INSTITUT POLYTECHNIQUE DE PARIS

Projet Informatique de 1ère Année à Tèlècom SudParis

PRO-3600

Sujet Image GeoGuessing

Membres Nathan Féret

Pierre Domachowski

Nour Rammal

Charles Meyer

Théophile Schmutz

Enseignante responsable Elisabeth Brunet

Date de soutenance 30 Mai 2022

Table des matières

1 Introduction **1**

2 Cahier des charges **3**

3 Développement **4**

3.1 Analyse du problème et spécification fonctionnelle 4

3.2 Conception préliminaire 4

3.2.1 Collecte et nettoyage des données 4

3.2.2 Conception et entraînement du CNN 6

3.2.3 Mise à disposition de l'outil sur une interface Web 8

3.2.4 Intéraction avec l'IA 8

3.2.5 Tests unitaires 8

3.3 Conception détaillée 8

3.4 Codage 10

4 Manuel d'utilisateur **11**

5 Conclusion **12**

Appendices **14**

Annexe A : Code source **14**

.1 Programme Master - GetDataBaseColab.py 14

.2 Programme DataCleaning - main.py 21

.3 Programme DataCleaning - DisplayFunctions.py 23

.4 Programme DataCleaning - FiltragePhoto.py 25

.5 Programme DataCleaning - ProduitConvolution.py 28

.6 Programme Master - TrainIA.py 30

.7 Programme Master - headlessBrowser.py 34
.8 Programme Master - DiscordBot39.py 36

Chapitre 1

Introduction

Notre projet a été très librement inspiré par le jeu *Geoguessr*[1]. Le jeu présente aux joueurs des images *Street View* (cf la figure 1.1) dont l'utilisateur, en utilisant toute connaissance préexistante, doit deviner l'emplacement de l'image. L'utilisation de connaissances préexistantes offre aux joueurs ayant beaucoup voyagé un avantage dans le jeu. Les connaissances préalables peuvent inclure l'utilisation du langage sur les panneaux de signalisation, le type de véhicule à moteur etc. En utilisant les avancées récentes de la technologie de reconnaissance d'images, nous avons cherché à reproduire ce jeu mais avec comme seul joueur, une IA. Le jeu note l'utilisateur en fonction de la proximité de la latitude et de la longitude de la supposition par rapport à la latitude et à la longitude réelles de l'emplacement de l'image *Street View*. Bien qu'il s'agisse d'une tâche extrêmement difficile pour une machine qui n'a aucune connaissance préalable d'un voyageur passionné, l'utilisation de CNN[2] pour d'autres tâches de reconnaissance d'images a donné des résultats étonnants. En effet, avec la pléthore d'images *Street View* disponibles, un réseau de neurones profond peut apprendre à remarquer ces subtilités. Comme le montre la figure 1.1 sur l'image de gauche, il y a des montagnes et des bâtiments en pierre rouge qui peuvent être utilisés par un réseau de neurones profonds pour s'assurer que l'image provient d'un rocher.

Figure. 1.1 : exemples d'images de googles street view. Bouldeer à gauche, La Tour Eiffel, Paris à droite

La possibilité de géolocaliser les images peut aider à géolocaliser de nouvelles données invisibles, facilitant ainsi la création d'un nouvel ensemble de données. Une autre application de cette technologie pourrait être d'aider à identifier un emplacement avec une connexion faible

1. https://www.geoguessr.com/
2. *Convolutional Neural Networks* - réseau de neurones convolutifs

ou inexistante. Une personne perdue dans une région à faible connexion pourrait prendre une image de l'environnement pour obtenir une estimation de son emplacement.

Chapitre 2

Cahier des charges

Le cahier des charges a changé au cours du projet :

1. Le moyen d'utiliser l'IA, en plus du site, n'est plus via une application mobile mais via un *bot Discord*,
2. La fonctionnalité de prendre des photos est donc reléguée à *Discord*,
3. Ce n'est plus une localisation précise qui est prédite mais une région de France,
4. Ajout d'une fonctionnalité : "devine la région" via le *bot Discord*, système de score

Toutefois, le coût d'utilisation est bien nul, comme spécifié en début de projet, et nous avons une interface graphique permettant à quiconque d'utiliser l'IA. L'affichage graphique de la région prédite est bien implémenté sur le site.

Chapitre 3

Développement

3.1 Analyse du problème et spécification fonctionnelle

La réalisation d'une IA de géolocalisation repose tout d'abord sur l'extraction des données et le stockage de ces dernières en un *traintest* et un *testset*. Or ces images doivent se limiter au territoire français métropolitain d'où la nécessité de définir un contour du territoire duquel seront extraites les images.

Ensuite vient l'étape de la réalisation de l'IA. Ce type de problème correspond à un apprentissage profond. L'architecture choisie est celle d'un réseau de neurones convolutif (CNN). Or la réalisation d'un CNN seul ne garantit pas des résultats optimaux, surtout étant donné le temps que nous avons pour la réalisation du projet, d'où l'intérêt d'utiliser un modèle pré-entraîné, à savoir une *ResNet*.

