevenLabs → TS-Ausgabe).
- 

## 4 API-Keys

- .env muss im Projektordner (C:\kinpc) liegen mit:
  - OPENAI\_API\_KEY=dein\_openai\_key
  - ELEVENLABS\_API\_KEY=dein\_elevenlabs\_key
  - ELEVENLABS\_VOICE\_ID=21m00Tcm4TlvDq8ikWAM
- 

## 5 Start-Reihenfolge (empfohlen)

1. **TeamSpeak-Server** starten.
2. **VB-Audio Cable checken** (Systemsteuerung → 48 kHz).
3. **Dein TS-Client** öffnen, in Channel „GTA“ joinen.
4. **NPC-TS-Client** mit -nosingleinstance starten, in Channel „GTA“ joinen.
5. **app.py starten** (Minimap).

## 6. npc\_bot\_proximity.py **starten** (NPC-Bot).

---

👉 Ergebnis:

- Deine Minimap (app.py) schickt Positionsdaten → NPC-Bot (npc\_bot\_proximity.py).
  - Bot prüft Distanz → generiert Antwort → ElevenLabs TTS → spielt über CABLE Input → NPC-TS-Client sendet ins TS.
- 

Möchtest du, dass ich dir gleich den **Beispiel-Code** für app.py schreibe, damit er automatisch die Spieler-/NPC-Positionen an `http://127.0.0.1:9000/update` schickt?

**Du:**

in der app.py kommt: PS C:\kinpc> python app.py C:\kinpc\app.py:105: DeprecationWarning: on\_event is deprecated, use lifespan event handlers instead. Read more about it in the [FastAPI docs for Lifespan Events](https://fastapi.tiangolo.com/advanced/events/).  
@app.on\_event("startup") PS C:\kinpc>

**ChatGPT:**

Das, was du siehst, ist kein „echter“ Fehler ❌, sondern nur eine **Warnung** ⚠️ von FastAPI: @app.on\_event("startup") ist inzwischen als *deprecated* markiert.

Deine app.py läuft trotzdem ganz normal. 🚀

---

### Optionen

1. **Ignorieren**
  - Für dein aktuelles Projekt kannst du die Warnung getrost stehen lassen.
  - Funktioniert genauso wie vorher.
2. **Modernisieren** (nur nötig, wenn du Lust hast)  
FastAPI will statt @app.on\_event("startup") lieber ein Lifespan-Handler:

```
from contextlib import asynccontextmanager from fastapi import FastAPI
@asynccontextmanager async def lifespan(app: FastAPI): # Startup-Code hier print("Server
startet...") yield # Shutdown-Code hier print("Server fährt runter...") app =
FastAPI(lifespan=lifespan)
```

Dann kannst du @app.on\_event("startup") entfernen.

---

👉 **Kurz und knapp:**

Du kannst app.py einfach so weiter nutzen. Die Warnung ist rein informativ.

---

Willst du, dass ich dir deine app.py gleich so umbau, dass der startup-Code auf die neue Lifespan-Variante angepasst ist (Warnung verschwindet)?

**Du:**

ok aber die app.py startet nicht

**ChatGPT:**

Nachgedacht für 8s

Alles klar—wir bringen deine app.py sicher zum Laufen. Hier sind drei schnelle Wege, je nachdem, was du lieber willst.

### 1) So startest du sie zuverlässig

**Variante A (empfohlen):** mit uvicorn

```
pip install fastapi uvicorn uvicorn app:app --host 127.0.0.1 --port 8000 --reload
```

Das erwartet, dass in app.py eine Variable `app = FastAPI(...)` existiert.

**Variante B:** direkt per python app.py

Dann muss in der Datei unten ein `if __name__ == "__main__"` Block stehen (siehe Template).

### 2) Häufige Gründe, warum „nichts passiert“

- **uvicorn nicht installiert** → `pip install uvicorn`
- **falscher Startbefehl** (mit `python app.py` ohne `uvicorn.run(...)` im File)
- **Port blockiert** (Standard 8000):  
Fehler „address already in use“ → wechsele Port: `--port 8001`
- **Firewall blockt** → teste `http://127.0.0.1:8000/health`

