Akinator-2

Dans cette deuxième étape, nous allons rendre ce projet un peu plus convivial en implémentant une interface graphique grâce à la bibliothèque **Tkinter**.

Tkinter est une bibliothèque intégrée à Python qui permet de créer des interfaces graphiques (GUI).
Elle permet de :

- Afficher des fenêtres (comme les fenêtres d'une application classique)
- Ajouter des boutons, textes, images, zones de saisie, etc.
- Gérer des interactions utilisateur : clics, entrées clavier, etc.

Notre jeu Mini Akinator avait d'abord une version en console, mais grâce à Tkinter, on peut maintenant le rendre plus moderne et accessible avec une interface graphique :

- L'utilisateur voit les questions affichées en grand
- Il répond avec des boutons (✅ Oui / ❌ Non)
- Une boîte de dialogue s'affiche pour apprendre en cas d'échec
- Et il peut facilement relancer le jeu avec un bouton 🔄

Structure d'une interface Tkinter

Création d'une fenêtre principale:

```
root = tk.Tk()              # Crée la fenêtre principale
root.geometry("500x400")    # Définit la taille
root.mainloop()             # Démarre la boucle d'événements
```

Ajout d'un widget
Un widget est un élément visuel : bouton, label, etc.

```
label = tk.Label(root, text="Bienvenue !")
label.pack()
```

Tkinter dans notre projet Akinator

Classe AkinatorGUI
On regroupe toute l'interface dans une classe objet pour bien organiser le code.

```
class AkinatorGUI:
    def __init__(self, master):
        ...
```

Élément	Rôle dans le jeu	Code associé
tk.Label	Affiche les questions et réponses	self.label = tk.Label(...)
tk.Button	Boutons pour "Oui", "Non" et	tk.Button(master, text="✅ Oui", ...)
simpledialog	Demande à l'utilisateur du texte (nouvelle	simpledialog.askstring(...)
messagebox	Affiche des messages d'information	messagebox.showinfo(...)

Le déroulement du jeu avec l'interface

<u>Étape 1 – Démarrer</u>
Le bouton "▶️ Démarrer" appelle start_game()
On affiche la première question ou réponse

```
def start_game(self):
    self.current_node = self.root
    self.label.config(text=self.get_text(self.current_node))
```

Étape 2 – Répondre

Quand l'utilisateur clique sur ✅ Oui ou ❌ Non, on appelle answer() :

```
def answer(self, is_yes):
    if self.current_node.is_leaf():
        # L'IA propose une réponse
        # Si erreur → apprendre
    else:
        # On continue dans l'arbre
        self.current_node = self.current_node.yes if is_yes else self.current_node.no
```

Étape 3 – Apprendre

Si l'IA s'est trompée :

- On demande la bonne réponse (simpledialog)
- On demande une question pour distinguer les deux (simpledialog)
- On demande si c'est "oui" ou "non" pour la bonne réponse (messagebox.askyesno)
- On modifie l'arbre pour intégrer ce nouveau savoir

Interface conviviale

Le projet utilise :

- Des boutons clairs et visuels (émoticônes)
- Une taille fixe (root.geometry("500x400"))
- Une police lisible (Helvetica 14 ou 16)
- Des marges avec pack(pady=...) pour aérer l'interface

Élément	Rôle
Tk()	Crée la fenêtre principale
Label	Affiche les questions/répons
Button	Permet de cliquer sur "Oui", "Non",
askstring()	Demande une réponse
askyesno()	Pose une question Oui/Non à
messagebox.showinfo	Affiche une notification

Analyse du programme bloc par bloc

```
import json
import os
import tkinter as tk
from tkinter import simpledialog, messagebox
```

json : sert à sauvegarder et charger l'arbre du jeu (question/réponse) sous forme de fichier .json.
os : permet de vérifier si le fichier JSON existe.
tkinter : pour construire l'interface graphique.
simpledialog : pour demander du texte à l'utilisateur.
messagebox : pour afficher des messages dans une boîte de dialogue

```
# Classe représentant un nœud de l'arbre de questions/réponses
class Node:
    def __init__(self, question=None, answer=None):
        # Une question (pour un nœud interne) ou une réponse (pour une feuille)
        self.question = question
        self.answer = answer
        # Les branches enfants (oui / non)
        self.yes = None
        self.no = None

    # Vérifie si ce nœud est une feuille (donc contient une réponse)
    def is_leaf(self):
        return self.answer is not None
```