Enfin, l'interaction avec l'IA se fera via un bot *discord*[1] ainsi que via un site web ou l'utilisateur devra télécharger l'image et avoir en sortie la région correspondante à son image. Ces deux outils viennent remplacer

3.2 Conception préliminaire

3.2.1 Collecte et nettoyage des données

Conformément au cahier des charges, les étapes suivantes ont été suivies pour collecter des données à travers la France :

La première étape consiste à créer un contour de la France. Les fichiers de forme les plus récents de la France contiennent non seulement les frontières entre la France et les pays voisins, mais également les frontières de l'Andorre, Monaco, entre la France et l'océan et de toutes les îles et territoires faisant partie de la France. Pour concentrer les efforts de collecte de données uniquement sur la France métropolitaine, la frontière de la France a dû être isolée. Pour ce faire, toutes les formes ont été converties en polygones. Le polygone a été constitué à l'aide des

1. Discord est un logiciel propriétaire gratuit de VoIP et de messagerie instantanée

coordonnées d'une dizaine de sommets du polygone recouvrant la France métropolitaine s'est avéré isoler les limites de cette dernière. Les opérations géométriques pour cette tâche ont été effectuées à l'aide de *Shapely*[2].

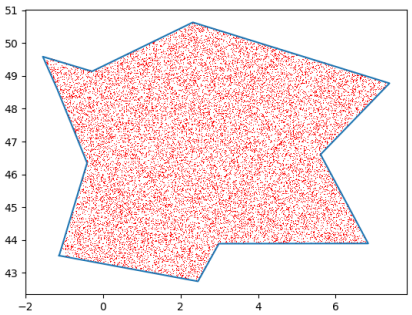

Figure. 3.1 : Polygone représentant la France avec 20000 points répartis uniformément

La deuxième étape consiste à la collecte des image depuis l'API de *Google Street View*.

Pour entraîner le CNN de notre IA, il nous faut une grande quantité de données. C'est-à-dire des dizaines de milliers d'images du paysage français. Pour cela nous avons généré des points aléatoirement sur le polygone susnommé. Nous avons simulé une loi uniforme sur ce polygone pour obtenir les 20000 coordonnées des points dont nous avons ensuite récupéré les *Street View* associées. A chaque image est associé un fichier dont les informations proviennent d'une requête metadata faite à la même API.

Les images et informations sont ensuite stockées sur un *Google Drive*.

Une fois les quelques 20000 images récoltées sur le *Google Drive*, nous remarquons rapidement que certaines images sont celles d'un mur. Pire encore, certaines affichent simplement "*Sorry, we have no imagery here*"

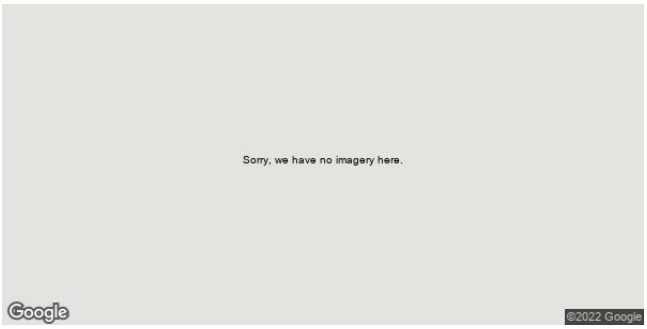

Figure. 3.2 : Erreur : "*Sorry, we have no imagery here*"

Pour se débarrasser de ces images non porteuses d'information utile. Une étape de *data cleaning* devient alors nécessaire. Pour ce faire, nous avons supprimé les images ayant une variance inférieure à un certain seuil déterminé empiriquement.

2. bibliothèque Python pour la manipulation d'objets géométriques et d'analyse du système de coordonnées cartésiennes. : https://shapely/manual.com/

Nous avons, pour affiner le filtrage des images, appliqué un filtre de détection de contours avant de calculer les variances : un image lisse (celle d'un mur), possède moins de contours qu'un paysage détaillé, et son filtrage est alors encore plus uniforme et donc plus facile à détecter. Cependant cette technique n'a, en pratique, pas amélioré la détection de murs.

3.2.2 Conception et entraînement du CNN

La tâche d'apprentissage automatique à accomplir consistait à prédire la région d'une image à partir d'une image aléatoire donnée. Pour ce faire, les images d'entrée sont chargées et converties en tableaux *NumPy*[3]. Le tableau est composé de valeurs RVB et a la forme suivante $(300, 600, 3)$. Pour l'entraînement, les numéros de région correspondant au vecteur d'image d'entrée donné sont convertis en un *hot vector*. Le vecteur ainsi formé a une forme de $(1, 12)$ avec toutes les valeurs nulles sauf l'emplacement représentant la région qui est à un. Le modèle d'apprentissage automatique formé sur les données comporte deux composants principaux, le modèle RestNet préformé gelé et l'architecture CNN pouvant être entrainé,

Figure. 3.3 : Vecteur associé à la région Auvergne-Rhône-Alpes

La première partie du modèle est le modèle pré-entraîné *ResNet*. *ResNet*, abréviation de Residual *Networks*, est un réseau de neurones utilisé comme modèle pour de nombreuses tâches de vision par ordinateur. Ce modèle a remporté le défi *ImageNet* en 2015. La propriété fondamentale du *ResNet* est la possibilité de former des réseaux de neurones extrêmement profonds avec plus de 150 couches. *ResNet* a introduit pour la première fois le concept de connexion de saut. Le *RestNet* empile les couches de convolution les unes après les autres, tout comme les modèles ordinaires. Mais *ResNet* ajoute également l'entrée d'origine à la sortie du bloc de convolution empilé. C'est ce qu'on appelle sauter la connexion. L'une des raisons pour lesquelles les connexions de saut fonctionnent est qu'elles atténuent le problème de la disparition du gradient en créant des chemins raccourcis alternatifs pour que le gradient puisse passer à travers. Elles permettent également au modèle d'apprendre une fonction d'identification qui garantit que la couche supérieure performera au moins aussi bien que la couche inférieure. Le modèle *Keras*[4]