### 3) Drop-in Minimal-app.py (startet sicher & modern)

Kopier das 1:1 in deine app.py. Es nutzt den neuen Lifespan-Handler (keine Deprecation-Warnung), hat `/health` und einen **/push**-Endpoint, der Positionsdaten an deinen NPC-Bot (`npc_bot_proximity.py` auf Port 9000) weiterleitet.

```
# app.py
from contextlib import asynccontextmanager
from typing import List, Dict, Any
import os
import requests
from fastapi import FastAPI, Body
NPC_URL = os.getenv("NPC_URL", "http://127.0.0.1:9000/update")

@asynccontextmanager
async def lifespan(app: FastAPI):
    print("Minimap-App startet ...")
    yield
    print("Minimap-App fährt herunter ...")

app = FastAPI(lifespan=lifespan)

@app.get("/health")
def health():
    return {"ok": True}

# Beispiel-Body: # { "npc": {"x": 10.2, "y": -3.4}, "players": [{"id": "p1", "name": "Florian", "x": 12.0, "y": -2.0}] }
@app.post("/push")
def push_positions(payload: Dict[str, Any] = Body(...)):
    # Eingabe minimal prüfen
    if "npc" not in payload:
        return {"ok": False, "error": "missing npc"}
    if "players" not in payload or not isinstance(payload["players"], list):
        payload["players"] = []
    # an den NPC-Bot weiterleiten
    try:
        r = requests.post(NPC_URL, json=payload, timeout=2.5)
        r.raise_for_status()
    except Exception as e:
        return {"ok": False, "error": f"forward failed: {e}"}

if __name__ == "__main__":
    # Start direkt mit: python app.py
    import uvicorn
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

### 4) Schnelltest

1. Start `npc_bot_proximity.py` (Port 9000).
2. Start app.py:

```
python app.py
```

oder

```
uvicorn app:app --host 127.0.0.1 --port 8000 --reload
```

3. Healthcheck:

4. `http://127.0.0.1:8000/health`

5. Positions-Ping (PowerShell):

```
curl -X POST http://127.0.0.1:8000/push ` -H "Content-Type: application/json" ` -d '{"npc": {"x": 10, "y": 10 }, "players": [{ "id": "p1", "name": "Florian", "x": 16, "y": 13 } ]}'
```

Sobald der Abstand  $\leq 8$  ist, sollte dein NPC sprechen (Cooldown 20s).

Wenn das so nicht startet, schick mir bitte genau die **Konsolen-Ausgabe** beim Start (und ob du mit `uvicorn app:app ...` oder `python app.py` startest). Dann fixen wir's gezielt in 1–2 Zeilen.

**Du:**

```
bekomme beim start von app.py PS C:\kinpc> uvicorn app:app --host 127.0.0.1 --port 8002 --
reload INFO: Will watch for changes in these directories: ['C:\kinpc'] INFO: Uvicorn running on
http://127.0.0.1:8002 (Press CTRL+C to quit) INFO: Started reloader process [8880] using
WatchFiles INFO: Started server process [40156] INFO: Waiting for application startup. INFO:
Application startup complete. INFO: 127.0.0.1:61498 - "GET /health HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:61498 - "GET /favicon.ico HTTP/1.1" 404 Not Found
```

**ChatGPT:**

Nachgedacht für 7s

Alles klar — dein Server läuft, aber die **Route /health existiert in deiner laufenden app.py nicht** (deshalb 404). Das heißt fast sicher: Uvicorn lädt **eine andere/ältere Datei** als die, in der du die Route definiert hast.

Mach's so, damit wir es eindeutig bekommen:

**Quick Fix: Mini-App mit /health**

Ersetze *testweise* den gesamten Inhalt von `C:\kinpc\app.py` durch dieses Minimalbeispiel:

```
# app.py from fastapi import FastAPI app = FastAPI() @app.get("/health") def health(): return {"ok": True}
```

Dann neu starten:

```
uvicorn app:app --host 127.0.0.1 --port 8002 --reload
```

Teste:

```
http://127.0.0.1:8002/health
```

→ Sollte **200** mit `{"ok": true}` liefern.

**Wenn es immer noch 404 ist**

Dann lädt Uvicorn nicht die Datei, die du denkst. Check:

1. Du startest im **richtigen Ordner**:

```
cd C:\kinpc dir app.py
```

2. Es gibt **keine zweite** app.py im Python-Pfad:

```
where app.py
```

3. Import erzwungen (um sicherzugehen, dass genau *diese* Datei importiert wird):

```
python -c "import os; os.chdir(r'C:\kinpc'); from app import app; import uvicorn;
uvicorn.run(app, host='127.0.0.1', port=8002)"
```

4. Uvicorn-Cache/Reloader frisch: Stoppe den Server (CTRL+C) und starte **ohne** --reload:

```
uvicorn app:app --host 127.0.0.1 --port 8002
```

Wenn der Minimaltest funktioniert, kannst du wieder deine größere Version (mit /push) nutzen. Falls du magst, schicke mir kurz den Inhalt deiner aktuellen app.py (nur die Routen + App-Erzeugung), dann sag ich dir genau, wo der Haken ist.

**Du:**