Un nœud est soit :

- une question (avec deux branches : oui et non),
- soit une réponse finale (feuille de l'arbre).

is_leaf() renvoie True si le nœud contient une réponse (et donc pas de question à poser).

```python
# Sérialisation de l'arbre dans un dictionnaire JSON
def serialize(node):
    if node is None:
        return None
    # Si c'est une feuille, on stocke juste la réponse
    if node.is_leaf():
        return {"answer": node.answer}
    # Sinon on stocke la question et les branches oui/non récursivement
    return {
        "question": node.question,
        "yes": serialize(node.yes),
        "no": serialize(node.no)
    }
```

La fonction Serialize

- transforme un arbre de Node en dictionnaire Python prêt à être converti en JSON.
- appelle récursivement serialize() sur les enfants.

```python
# Désérialisation : recrée l'arbre à partir d'un dictionnaire JSON
def deserialize(data):
    if "answer" in data:
        # Si c'est une feuille
        return Node(answer=data["answer"])
    # Sinon, c'est un nœud avec une question
    node = Node(question=data["question"])
    node.yes = deserialize(data["yes"])
    node.no = deserialize(data["no"])
    return node
```

La fonction deserialize

- transforme un dictionnaire JSON en objet Node.
- Fonction récursive : elle reconstruit tout l'arbre.

```python
# Sauvegarde de l'arbre dans un fichier JSON
def save_tree(node, filename):
    with open(filename, 'w') as f:
        json.dump(serialize(node), f)

# Chargement de l'arbre depuis un fichier JSON
# Si le fichier n'existe pas, on crée un arbre simple avec la réponse "un chat"
def load_tree(filename):
    if os.path.exists(filename):
        with open(filename, 'r') as f:
            return deserialize(json.load(f))
    else:
        return Node(answer="un chat")
```

save_tree() : enregistre l'arbre dans un fichier .json.
load_tree() : charge l'arbre existant ou en crée un tout simple (réponse par défaut : "un chat").

```python
# Classe principale gérant l'interface graphique et la logique du jeu
class AkinatorGUI:
    def __init__(self, master):
```

Le constructeur de la classe AkinatorGUI reçoit une fenêtre Tkinter (master) et construit l'interface utilisateur du jeu.

```python
# Nom du fichier où l'arbre est stocké
self.tree_file = "base_jeu.json"
# Chargement ou création de l'arbre
self.root = load_tree(self.tree_file)
self.current_node = self.root  # Le nœud courant dans l'arbre (progression du jeu)
```

On charge l'arbre de questions depuis le fichier (ou on en crée un par défaut).
current_node : position actuelle dans l'arbre pendant le jeu.

```
# Label pour afficher les questions ou réponses
self.label = tk.Label(master, text="Bienvenue dans Mini Akinator !\nClique sur Démarrer quand tu es prêt.",
                      font=("Helvetica", 16), wraplength=400)
self.label.pack(pady=20)

# Bouton Oui, désactivé au départ car le jeu n'a pas encore commencé
self.button_yes = tk.Button(master, text="✅ Oui", font=("Helvetica", 14),
                            command=lambda: self.answer(True), width=10, state=tk.DISABLED)
self.button_yes.pack(pady=5)

# Bouton Non, désactivé au départ
self.button_no = tk.Button(master, text="❌ Non", font=("Helvetica", 14),
                           command=lambda: self.answer(False), width=10, state=tk.DISABLED)
self.button_no.pack(pady=5)

# Bouton Démarrer (au lancement visible), sert aussi pour rejouer
self.start_button = tk.Button(master, text="▶️ Démarrer", font=("Helvetica", 14),
                              command=self.start_game, width=15)
self.start_button.pack(pady=20)
```

Label : texte affiché à l'écran (question ou réponse).
Boutons Oui / Non : désactivés au départ. Ils appellent self.answer(...).
start_button : sert à lancer ou relancer le jeu.

```
# Démarre le jeu (appelé au clic sur Démarrer ou Rejouer)
def start_game(self):
    # On masque le bouton Démarrer/Rejouer pendant la partie
    self.start_button.pack_forget()
    # On active les boutons Oui et Non
    self.button_yes.config(state=tk.NORMAL)
    self.button_no.config(state=tk.NORMAL)
    # On remet le nœud courant à la racine de l'arbre
    self.current_node = self.root
    # Affiche la première question ou réponse
    self.label.config(text=self.get_text(self.current_node))
```

