Documentation Technique – Partie IA

Projet GameForge – Générateur de Jeux Vidéo

Auteur : Lyes Ait Tayeb

Introduction

La partie IA du projet GameForge est dédiée à la génération de contenu créatif (texte et images) à
l'aide de modèles hébergés sur la plateforme Hugging Face. Elle permet de générer automatiquement un univers narratif, des personnages et des visuels conceptuels.

Configuration & Variables

Le fichier utilise des variables d'environnement pour stocker les informations sensibles : -
HF_TOKEN : clé API Hugging Face - TEXT_MODEL : modèle utilisé pour le texte (par défaut
mistralai/Mistral-7B-v0.1) - IMG_MODEL : modèle utilisé pour l'image (par défaut stabilityai/stable-diffusion-2-1) Cette configuration rend le code flexible et sécurisé.

Fonction _hf_post()

Cette fonction centralise l'appel HTTP POST à l'API Hugging Face. Elle construit l'URL, ajoute les
headers (avec le token d'authentification), et gère les éventuelles erreurs en utilisant raise_for_status(). Elle permet d'éviter la duplication de code.

Fonction generate_structured_game()

Cette fonction est le cœur de la génération narrative. Elle construit un prompt en français
demandant un JSON structuré contenant : - Universe : description courte de l'univers - Scenario :
structuré en 3 actes - Twist : retournement narratif - Characters : liste de personnages - Locations :
lieux emblématiques - Pitch : phrases marketing Ensuite, la fonction appelle Hugging Face, puis
tente de parser la réponse : - Si Hugging Face renvoie un JSON bien formé, il est directement
utilisé. - Si la réponse contient du bruit (texte, markdown, backticks), le code extrait uniquement le
bloc JSON. - Si le parsing échoue, on renvoie un dictionnaire brut avec 'raw_text'.

Fonction generate_concept_image()

Cette fonction génère des visuels à partir d'un prompt. Elle appelle le modèle image Hugging Face
et gère deux cas : - Si la réponse est une image binaire (content-type image/png), elle est directement chargée avec PIL. - Si la réponse est un JSON contenant une image

encodée en
base64, on la décode et on la convertit. Cela garantit une compatibilité avec différents modèles
d'images.

Fonction random_seed_game()

Cette fonction génère aléatoirement des paramètres de jeu (titre, genre, ambiance, mots-clés,
références). Elle est utile pour tester rapidement le système ou donner de l'inspiration à l'utilisateur.

Problèmes rencontrés & Solutions

- Répétition du même texte : certains modèles produisent des sorties similaires si le prompt reste
identique. Solution : ajouter du hasard (ex. température, seed aléatoire). - JSON invalide : les
modèles peuvent ajouter du texte non-JSON. Solution : extraction et nettoyage du bloc JSON avant
parsing. - Images identiques : si le modèle image n'utilise pas de seed, il peut répéter la même
sortie. Solution : ajouter un seed aléatoire dans la génération d'image.

Conclusion & Améliorations

Cette partie IA est un élément central de GameForge, car elle automatise la création de contenus.
Cependant, elle peut être améliorée par : - L'utilisation de modèles plus spécialisés pour la
génération JSON (ex. LLaMA 2 instruct). - L'ajout d'un paramètre 'seed' explicite pour diversifier les
images. - La mise en cache des résultats pour éviter de payer plusieurs fois les mêmes générations. - L'intégration d'une validation automatique du JSON (schémas).

Explication détaille du code

1.Importations

```
import os, io, json, random, base64
from typing import Dict, Any
import requests
from PIL import Image
```

os → permet de lire les variables d’environnement (comme le token HuggingFace).
io → manipulation des flux binaires (utilisé pour ouvrir les images).

json → sérialisation et désérialisation JSON (les modèles HF renvoient souvent du JSON ou du texte
structuré).
random → génération aléatoire (utile pour varier les seeds de génération).
base64 → décodage des images renvoyées parfois encodées en Base64 par HF.
typing.Dict, Any → typage Python (meilleure lisibilité).
requests → librairie HTTP pour appeler l'API Hugging Face.
PIL.Image → permet de manipuler les images générées.