```
check das mal dass soll auf port 8002 laufen ! from __future__ import annotations import
asyncio, json, math, time, uuid from dataclasses import dataclass from typing import Dict from
fastapi import FastAPI, WebSocket, WebSocketDisconnect from fastapi.responses import
HTMLResponse MAP_WIDTH, MAP_HEIGHT = 800, 500 PLAYER_RADIUS, NPC_RADIUS = 10, 10
PLAYER_SPEED, TICK_RATE = 180.0, 30 @dataclass class Vec2: x: float; y: float def clamp(self,
w, h, r): self.x = max(r, min(w - r, self.x)) self.y = max(r, min(h - r, self.y)) return self @dataclass
class Actor: id: str; kind: str; pos: Vec2; radius: int @dataclass class PlayerState: actor: Actor;
input_keys: Dict[str, bool]; last_update_ts: float @dataclass class WorldState: width: int;
height: int; npc: Actor; players: Dict[str, PlayerState] def snapshot(self): return { "width": self.width,
"height": self.height, "npc":
{"id": self.npc.id, "kind": self.npc.kind, "x": self.npc.pos.x, "y": self.npc.pos.y, "r": self.npc.radius},
"players":
{pid: {"id": ps.actor.id, "kind": ps.actor.kind, "x": ps.actor.pos.x, "y": ps.actor.pos.y, "r": ps.actor.radius}
for pid, ps in self.players.items() } world = WorldState( width=MAP_WIDTH,
height=MAP_HEIGHT, npc=Actor(str(uuid.uuid4()), "npc", Vec2(MAP_WIDTH*0.8,
MAP_HEIGHT*0.5), NPC_RADIUS), players={} ) app = FastAPI(title="KI NPC - Mini-Map Service
(Schritt 1)") connections: Dict[str, WebSocket] = {} @app.get("/") async def index(): return
HTMLResponse(_HTML) @app.websocket("/ws") async def ws_endpoint(ws: WebSocket): await
ws.accept() cid = str(uuid.uuid4()) spawn = Vec2(MAP_WIDTH*0.2, MAP_HEIGHT*0.5) ps =
PlayerState( Actor(cid, "player", spawn, PLAYER_RADIUS), {"w": False, "a": False, "s": False, "d":
False}, time.perf_counter() ) world.players[cid] = ps connections[cid] = ws await
ws.send_json({"type": "hello", "client_id": cid}) try: while True: msg = await ws.receive_text() data
= json.loads(msg) if data.get("type") == "input": p = world.players.get(cid) if p: for k in
("w", "a", "s", "d"): if k in data["keys"]: p.input_keys[k] = bool(data["keys"][k]) except
WebSocketDisconnect: pass finally: world.players.pop(cid, None) connections.pop(cid, None)
async def game_loop(): dt = 1.0 / TICK_RATE while True: # update player positions for ps in
world.players.values(): dx = (-1 if ps.input_keys["a"] else 0) + (1 if ps.input_keys["d"] else 0) dy = (-
1 if ps.input_keys["w"] else 0) + (1 if ps.input_keys["s"] else 0) length = math.hypot(dx, dy) if
length: dx /= length; dy /= length ps.actor.pos.x += dx * PLAYER_SPEED * dt ps.actor.pos.y += dy *
PLAYER_SPEED * dt ps.actor.pos.clamp(MAP_WIDTH, MAP_HEIGHT, ps.actor.radius) #
```