3. https://numpy.org/ - bibliothèque pour langage de programmation Python
4. bibliothèque open source écrite en python, dépend de *TensorFlow*

RestNet pré-entraîné est utilisé pour ce projet. Le modèle est chargé et les poids sont figés et "trainable" est défini sur faux. Ceci est fait car le modèle *ResNet* n'est utilisé que pour convertir l'image en une représentation vectorielle complète significative. Cela permet d'utiliser l'apprentissage par transfert du *ResNet* pour aider le CNN entraînable à l'étape suivante. La structure *ResNet* peut être vue dans la figure 3.4 ci-dessous.

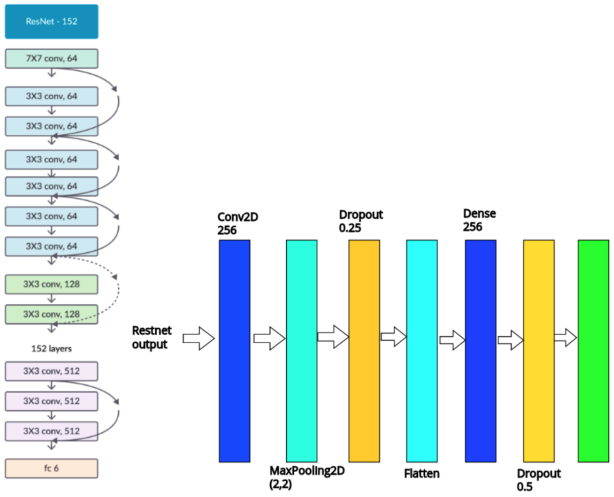

Figure. 3.4 : À gauche le modèle *RestNet* pré-entraîné et à droite le modèle *MNIST* avec des paramètres à entraîner

Pour le modèle entraînable, un réseau de neurones convolutif multicouches a été utilisé. Le modèle utilise un *dropout* entre les couches pour éviter l'*overfitting*[5]. Le modèle consiste en une couche convolutive bidimensionnelle qui prend une image de forme $(300, 600, 3)$ qui couvre la largeur et la hauteur de l'image ainsi que les trois valeurs RVB pour chaque pixel. Ceci est suivi d'une couche de max *pooling* de la taille du pool $(2, 2)$ suivie d'un *dropout* pour éviter l'*overfitting*. La sortie de cette couche est aplatie pour la rendre compatible avec la couche de sortie *softmaxed*. La sortie aplatie est passée à travers une couche dense cachée suivie d'une couche de décrochage. La couche de sortie finale est une couche dense qui produit une sortie *softmaxed* sur 12 régions (grilles). Le modèle est ensuite compilé avant que l'entraînement puisse avoir lieu.

Le programme pour entraîner l'IA est exécuté sur un *notebook* du site *kaggle* (site pensé pour le *machine learning*) pour profiter de leur capacité de calcul supérieure à celle de nos ordinateurs personnels (quota de 30h d'accès GPU, optimal pour le traitement d'images) Après quelques heures d'entraînement, l'IA atteint une précision de 42%. [6]

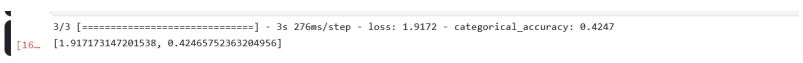

Figure. 3.5 : Précision du réseau après entraînement sur un *test set* disjoint du *train set*

5. le modèle connait par coeur les données d'entraînement plutôt que de les comprendre
6. soit plus de 5 fois mieux qu'un tentative de réponse aléatoire (1 chance sur 12 classes soit 8% de précision)

3.2.3 Mise à disposition de l'outil sur une interface Web

La dernière étape était de développer une interface Web pour donner un accès libre à notre IA.

— pour cela nous avons commencé par élaborer la maquette graphique du site Web afin de définir son allure générale, les polices d'écriture, les couleurs etc.

— puis, nous avons développé le site conformément à la maquette. Ainsi, ce dernier dispose d'une page d'accueil qui permet de rediriger les utilisateurs sur les pages plus fonctionnelles, d'une page qui présente l'équipe et le projet, une autre qui donne nos contacts et enfin, la page la plus importante qui comporte une rubrique pour téléverser une image issue de l'ordinateur (image de 300×600 pixels), un bouton pour lancer l'IA et une partie de cette page est réservée à l'affichage du résultat de l'IA i.e. la région dont l'image semble être issue.