2.Configuration des modèles et API

HF_TOKEN = os.environ.get("HUGGINGFACE_API_TOKEN", "")
TEXT_MODEL = os.environ.get("HF_TEXT_MODEL", "mistralai/Mistral-7B-v0.1")
IMG_MODEL = os.environ.get("HF_IMG_MODEL", "stabilityai/stable-diffusion-2-1")
API_BASE = "https://api-inference.huggingface.co/models"
HEADERS = {"Authorization": f"Bearer {HF_TOKEN}"}
HF_TOKEN → récupère le token d'API dans les variables d'environnement.
TEXT_MODEL → modèle IA texte par défaut (Mistral 7B).
IMG_MODEL → modèle IA image par défaut (Stable Diffusion v2.1).
API_BASE → URL de base de l'API Hugging Face.
HEADERS → inclut l'authentification via Bearer token.
Sans HF_TOKEN, aucune requête ne fonctionnera.

3. Fonction interne _hf_post

def _hf_post(model: str, payload: Dict[str, Any], stream: bool = False):
url = f"{API_BASE}/{model}"
resp = requests.post(url, headers=HEADERS, json=payload, stream=stream,
timeout=120)
resp.raise_for_status()
return resp
Envoie une requête POST à un modèle Hugging Face.
Paramètres :
model → nom du modèle (ex: "mistralai/Mistral-7B-v0.1").
payload → le prompt et les paramètres de génération.
stream → si True, reçoit la réponse en flux (utile pour images).
timeout=120 → évite les blocages.
Retour → un objet Response de requests.

4.Génération de texte structuré (generate_structured_game)

def generate_structured_game(title: str, genre: str, ambiance: str, keywords: str,

references: str) ->
Dict[str, Any]:prompt = f""" ... """
Étapes :
Construction du prompt :
Le texte demande explicitement un JSON strict avec les champs :
"universe" : description de l'univers.
"scenario" : découpé en act1, act2, act3.
"twist" : retournement narratif.
"characters" : liste de 2 à 4 personnages.
"locations" : lieux emblématiques.
"pitch" : mini-pitch marketing.
Le modèle doit répondre uniquement en JSON.
Payload envoyé à HF :
payload = {
"inputs": prompt.strip(),
"parameters": {"max_new_tokens": 600, "temperature": 0.8, "return_full_text": False},
"options": {"wait_for_model": True},
}
max_new_tokens → limite de sortie.
temperature=0.8 → plus la valeur est haute, plus la génération est créative.
return_full_text=False → évite que le prompt soit répété dans la réponse.
wait_for_model=True → force Hugging Face à attendre le chargement du modèle.
Nettoyage de la réponse :
Vérifie si Hugging Face renvoie un JSON brut ou un texte.
Si bruit autour du JSON, extraction entre { ... }.
Si parsing impossible → retour sous forme de dictionnaire {"raw_text": ...}.
Retour : un dict représentant le jeu généré.

5. Génération d'images (generate_concept_image)

def generate_concept_image(prompt: str) -> Image.Image:
payload = {"inputs": prompt, "options": {"wait_for_model": True}}
resp = _hf_post(IMG_MODEL, payload, stream=True)
Étapes :
Envoie le prompt au modèle stable-diffusion.
Cas possibles :
Hugging Face renvoie une image binaire → convertie en PIL.Image.
Hugging Face renvoie du JSON contenant du base64 → décodé, puis converti.
Si aucun cas ne correspond → lève une RuntimeError.
Retour : un objet PIL.Image.

6. Génération aléatoire (random_seed_game)

```
def random_seed_game() -> Dict[str, str]:
genres = ["RPG", "FPS", "Metroidvania", "Visual Novel", "Rogue-lite", "Tactique"]
...
return {
"title": f"Proto-{random.randint(1000,9999)}",
"genre": random.choice(genres),
...}
```

Tire au hasard :
un genre (RPG, FPS...),
une ambiance (cyberpunk, dark fantasy...),
des mots-clés,
une référence culturelle.
Retour → dictionnaire prêt à être envoyé au modèle.

Résumé

generate_structured_game() → génère un JSON complet décrivant un jeu.
generate_concept_image() → génère une image conceptuelle à partir d'un prompt.
random_seed_game() → crée un jeu aléatoire pour le mode exploration libre.
_hf_post() → fonction générique pour appeler Hugging Face