```

broadcast world if connections: payload = json.dumps({"type": "state", "world":
world.snapshot()}) for pid, ws in list(connections.items()): try: await ws.send_text(payload)
except: connections.pop(pid, None) world.players.pop(pid, None) await asyncio.sleep(dt)
@app.on_event("startup") async def _startup(): asyncio.create_task(game_loop()) _HTML =
"""<!DOCTYPE html><html><head><meta charset=utf-8><title>Mini-Map</title>
<style>html,body{height:100%;margin:0;background:#0b1020;color:#e7ecff;font-family:sans-
serif} .wrap{display:flex;align-items:center;justify-content:center;height:100%}
.panel{width:920px;max-width:96vw;background:#0f1730;border:1px solid #1e2a55;border-
radius:16px;padding:20px;box-shadow:0 10px 30px rgba(0,0,0,.4)}
canvas{background:#0a0f24;border:1px solid #233266;border-
radius:12px;display:block;margin-top:12px} .row{display:flex;gap:16px;align-
items:center;justify-content:space-between} .mono{font-family:ui-
monospace,Consolas,Menlo,monospace} .pill{background:#12204a;border:1px solid
#1f2f66;padding:6px 10px;border-radius:999px} </style></head><body> <div class='wrap'><div
class='panel'> <div class='row'><div><h2 style='margin:0 0 4px'>Mini-Map Test</h2> <div
style='opacity:.8'>Controls: <span class='pill mono'>W</span> <span class='pill
mono'>A</span> <span class='pill mono'>S</span> <span class='pill mono'>D</span></div>
</div><div class='mono' id='client'>Connecting...</div></div> <canvas id='view' width='800'
height='500'></canvas> </div></div> <script> const
c=document.getElementById('view'),ctx=c.getContext('2d'); let
cid=null,world=null,keys={w:false,a:false,s:false,d:false}; const ws=new
WebSocket((location.protocol==='https:'?'wss':'ws')+':/'+location.host+'/ws');
ws.onmessage=e=>{const
m=JSON.parse(e.data);if(m.type==='hello'){cid=m.client_id;document.getElementById('client').t
extContent='Client: '+cid; } if(m.type==='state'){world=m.world;};
ws.onclose=()=>{document.getElementById('client').textContent='Disconnected';};
document.addEventListener('keydown',e=>{const k=e.key.toLowerCase();if(k in keys &&
!keys[k]){keys[k]=true;send();}}); document.addEventListener('keyup',e=>{const
k=e.key.toLowerCase();if(k in keys && keys[k]){keys[k]=false;send();}}); function send(){
if(ws.readyState===WebSocket.OPEN){ ws.send(JSON.stringify({type:'input',keys})); } } function
draw(){ ctx.clearRect(0,0,c.width,c.height); ctx.strokeStyle='#3850aa'; ctx.lineWidth=2;
ctx.strokeRect(1,1,c.width-2,c.height-2); if(!world){ requestAnimationFrame(draw); return; }
const n=world.npc; circle(n.x,n.y,n.r,'#ff6b6b'); for(const pid in world.players){ const
p=world.players[pid]; const me=(pid===cid); circle(p.x,p.y,p.r, me?'#7dd3fc':'#94a3b8'); if(me)
circle(p.x,p.y,p.r+4,'rgba(125,211,252,0.2)',true); } requestAnimationFrame(draw); } function
circle(x,y,r,col,fillOnly){ ctx.beginPath(); ctx.arc(x,y,r,0,Math.PI*2); if(fillOnly){ ctx.fillStyle=col;
ctx.fill(); } else { ctx.fillStyle=col; ctx.strokeStyle=col; ctx.lineWidth=2; ctx.fill(); ctx.stroke(); } }
draw(); </script></body></html>"""

```

### ChatGPT:

Nachgedacht für 4 Sekunden

Top – deine app.py ist sauber. Sie hat:

- GET / → liefert das HTML mit Canvas (Mini-Map UI)
- WS /ws → WebSocket für Eingaben/State
- einen Game-Loop via @app.on\_event("startup")

Es gibt **keine** /health-Route, daher kam davor 404. Das ist okay.

## So startest du sie auf Port 8002

```
cd C:\kinpc uvicorn app:app --host 127.0.0.1 --port 8002 --reload
```

Dann im Browser: <http://127.0.0.1:8002/>

(Das HTML nutzt location.host, also verbindet sich der WebSocket automatisch zu <ws://127.0.0.1:8002/ws>.)

---

## NPC verbinden (Positionsdaten an den Bot schicken)

Deine app.py broadcastet den Weltzustand an Browser, **schickt aber noch nichts** an den NPC-Bot (npc\_bot\_proximity.py lauscht auf <http://127.0.0.1:9000/update>). Füge eine **asynchrone HTTP-POST** in den Game-Loop ein.

### 1) httpx installieren

```
pip install httpx
```

### 2) Oben in deiner app.py ergänzen

```
import httpx NPC_URL = "http://127.0.0.1:9000/update"
```

### 3) Hilfsfunktion einbauen (direkt über async def game\_loop():)

```
_last_push_ts = 0.0 async def _push_positions(): global _last_push_ts # max. alle 0.2s senden  
now = time.perf_counter() if now - _last_push_ts < 0.2: return _last_push_ts = now payload =  
world.snapshot() # world.snapshot() liefert dict mit "npc" und "players" bereits im richtigen  
Format async with httpx.AsyncClient(timeout=1.5) as client: try: await client.post(NPC_URL,  
json={"npc": {"x": payload["npc"]["x"], "y": payload["npc"]["y"]}, "players": [ {"id": pid, "name":  
pid, "x": p["x"], "y": p["y"]} for pid, p in payload["players"].items() ] }) except Exception: # still  
schweigen – der NPC-Bot kann auch mal aus sein pass
```

### 4) Im game\_loop() am Ende vor await asyncio.sleep(dt) aufrufen

```
# ... nach dem Broadcast await _push_positions() await asyncio.sleep(dt)
```

Fertig: Jetzt sendet die Minimap ~5×/Sek. die Positionen an den NPC-Bot. Sobald du dich auf  $\leq 8$  m nherst (Default im Bot), sollte er dich ansprechen.

---

## Optional: Deprecation beseitigen (spter)

Die Warnung zu `@app.on_event("startup")` ist nur ein Hinweis. Wenn du magst, kann ich sie in eine moderne Lifespan-Variante umbauen – fr den Durchstich ist's nicht ntig.

---

Wenn du willst, geb' ich dir deine app.py auch einmal komplett mit den obigen nderungen zusammengefgt – sag kurz Bescheid.

## Du:

kannst du mir bitte sowohl die korrekte app.py geben, wenn es hier nderungen geben soll sowohl die die anderen .py wenn es nderungen gibt ! Aber bitte komplett: Will keine einzelnen

änderungen machen! Dann bitte eine Datei mit der ich die vollständige Anwendung starten kann ohne einzeln irgendwas zu starten! Will eine .bat haben die ich start und alles startet komplett

### ChatGPT:

Nachgedacht für 7s

alles klar — hier bekommst du **komplette Dateien** (Copy/Paste), inkl. **.bat**, damit du alles mit **einem Doppelklick** startest.

---

#### 1) .env (im selben Ordner)