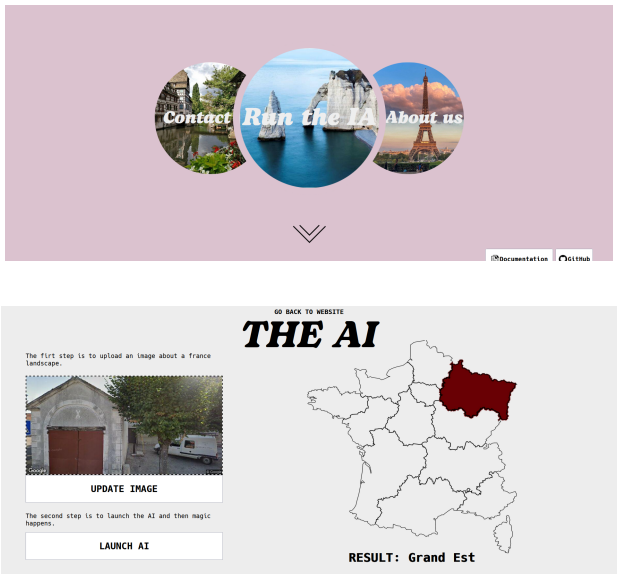

Figure. 3.6 : Visuel du site

3.2.4 Intéraction avec l'IA

Notre projet propose deux manière d'interagir avec l'IA Via le site internet, ou via le bot *Discord* (qui utilise un navigateur sans tête).

3.2.5 Tests unitaires

Les tests unitaires ne se prêtant pas vraiment à notre projet, nous avons décidé, avec l'accord de notre tutrice, de ne pas en faire.

3.3 Conception détaillée

Initialement nous voulions développer une application mobile native afin d'interagir avec l'IA. Cependant relier *TensorFlow* et *React Native* s'est avéré bien plus compliqué que ce que

nous avions anticipé. En partie à cause de problèmes de compatibilité et de versions. Le manque cruel de documentation à ce sujet est également à déplorer. Les rares tutoriels et explications officielles sur les sites de ces outils menaient à moult erreurs. Nous nous sommes donc résolus à devoir trouver une autre solution pour communiquer avec l'IA

Nous est alors venue l'idée d'interagir via un bot *Discord*, et de l'héberger sur une machine virtuelle de *MiNET*[7], et de copier les fichiers de l'IA.

Cependant nous n'avons pas pu implémenter *TensorFlow* directement sur la machine virtuelle de minet. Un message d'erreur nous indique qu'il manque effet le set d'instruction SSE4.2 nécessaire à l'implémention de *TensorFlow*.

Pour que le bot puisse tout de même communiquer avec l'IA, elle devait alors faire une requête au site internet sur lequel le modèle était implémenté grâce à une conversion depuis le modèle entraîné en python vers *TensorFlowjs*. Nous avons donc implémenté un *browser* sur la machine virtuelle sur laquelle le bot tourne. Le module *selenium* en python a permis de lancer un *headless-browser*, qui répond totalement à nos besoins.

Un *headless-browser* (navigateur sans tête) étant tout simplement un navigateur sans interface graphique, pratique puisque la machine virtuelle n'a pas de GUI.

Le bot est fonctionnel à tout moment. Il est lancé via le gestionnaire de processus pm2[8] qui est très utile.

```
nferet@discord-bot:~/DiscordBot$ pm2 list
```

id	name	namespace	version	mode	pid	uptime	↺	status	cpu	mem	user	watching
2	DiscordBot39	default	N/A	fork	83459	2m	0	online	0%	50.1mb	nferet	disabled

Figure. 3.7 : Capture de pm2 fonctionnant

Nous avons aussi implémenté un système de score : à chaque partie gagné, le score augmente de 1, il est possible d'afficher le tableau des scores, ainsi que d'obtenir le score d'un joueur.

Une limitation subsiste. Un problème de concurrence. En l'état actuel du bot, un seul utilisateur peut communiquer avec le bot, et donc avec l'IA. Une solution pour supprimer cette limitation est peut être le *threading* dans le code du bot, malheureusement, par manque de temps nous n'avons pas implémenté cette solution.

Pour implémenter l'IA sur notre site, nous avons tout simplement converti le modèle entraîné sur python en un modèle *TensorFlowjs* utilisable dans un script *javascript*. La bibliothèque *TensorFlowjs* en python propose une fonction permettant cette conversion.

7. Association de réseau de TSP - https://www.minet.net
8. *Process Manager 2* - gestionnaire de processus Open Source très répandu

3.4 Codage

La documentation [9] est détaillée sur notre *GitHub* et site :

cliquer ici pour accéder au *GitHub*
cliquer ici pour accéder au site

9. La documentation a été généré sur le site https://www.sphinx.com

Chapitre 4

Manuel d'utilisateur

Le manuel d'utilisateur est divisé en deux temps :

— **Utilisation du Site :** Pour pouvoir utiliser Image Geoguessing, il suffit de se rendre sur notre site et suivre ces instructions. Une fois sur la page d'accueil il faut :

1. cliquer sur le cercle "Run the AI",
2. cliquer sur "Update Image",
3. ouvrir l'image à analyser dans les fichiers de l'ordinateur, Si elle fait plus de 300*600 pixels elle sera rognée
4. cliquer sur "Launch AI" pour faire fonctionner l'IA,
5. constater la région dont est issue l'image a priori, qui s'affiche sous la carte de France.

— **Utilisation du bot *Discord GeoGuessing* :** (Pré-Requis : être présent sur un serveur discord où le bot est présent. Le préfixe pour envoyer une commande est " ? ?")

1. "play" pour lancer une partie de *GeoGuessing* : Après quelques instants de chargement, une image s'affiche et le bot vous invite à deviner la région où la photo a été prise,
2. "guess NomRegion" pour essayer de deviner, le bot vous ajoute un point si vous avez bon, et vous demande de recommencer sinon,
3. "stop" pour abandonner la partie,

Vous pouvez également consulter votre score avec la commande "score", et le tableau des scores avec la commande "showleader n" (n étant le nombre de lignes à afficher).