Active les boutons.
Affiche la première question ou réponse.
Cache le bouton "Démarrer".

```
# Renvoie le texte à afficher pour un nœud (question ou réponse)
def get_text(self, node):
    if node.is_leaf():
        return node.answer  # Feuille → réponse
    else:
        return node.question  # Nœud interne → question
```

Retourne le texte à afficher selon le type du nœud (question ou réponse).

```
# Fonction appelée quand on clique Oui ou Non
def answer(self, is_yes):
    if self.current_node.is_leaf():
        # Si on est sur une feuille, c'est la fin de la partie
        if is_yes:
            # L'IA a deviné correctement
            messagebox.showinfo("Bravo !", "J'ai deviné ! 😎")
        else:
            # L'IA s'est trompée, elle va apprendre
            self.learn()
        # Après fin partie : désactiver Oui/Non
        self.button_yes.config(state=tk.DISABLED)
        self.button_no.config(state=tk.DISABLED)
        # Afficher le bouton Rejouer
        self.start_button.config(text="🔄 Rejouer")
        self.start_button.pack(pady=20)
    else:
        # Si c'est une question, on descend dans l'arbre selon la réponse
        self.current_node = self.current_node.yes if is_yes else self.current_node.no
        # Met à jour le texte affiché (question ou réponse)
        self.label.config(text=self.get_text(self.current_node))
```

Si c’est une feuille :

- Si l’IA a deviné : message de réussite.
- Sinon : on appelle learn() pour apprendre une nouvelle réponse.
- Ensuite, on désactive Oui/Non et affiche "Rejouer".

Si c’est une question, on descend dans l’arbre.


```python
# Fonction pour apprendre une nouvelle question/réponse si l'IA s'est trompée
def learn(self):
    # Demande à l'utilisateur ce à quoi il pensait réellement
    correct = simpledialog.askstring("Je me suis trompé 😳", "À quoi pensais-tu ?")
    if not correct:
        return  # Si rien de saisi, on abandonne

    # Demande une question pour distinguer la bonne réponse de l'ancienne
    question = simpledialog.askstring(
        "Apprends-moi !",
        f"Une question pour distinguer {correct} de {self.current_node.answer} ?"
    )
    if not question:
        return  # Abandon si pas de question

    # Demande si la réponse à cette question pour la nouvelle réponse est oui ou non
    is_yes = messagebox.askyesno("Pour toi...",
                                 f"Pour {correct}, la réponse à cette question est-elle OUI ?")

    # Sauvegarde l'ancienne réponse pour ne pas la perdre
    old_answer = self.current_node.answer
    # Transforme le nœud actuel en question
    self.current_node.question = question
    if is_yes:
        # Branche oui = nouvelle réponse, branche non = ancienne réponse
        self.current_node.yes = Node(answer=correct)
        self.current_node.no = Node(answer=old_answer)
    else:
        # Branche non = nouvelle réponse, branche oui = ancienne réponse
        self.current_node.no = Node(answer=correct)
        self.current_node.yes = Node(answer=old_answer)
    # On enlève l'ancienne réponse du nœud car il est devenu une question
    self.current_node.answer = None

    messagebox.showinfo("Merci !", "J'ai appris quelque chose de nouveau 😊")
```

Demande :

- À quoi pensait l'utilisateur.
- Une question pour différencier la nouvelle réponse de l'ancienne.
- Si la réponse à cette question est "oui" pour le nouvel objet.
- Modifie l'arbre en conséquence.

Le nœud courant devient une nouvelle question.


```python
if __name__ == "__main__":
    # Création de la fenêtre principale Tkinter
    root = tk.Tk()
    root.geometry("500x400")  # Taille de la fenêtre
    app = AkinatorGUI(root)   # Création de l'application
    root.mainloop()           # Boucle principale Tkinter
```

Initialise la fenêtre principale.
Crée une instance de AkinatorGUI.
Démarre la boucle Tkinter (qui attend les actions de l'utilisateur).