```
OPENAI_API_KEY=dein_openai_key
ELEVENLABS_API_KEY=dein_elevenlabs_key
ELEVENLABS_VOICE_ID=21m00Tcm4TlvDq8ikWAM
```

---

#### 2) requirements.txt

```
fastapi
uvicorn
httpx
python-dotenv
requests
numpy
sounddevice
soundfile
scipy
```

---

#### 3) app.py (Mini-Map + Positions-Forwarding an den NPC auf Port 9000)

```
from __future__ import annotations
import asyncio, json, math, time, uuid
from dataclasses import dataclass
from typing import Dict
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from fastapi.responses import HTMLResponse
import httpx

MAP_WIDTH, MAP_HEIGHT = 800, 500
PLAYER_RADIUS, NPC_RADIUS = 10, 10
PLAYER_SPEED, TICK_RATE = 180.0, 30
NPC_URL = "http://127.0.0.1:9000/update"

@dataclass
class Vec2:
    x: float
    y: float

    def clamp(self, w, h):
        self.x = max(0, min(w, self.x))
        self.y = max(0, min(h, self.y))

    def return(self):
        return self

@dataclass
class Actor:
    id: str
    kind: str
    pos: Vec2
    radius: int

@dataclass
class PlayerState:
    actor: Actor
    input_keys: Dict[str, bool]
    last_update_ts: float

@dataclass
class WorldState:
    width: int
    height: int
    npc: Actor
    players: Dict[str, PlayerState]

    def snapshot(self):
        return {
            "width": self.width,
            "height": self.height,
            "npc": {
                "id": self.npc.id,
                "kind": self.npc.kind,
                "x": self.npc.pos.x,
                "y": self.npc.pos.y,
                "r": self.npc.radius,
            },
            "players": {
                pid: {
                    "id": ps.actor.id,
                    "kind": ps.actor.kind,
                    "x": ps.actor.pos.x,
                    "y": ps.actor.pos.y,
                    "r": ps.actor.radius,
                }
            }
        }
```