Chapitre 5

Conclusion

Le modèle final correspond donc à une IA qui prédit la région correspondante à l'image en entrée avec une précision de près de 50%, soit 5 fois plus qu'une conjecture humaine. L'interaction avec cette dernière peut se faire via le site que nous avons conçu ou encore, le bot *Discord*.

Au vu des résultats obtenus par le modèle, nous constatons que plusieurs voies d'amélioration sont possibles. Premièrement, les données *Google Street View* collectées sont parfois très semblables les unes aux autres. Si le modèle recevait des images des repères de définition caractéristiques de chaque emplacement, il pourrait mieux fonctionner. L'augmentation du nombre d'époch pourrait également aider à améliorer le modèle. Enfin, en tenant compte de ce que nous avons mentionné précédemment, nous pouvons envisager une amélioration de la précision géographique du modèle en essayant de prédire la ville correspondant à l'image plutôt que la région.

Bibliographie

https://support.mozilla.org/

https://developer.mozilla.org/fr/

https://discordpy.readthedocs.io/en/stable/

https://discord.com/developers/docs

https://stackoverflow.com/

https://api.jquery.com/

https://www.sphinx-doc.org/en/master/

https://keras.io/api/applications/

https://developers.google.com/maps/documentation/streetview/overview

Annexe A : Code source

.1 Programme Master - GetDataBaseColab.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
""" Signs a URL using a URL signing secret """

import hashlib
import hmac
import base64
import urllib.parse as urlparse


def sign_url(input_url=None, secret=None):
    """ Sign a request URL with a URL signing secret.
      Usage:
      from urlsigner import sign_url
      signed_url = sign_url(input_url=my_url, secret=SECRET)
      Args:
      input_url - The URL to sign
      secret    - Your URL signing secret
      Returns:
      The signed request URL
  """

    if not input_url or not secret:
        raise Exception("Both input_url and secret are required")

    url = urlparse.urlparse(input_url)

    # We only need to sign the path+query part of the string
    url_to_sign = url.path + "?" + url.query

```

```
    # Decode the private key into its binary format
    # We need to decode the URL-encoded private key
    decoded_key = base64.urlsafe_b64decode(secret)

    # Create a signature using the private key and the URL-encoded
    # string using HMAC SHA1. This signature will be binary.
    signature = hmac.new(decoded_key, str.encode(url_to_sign), hashlib.sha1)

    # Encode the binary signature into base64 for use within a URL
    encoded_signature = base64.urlsafe_b64encode(signature.digest())

    original_url = url.scheme + "://" + url.netloc + url.path + "?" +
    ↪  url.query

    # Return signed URL
    return original_url + "&signature=" + encoded_signature.decode()
  # Import PyDrive and associated libraries.


# This only needs to be done once in a notebook.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once in a notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

!mkdir /content/PRO3600_DATA
!mkdir /content/PRO3600_DATA/Images
!mkdir /content/PRO3600_DATA/Infos

from shapely.geometry import Point,Polygon

import urllib.parse as parse

import requests
import random
```

```
72
73 from PIL import Image
74 import io
75 import base64
76 import numpy as np
77 import time
78
79 API_KEY="AIzaSyDorg3QnkxRSctW_8vhQZ_hi3FQjX__kuU"
80 secret='ku9pfVbcb67-bRabEr2FMgmCBQk='
81
82 path="/content/PRO3600_DATA"
83 params='size=600x300&source=outdoor&radius=500'
84
85 Nb_img=20000 # on veut X images
86
87 def getcos():
88     """
89     renvoie -au hasard uniforme- un couple lat lon de reels bornés
90     """
91     point=Point((0,0))
92
93     latmin=40
94     latmax=51
95
96     longmin=-2
97     longmax=8
98     while not point.within(France): #tant que le point généré n'est pas valide
99       lat=random.random()
100      long=random.random()
101
102      lat = lat*(latmax-latmin)+ latmin
103      long = long*(longmax-longmin)+ longmin # rd[0,1] -> rd[min,max]
104
105      lat=round(lat,6)
106      long=round(long,6) # on met le bon nombre de decimales
107      point=Point(lat,long)
108
109     return lat, long
110
111
112 def GetMetaRequete(address,params=params):
113     """renvoie la requete __METADATA__
```

```
114      utilisé pour savoir si la requet est valide ou non
115
116      param = arg1=value1&arg2=value2    """
117
118
     ↪  url="https://maps.googleapis.com/maps/api/streetview/metadata?{}&location={}".forma
119
120     url=url+"&key="+API_KEY # ajout de ka clé api a la requete
121
122     signed_url=sign_url(url,secret) # on signe la requete
123
124     req=requests.get(signed_url) # on envoie a l'api de metadonnées
125     return req
126
127
128
129 def GetImage(req):
130     """
131     req étant composé de bits, on traduit en un array 'usuel"
132     renvoie l'array
133     """
134     bits=req.content # on prend le contenu de la requete
135     b=bits.strip() # on enleve les \n
136     c=io.BytesIO(b) # on décode les bits
137     img=Image.open(c) # on ouvre avec Image
138     return np.array(img) # pour ensuite récupérer sous forme de matrice
139
140
141 def GetRequeteReverseGeo(lat,lng):
142     """
143     envoie une requete a l'api de reverse geocoding pour obtenir les infos sur
 ↪  l'endroit auquel correspondent les coordonées
144     """
145
146
     ↪  url="https://maps.googleapis.com/maps/api/geocode/json?latlng={},{}".format(lat,lng
147     url=url+"&key="+API_KEY
148
149     req=requests.get(url) # pas besoin de signer l'url
150     return req
151
152
```