```

for pid,ps in self.players.items() } world = WorldState( width=MAP_WIDTH,
height=MAP_HEIGHT, npc=Actor(str(uuid.uuid4()), "npc", Vec2(MAP_WIDTH*0.8,
MAP_HEIGHT*0.5), NPC_RADIUS), players={} ) app = FastAPI(title="KI NPC - Mini-Map Service")
connections: Dict[str, WebSocket] = {} _last_push_ts = 0.0 @app.get("/") async def index(): return
HTMLResponse(_HTML) @app.websocket("/ws") async def ws_endpoint(ws: WebSocket): await
ws.accept() cid = str(uuid.uuid4()) spawn = Vec2(MAP_WIDTH*0.2, MAP_HEIGHT*0.5) ps =
PlayerState( Actor(cid, "player", spawn, PLAYER_RADIUS), {"w": False, "a": False, "s": False, "d":
False}, time.perf_counter() ) world.players[cid] = ps connections[cid] = ws await
ws.send_json({"type": "hello", "client_id": cid}) try: while True: msg = await ws.receive_text() data
= json.loads(msg) if data.get("type") == "input": p = world.players.get(cid) if p: for k in
("w","a","s","d"): if k in data["keys"]: p.input_keys[k] = bool(data["keys"][k]) except
WebSocketDisconnect: pass finally: world.players.pop(cid, None) connections.pop(cid, None)
async def _push_positions(): global _last_push_ts now = time.perf_counter() if now -
_last_push_ts < 0.2: # max ~5/s return _last_push_ts = now snap = world.snapshot() payload = {
"npc": {"x": snap["npc"]["x"], "y": snap["npc"]["y"]}, "players": [ {"id": pid, "name": pid, "x": p["x"],
"y": p["y"]} for pid, p in snap["players"].items() ] } try: async with httpx.AsyncClient(timeout=1.5)
as client: await client.post(NPC_URL, json=payload) except Exception: # NPC-Bot darf aus sein
- still weiterlaufen pass async def game_loop(): dt = 1.0 / TICK_RATE while True: # update player
positions for ps in list(world.players.values()): dx = (-1 if ps.input_keys["a"] else 0) + (1 if
ps.input_keys["d"] else 0) dy = (-1 if ps.input_keys["w"] else 0) + (1 if ps.input_keys["s"] else 0)
length = math.hypot(dx, dy) if length: dx /= length; dy /= length ps.actor.pos.x += dx *
PLAYER_SPEED * dt ps.actor.pos.y += dy * PLAYER_SPEED * dt ps.actor.pos.clamp(MAP_WIDTH,
MAP_HEIGHT, ps.actor.radius) # broadcast world if connections: payload = json.dumps({"type":
"state", "world": world.snapshot()}) for pid, ws in list(connections.items()): try: await
ws.send_text(payload) except: connections.pop(pid, None) world.players.pop(pid, None) await
_push_positions() await asyncio.sleep(dt) @app.on_event("startup") async def _startup():
asyncio.create_task(game_loop()) _HTML = """<!DOCTYPE html><html><head><meta
charset=utf-8><title>Mini-Map</title>
<style>html,body{height:100%;margin:0;background:#0b1020;color:#e7ecff;font-family:sans-
serif}.wrap{display:flex;align-items:center;justify-content:center;height:100%}
.panel{width:920px;max-width:96vw;background:#0f1730;border:1px solid #1e2a55;border-
radius:16px;padding:20px;box-shadow:0 10px 30px rgba(0,0,0,.4)}
.canvas{background:#0a0f24;border:1px solid #233266;border-
radius:12px;display:block;margin-top:12px}.row{display:flex;gap:16px;align-
items:center;justify-content:space-between}.mono{font-family:ui-
monospace,Consolas,Menlo,monospace}.pill{background:#12204a;border:1px solid
#1f2f66;padding:6px 10px;border-radius:999px} </style></head><body> <div class='wrap'><div
class='panel'> <div class='row'><div><h2 style='margin:0 0 4px'>Mini-Map Test</h2> <div
style='opacity:.8'>Controls: <span class='pill mono'>W</span> <span class='pill
mono'>A</span> <span class='pill mono'>S</span> <span class='pill mono'>D</span></div>
</div><div class='mono' id='client'>Connecting...</div></div> <canvas id='view' width='800'
height='500'></canvas> </div></div> <script> const
c=document.getElementById('view'),ctx=c.getContext('2d'); let
cid=null,world=null,keys={w:false,a:false,s:false,d:false}; const ws=new
WebSocket((location.protocol==='https'? 'wss': 'ws')+'://'+location.host+'/ws');
ws.onmessage=e=>{const
m=JSON.parse(e.data);if(m.type==='hello'){cid=m.client_id;document.getElementById('client').t
extContent='Client: '+cid; } if(m.type==='state'){world=m.world;}};
ws.onclose=()=>{document.getElementById('client').textContent='Disconnected';};

```

```
document.addEventListener('keydown',e=>{const k=e.key.toLowerCase();if(k in keys &&
!keys[k]){keys[k]=true;send();}}); document.addEventListener('keyup',e=>{const
k=e.key.toLowerCase();if(k in keys && keys[k]){keys[k]=false;send();}}); function send(){
if(ws.readyState===WebSocket.OPEN){ ws.send(JSON.stringify({type:'input',keys})); } } function
draw(){ ctx.clearRect(0,0,c.width,c.height); ctx.strokeStyle='#3850aa'; ctx.lineWidth=2;
ctx.strokeRect(1,1,c.width-2,c.height-2); if(!world){ requestAnimationFrame(draw); return; }
const n=world.npc; circle(n.x,n.y,n.r,'#ff6b6b'); for(const pid in world.players){ const
p=world.players[pid]; const me=(pid===cid); circle(p.x,p.y,p.r, me?'#7dd3fc':'#94a3b8'); if(me)
circle(p.x,p.y,p.r+4,'rgba(125,211,252,0.2)',true); } requestAnimationFrame(draw); } function
circle(x,y,r,col,fillOnly){ ctx.beginPath(); ctx.arc(x,y,r,0,Math.PI*2); if(fillOnly){ ctx.fillStyle=col;
ctx.fill(); } else { ctx.fillStyle=col; ctx.strokeStyle=col; ctx.lineWidth=2; ctx.fill(); ctx.stroke(); } }
draw(); </script></body></html>"""
```

---

#### 4) npc\_bot\_proximity.py (NPC reagiert bei Nähe; feste VB-Cable-IDs 41/31 + Resampling)