```
153 def getaddress(geodata):
154   string = geodata.content.decode() # on recup le string a partir de la
      ↪ requete
155   lines = string.split("\n") # on découpe le bloc  en lignes
156   ok=True
157   for line in lines:
158     if "formatted_address" in line and ok: # on prend la 1ere occurence de la
        ↪ ligne formatted
159       ok=False
160       GoodLine=line
161
162   # possible UnboundLocalError si pas de formatted dans le fichier json
163   # on peut le traiter pour faire propre, sinon on laisse, ça arrive 1
      ↪ fois/4000
164
165   GoodLine=GoodLine.split(":") # on découpe juste le bout qui nous interesse
166   GoodLine=GoodLine[1][2:-2]
167   GoodLine=GoodLine.split(" ")
168   GoodLine=GoodLine[-2]+GoodLine[-1]
169
170
171   return GoodLine # on renvoie l'adresse
172
173
174 def saveInfo(geodata,nameinfo):
175   """enregistre les données geodata sur colab au format txt"""
176   pathinfo=path+"/Infos/"+nameinfo
177   string = geodata.content.decode()
178   file = open(pathinfo,"w")
179   file.write(string)
180   file.close()
181
182
183 def saveimg(mat,nom):
184     """enregistre la matrice mat sous forme d'image sous le nom "nom" au
        ↪ chemin "pathimg"
185     """
186     pathimg=path+"/Images/"
187     img=Image.fromarray(mat)
188     img.save(pathimg+nom)
189
190
```

```
191 def saveToDrive(pathimg,id):
192   """
193   enregistre l'image au chemin pathimg sur le drive, avec le nom pathimg
194   """
195
196   gfile = drive.CreateFile({'parents': [{'id': id}]}) #crée le fichier vide
197   upload_file=pathimg
198   gfile.SetContentFile(upload_file) # ajoute le contenu de pathimg au fichier
199   gfile.Upload() #upload au drive
200
201
202 def GetRequeteFromAddress(address,params=params):
203   """ renvoie la resultat de la requete API a l'adresse donnée """
204   address=parse.quote(address) # ENCODE L URL C'EST IMPORTANT
205   url=
      ↪  "https://maps.googleapis.com/maps/api/streetview?{}&location={}".format(params,addres
206   url=url+"&key="+API_KEY
207   signed_url=sign_url(url,secret)
208   req=requests.get(signed_url)
209   return req
210
211
212 def main():
213
214   compteur=0
215   while compteur<Nb_img: # pour le nombre d'images voulues
216
217     time.sleep(0.1) # on rajoute 0.1s de délai par précaution(?)
218     lat,lng=getcos() # on genere un point sur la france
219     print(lat,lng)
220
221     geodata=GetRequeteReverseGeo(lat,lng) # on recup les geodonnées de l'api
222     address=getaddress(geodata) # on extrait l'adresse
223
224     print("Image : ",compteur , "Addresse : ", address)
225     req=GetRequeteFromAddress(address) # on prend la street view
226     # (potentiels bugs ? nan tkt)
227
228     img=GetImage(req) # on récup l'image
229
230     # on enregistre sur le drive l'image et le fichier d'infos associé
231
```

```
232     nameimg="img{}.png".format(compteur)
233     nameinfo="Info{}.txt".format(compteur)
234 
235     saveimg(img,nameimg)
236     saveInfo(geodata,nameinfo)
237 
238     ImageFolderID='1hPhwEse7OiL1-Q511pqwnJZYgDJsKpdE'
239     saveToDrive(path+"/Images/"+nameimg,ImageFolderID)
240 
241     InfoFolderID = '1T_UlkbKyxbu6skvkb1AjhL_jEuLnCW7T'
242     saveToDrive(path+"/Infos/"+nameinfo,InfoFolderID)
243     print("Done")
244     compteur+=1
245     print(compteur)
246 
247 
248 
249 
250 
251 def createPoly():
252   """crée le polygone représentant la france"""
253   Lpoints=[[43.525093, -1.134750],[42.741205, 2.447694],[43.888962,
      ↪  2.987746],[43.894815, 6.840072],[46.605900, 5.612790],[48.774532,
      ↪  7.390237],[50.630174, 2.315648],[49.137655, -0.284608],[49.585080,
      ↪  -1.555339],[48.608597, -1.188893],[48.608597, -1.188893],[46.346987,
      ↪  -0.413643]]
254   PointList=[]
255   for pt in Lpoints:
256       PointList.append((pt[0],pt[1])) #on remplace les listes par des tuples
257 
258   poly=Polygon(PointList)
259   return poly # on renvoie le polygone généré
260 
261 France=createPoly()
262 main()
```

.2 Programme DataCleaning - main.py

```
import ProduitConvolution as pc
import FiltragePhoto as fp
import DisplayFunctions as df

import matplotlib.pyplot as plt
import numpy as np
from skimage.io import imread, imshow

import shutil



plt.close()