```
# npc_bot_proximity.py # Minimap -> POST /update mit Positionen # NPC begrüßt Spieler bei
Nähe (GPT -> ElevenLabs -> Wiedergabe über CABLE Input) import os, time, math, threading
import numpy as np import requests import sounddevice as sd import soundfile as sf from
scipy.signal import resample_poly from dotenv import load_dotenv from fastapi import FastAPI,
Request import uvicorn # ===== Audio-Setup (deine festen IDs) ===== IN_DEVICE_ID = 41 #
CABLE Output (2 In / 0 Out) <- aufnehmen wäre optional OUT_DEVICE_ID = 31 # CABLE Input (0
In / 2 Out) <- hier spielt der Bot TARGET_SR = 48000 # Stelle CABLE Input/Output in Windows auf
48 kHz # ===== Bot-Verhalten ===== GREET_RADIUS = 8.0 # Meter COOLDOWN_SEC = 20
SYSTEM_PROMPT = ( "Du bist ein freundlicher NPC in einer minimalistischen GTA-ähnlichen
Welt. " "Antworte sehr kurz (1–2 Sätze), locker und auf Deutsch." ) # ===== State =====
positions_lock = threading.Lock() npc_pos = (0.0, 0.0) players = {} # id -> {"name": str, "x": float,
"y": float, "last_greet": float} # ===== Helpers ===== def dist(a, b): return math.hypot(a[0] - b[0],
a[1] - b[1]) def to_stereo(data: np.ndarray) -> np.ndarray: if data.ndim == 1: data = data[:, None] if
data.shape[1] == 1: data = np.repeat(data, 2, axis=1) return data def resample_if_needed(data:
np.ndarray, sr_in: int, sr_target: int): if sr_in == sr_target: return data, sr_in g =
math.gcd(int(sr_in), int(sr_target)) up, down = int(sr_target // g), int(sr_in // g) return
resample_poly(data, up, down, axis=0), sr_target def require_env(name: str) -> str: val =
os.getenv(name, "").strip() if not val: raise RuntimeError(f"{name} fehlt in .env") return val #
===== APIs ===== def openai_chat_reply(prompt_text: str, api_key: str) -> str: payload = {
"model": "gpt-4o-mini", "messages": [ {"role": "system", "content": SYSTEM_PROMPT}, {"role":
"user", "content": prompt_text} ], "temperature": 0.6, } r = requests.post(
"https://api.openai.com/v1/chat/completions", headers={"Authorization": f"Bearer {api_key}",
"Content-Type": "application/json"}, json=payload, timeout=60 ) r.raise_for_status() return
r.json()["choices"][0]["message"]["content"].strip() def elevenlabs_tts_to_wav(text: str, api_key:
str, voice_id: str, out_path_wav: str): url = f"https://api.elevenlabs.io/v1/text-to-
speech/{voice_id}" headers = {"xi-api-key": api_key.strip(), "accept": "audio/wav", "Content-
Type": "application/json"} body = {"text": text, "model_id": "eleven_multilingual_v2",
"voice_settings": {"stability": 0.5, "similarity_boost": 0.8}} resp = requests.post(url,
headers=headers, json=body, timeout=120) if resp.status_code == 401: raise
RuntimeError("ElevenLabs 401 Unauthorized – API-Key/Voice-ID prüfen.") if resp.status_code ==
404: raise RuntimeError("ElevenLabs 404 – Voice-ID unbekannt/nicht zugelassen.")
resp.raise_for_status() with open(out_path_wav, "wb") as f: f.write(resp.content) def
speak_text(text: str, eleven_key: str, voice_id: str): wav_out = "npc_reply.wav"
```

```

elevenlabs_tts_to_wav(text, eleven_key, voice_id, wav_out) data, sr_in = sf.read(wav_out,
dtype="float32", always_2d=True) data, sr_play = resample_if_needed(data, sr_in, TARGET_SR)
data = to_stereo(data) sd.play(data, sr_play, device=OUT_DEVICE_ID, blocking=True) # =====
Proximity Watcher ===== def proximity_loop(openai_key: str, eleven_key: str, voice_id: str): while
True: time.sleep(0.2) now = time.time() with positions_lock: my_pos = npc_pos snapshot = [(pid,
pdata["name"], (pdata["x"], pdata["y"]), pdata.get("last_greet", 0)) for pid, pdata in
players.items()] for pid, pname, ppos, last in snapshot: if dist(my_pos, ppos) <= GREET_RADIUS
and (now - last) > COOLDOWN_SEC: user_text = f"Begrüße den Spieler '{pname}' freundlich und
kurz (max. 2 Sätze)." try: reply = openai_chat_reply(user_text, openai_key) speak_text(reply,
eleven_key, voice_id) except Exception as e: print("Greet-Fehler:", e) with positions_lock:
players[pid]["last_greet"] = time.time() # ===== FastAPI App ===== app = FastAPI()
@app.post("/update") async def update_positions(req: Request): body = await req.json() npc =
body.get("npc") or {} plist = body.get("players") or [] with positions_lock: if "x" in npc and "y" in
npc: global npc_pos npc_pos = (float(npc["x"]), float(npc["y"])) for p in plist: pid = str(p.get("id",
"")) if not pid: continue name = p.get("name") or pid x = float(p.get("x", 0.0)); y = float(p.get("y",
0.0)) entry = players.get(pid, {"name": name, "x": x, "y": y, "last_greet": 0.0})
entry.update({"name": name, "x": x, "y": y}) players[pid] = entry return {"ok": True} def main():
load_dotenv() openai_key = require_env("OPENAI_API_KEY") eleven_key =
require_env("ELEVENLABS_API_KEY") voice_id = os.getenv("ELEVENLABS_VOICE_ID",
"21m00Tcm4TlvDq8ikWAM").strip() or "21m00Tcm4TlvDq8ikWAM" dev_in =
sd.query_devices(IN_DEVICE_ID) dev_out = sd.query_devices(OUT_DEVICE_ID) print(f"[Input ]
id={IN_DEVICE_ID} -> {dev_in['name']} | default_sr={dev_in.get('default_samplerate')}")
print(f"[Output] id={OUT_DEVICE_ID} -> {dev_out['name']} |
default_sr={dev_out.get('default_samplerate')}") print(f"Target SR: {TARGET_SR} Hz") print("HTTP-
Endpoint: POST http://127.0.0.1:9000/update") t = threading.Thread(target=proximity_loop,
args=(openai_key, eleven_key, voice_id), daemon=True) t.start() uvicorn.run(app,
host="127.0.0.1", port=9000) if __name__ == "__main__": main()

```

---

## 5) run\_all.bat (ein Doppelklick startet alles Nötige)