# -------- Constantes --------
chemin = "../ImagesBDD/"
cheminBDD = 'C:\\Users\\utilisateur\\Documents\\Télécom SudParis\\Cours S6\DEV Info\\ImagesBDD'
fig = 0
L=[]
Lvar=[]

# -------- Variables --------

seuil = 0.7 # Seuil à parir duquel on considère que l'image est un mur
seuilVar = 0.03
eps = 20 # Marge d'acceptabilité des couleurs
iterationsFin = 20000
# Afficher ou non l'image et histogramme de chaque mur
cond = False

# -------- Tests | Cellule 1 --------
for i in range(iterationsFin):
    string = chemin + 'img' + str(i) + '.png'

    try :
        image = imread(string, as_gray=True)
        Lvar.append(np.var(image))

        if fp.detectWall(image,seuil,eps):
```

```
40
41              if cond : # Afficher les images et histgrammes de chaque mur ?
42                  image = imread(string, as_gray=True)
43                  df.imageHist(image)
44
45              L.append(True)
46              cheminBinImage = str(cheminBDD+ '\\Murs\\img'+ str(i) + '.png')
47              cheminBDDImage = str(cheminBDD+ '\\img'+ str(i) + '.png')
48              shutil.move(cheminBDDImage, cheminBinImage)
49          else :
50              L.append(False)
51
52      except : # Si la photo n'existe pas
53          L.append(False)
54
55  Ltrie1 = L.copy()
56
57
58  L = [False]*iterationsFin
59
60  for i in range(len(L)):
61
62      if not L[i]:
63
64          string = chemin + 'img' + str(i) + '.png'
65          try : # Si la photo existe
66              image = imread(string, as_gray=True)
67              imageConvu = pc.convuProduct(image)
68              if fp.detectVar(imageConvu,seuilVar):
69                  if cond : # Afficher les images et histgrammes de chaque mur ?
70                      image = imread(string, as_gray=True)
71                      df.imageHist(image)
72
73                  cheminBinImage = str(cheminBDD+ '\\Murs\\img'+ str(i) +
                    ↪  '.png')
74                  cheminBDDImage = str(cheminBDD+ '\\img'+ str(i) + '.png')
75                  shutil.move(cheminBDDImage, cheminBinImage)
76                  L[i] = True
77          except : # Si la photo n'existe pas
78              pass
```

.3 Programme DataCleaning - DisplayFunctions.py

```
import ProduitConvolution as pc
import FiltragePhoto as fp
import DisplayFunctions as df

import matplotlib.pyplot as plt
import numpy as np
from skimage.io import imread, imshow
from skimage import exposure



def imageHist(image):
    """
    - image : np.array

    Génère et affiche l'histogramme d'une image, qu'elle soit en nuance de gris ou en couleurs

    Est appelé par : showGrey
    """

    _, axis = plt.subplots(ncols=2, figsize=(12, 3))
    if (image.ndim == 2):
        # Grascale image

        axis[0].imshow(image, cmap=plt.get_cmap('gray'))
        axis[1].set_title('Histogram')
        axis[0].set_title('Grayscale Image')
        hist = exposure.histogram(image)
        axis[1].plot(hist[0])

    else:
        # Color image

        axis[0].imshow(image, cmap='gray')
        axis[1].set_title('Histogram')
        axis[0].set_title('Colored Image')
        rgbcolors = ['red', 'green', 'blue']
        for i, mycolor in enumerate(rgbcolors):
            axis[1].plot(exposure.histogram(image[...,i])[0], color=mycolor)
```

```
40      plt.show()
41
42
43
44
45  def showGrey(path,boolean):
46      """
47      - path : String, contient le chemin d'accès vers une image
48      - boolean : Bool, permet d'afficher ou non l'histogramme de l'image
    ↪   filtrée
49
50
51
52  """
53      image = imread(path, as_gray=True)
54      imageHist(image)
55      if boolean:
56          imageHist(pc.convuProduct(image))
57      plt.show()
```

.4 Programme DataCleaning - FiltragePhoto.py

```
import ProduitConvolution as pc

from skimage.io import imread
from skimage.io import imshow
from skimage import exposure
import matplotlib.pyplot as plt
import numpy as np
import PIL
from skimage import io



def detectWall(image,seuil,eps):
    """
    - image : np.ndarray
    - seuil : float
    - eps : float

    renvoie un booleen valant True si l'image est un mur, selon un critère basé sur la prédominance d'une couleur sur les autres
    """
    n,p = np.shape(image)

    # Compte le nombre de pixels qu'il y a pour chaque nuance de gris
    count = [0]*256
    for i in range(n):
        for j in range(p):
            count[int(image[i,j]*255)] += 1

    # Détermine s'il y a une prédomiance d'une couleur sur le mur
    iMax = count.index(max(count)) # Nuance de gris du pixel le plus présent sur l'image en nuance de gris
    compteur = 0
    borneSup = min(iMax+eps,256)
    if iMax-eps < 0 :
        borneInf = 0
    else :
        borneInf = iMax - eps