```

@echo off setlocal ENABLEDELAYEDEXPANSION REM ===== Pfad in Skript-Ordner ===== cd /d
"%~dp0" REM ===== Optional: TeamSpeak-Pfade (auskommentiert lassen, wenn du manuell
startest) REM set TS_PATH="C:\Program Files\TeamSpeak 3 Client\ts3client_win64.exe" REM if
exist %TS_PATH% ( REM echo Starte TeamSpeak-Client 1... REM start "TS-Client 1" %TS_PATH%
REM echo Starte TeamSpeak-Client 2 (NPC)... REM start "TS-Client NPC" %TS_PATH% -
nosingleinstance REM ) else ( REM echo Hinweis: Passe TS_PATH in run_all.bat an, um TS-
Clients automatisch zu starten. REM ) REM ===== Python + Abhaengigkeiten sicherstellen =====
where python >nul 2>nul if errorlevel 1 ( echo Fehler: "python" nicht im PATH. Bitte Python
installieren und neu versuchen. pause exit /b 1 ) if not exist "venv" ( echo Erstelle virtuelle
Umgebung... python -m venv venv ) call venv\Scripts\activate echo Installiere/aktualisiere
Abhaengigkeiten... pip install --upgrade pip >nul pip install -r requirements.txt REM ===== .env
vorhanden? ===== if not exist ".env" ( echo. echo Fehler: .env nicht gefunden. Bitte Datei ".env"
mit deinen API-Keys anlegen. echo Beispiel: echo OPENAI_API_KEY=sk-... echo
ELEVENLABS_API_KEY=... echo ELEVENLABS_VOICE_ID=21m00Tcm4TlvDq8ikWAM pause exit
/b 1 ) REM ===== Starte NPC-Bot (Port 9000) ===== echo Starte NPC-Bot... start "NPC Bot" cmd
/k "venv\Scripts\python.exe npc_bot_proximity.py" REM ===== Starte Mini-Map (Port 8002)
===== echo Starte Mini-Map... start "Mini-Map" cmd /k "venv\Scripts\python.exe -m uvicorn
app:app --host 127.0.0.1 --port 8002 --reload" echo. echo -----

```

echo Alles gestartet: echo - NPC Bot: <http://127.0.0.1:9000/update> (lauschend) echo - Mini-Map: <http://127.0.0.1:8002/> echo ----- echo Bitte stelle in TeamSpeak ein: echo - NPC-Client: Wiedergabe = CABLE Input, Aufnahme = CABLE Output, Dauersenden echo - Dein Client: normales Mikro/Lautsprecher echo. pause

---

### Was du tun musst

1. Dateien in **denselben Ordner** legen (z. B. C:\kinpc).
2. .env mit deinen Keys befüllen.
3. **Doppelklick auf** run\_all.bat.
4. Browser: <http://127.0.0.1:8002/> öffnen, mit WASD bewegen.
5. In TeamSpeak:
  - Du im Channel „GTA“.
  - **NPC-Client** (2. Instanz) im selben Channel, **Wiedergabe=CABLE Input, Aufnahme=CABLE Output, 48 kHz**.
6. Wenn du dich auf  $\leq 8\text{m}$  nherst, spricht der Bot dich an (Cooldown 20s).

Wenn du willst, passe ich dir noch GREET\_RADIUS/COOLDOWN\_SEC an — oder wir ergnzen einen kleinen /config-Endpoint frs Live-Tuning.

ChatGPT kann Fehler machen. berprfe wichtige Informationen. Siehe Cookie-Voreinstellungen.