    # Compte le nombre de pixel ayant une couleur entre [borneInf,borneSup]
```

```
39      for i in range(borneInf,borneSup):
40          compteur += count[i]
41      compteur *= 1/(n*p) # Calul de la moyenne
42
43      # Critère de prédominance
44      if compteur > seuil: # Les pixels sont quasi tous à peu près de la même
        ↪  couleur
45          return True
46      else :
47          return False
48
49
50
51  def detectVar(image,seuil):
52      """
53      - image : np.ndarray
54      - seuil : float
55
56      renvoie un booleen valant True si l'image est un mur, selon un critère
        ↪  basé sur la variance de l'image
57
58      """
59      # Critère de variance
60      if np.var(image) < seuil:
61          return True # On considère que l'image est un mur
62      else :
63          return False
64
65
66
67
68
69  def booleanHist(L):
70      """
71      - L : Liste,
72
73      L est de la forme [[float0,boolean0],[float1,boolean1], ...]
74
75
76
77      Permet de préparer des données pour faire un histogramme statistiques
78      Utilisable pour des étudier les statistiques
```

```
79
80      renvoie le couple de triplet
    ↪   [varMaxTrue,varMoyTrue/nbr,varMinTrue],[varMaxFalse,varMoyFalse/nbr,varMinFalse
81      (notation : varMaxTrue = variance maximale tel qu'un mur est détecté)
82      """
83      varMaxFalse = 0
84      varMinFalse = 10
85      varMoyFalse = 0
86
87      varMaxTrue = 0
88      varMinTrue = 10
89      varMoyTrue = 0
90
91      nbr = len(L)
92
93      for i in range(nbr):
94          if L[i][1]:
95              varMaxTrue = max(varMaxTrue,L[i][0])
96              varMinTrue = min(varMinTrue,L[i][0])
97              varMoyTrue += L[i][0]
98          else :
99              varMaxFalse = max(varMaxFalse,L[i][0])
100             varMinFalse = min(varMinFalse,L[i][0])
101             varMoyFalse += L[i][0]
102
103     return
        ↪   [varMaxTrue,varMoyTrue/nbr,varMinTrue],[varMaxFalse,varMoyFalse/nbr,varMinFalse]
104
105
106
107
108 """
109 def concatenation(L1,L2):
110     # Sert juste à tester le code
111
112     if len(L1) == len(L2):
113         res=[]
114         for i in range(len(L1)):
115             res.append([L1[i],L2[i]])
116     return res
117 """
```

.5 Programme DataCleaning - ProduitConvolution.py

```
# Produit de Convultion

from skimage.io import imread, imshow
from skimage import exposure
import matplotlib.pyplot as plt
import numpy as np
import PIL
from skimage import io



def convuLocal(portionImage,filtre):
    """
    - portionImage : np.ndarray
    - filtre : np.ndarray

    - Génère ValueError si les matrices ne sont pas de même taille

    Renvoie le produit le convolution entre les deux matrices
    """


    if np.shape(portionImage)!=np.shape(filtre):
        raise ValueError("Matrices de dimensions différentes")

    else:
        n,p = np.shape(portionImage)
        somme = 0
        for i in range(n):
            for j in range(p):
                somme += filtre[i][j]*portionImage[i][j]

        return somme



def convuProduct(image):
    """
    - image : np.ndarray de dimension n,p

```

```
    renvoie le produit de convolution de image par le filtre f de dim(3x3)

    f = -1, 0, 1,
        -1, 0, 1,
        -1, 0, 1,

    """
    n,p = np.shape(image)

    dimFiltre = 3                              # Filtre est de la forme :
    filtre = np.array([[-1,0,1]]*dimFiltre) #    -1 0 1
    matrice = np.zeros((n,p))                  #    -1 0 1
    halfDimFiltre = int(dimFiltre/2)           #    -1 0 1

    # On applique le produit de convolution à chaque matrice extraite 3x3
    for i in range(1,n-1): # On parcourt toute la matrice sauf les bords
        for j in range(1,p-1):
            portionImage=np.zeros([dimFiltre,dimFiltre])

            # Création de la matrice extraite
            for l in range(dimFiltre):
                L = image[i+halfDimFiltre-l][j-halfDimFiltre :j+halfDimFiltre
                ↪  +1]
                portionImage[l]=L

            # On calcule le produit de convolution entre la matrice extraite
            ↪  et le filtre
            resij = convuLocal(portionImage,filtre)
            matrice[i][j]+=resij

    # On enlève tous les bords pour avoir une matrice de dimension (n-1)x(p-1)
    L=[]
    for i in range(1,n-1):
        L.append(matrice[i][1:-1])
    return np.array(L)
```

.6 Programme Master - TrainIA.py

```
import gc
import numpy as np
from PIL import Image
import tensorflow as tf

class Geoguessr:
    def __init__(self,
                 inputShape=(300,600,3), gridCount=12,
                 hidden1=256, hidden2=512):



        # load restnet model
        restnet = tf.keras.applications.resnet50.ResNet50(include_top=False,
                                                           ↪  weights='imagenet',
                                                           ↪  input_shape=inputShape)
        self.model = tf.keras.models.Sequential()
        self.model.add(restnet)

        # freeze resnet model

        self.model.layers[0].trainable = False

        self.model.add(tf.keras.layers.Conv2D(hidden1, (3, 3),
        ↪  activation='relu',
                                              input_shape=inputShape))
        self.model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
        self.model.add(tf.keras.layers.Dropout(0.25))
        self.model.add(tf.keras.layers.Flatten())
        self.model.add(tf.keras.layers.Dense(hidden2, activation='relu'))
        self.model.add(tf.keras.layers.Dropout(0.5))
        self.model.add(tf.keras.layers.Dense(gridCount, activation="softmax"))

        self.model.compile(loss=tf.keras.losses.categorical_crossentropy,
                           optimizer=tf.keras.optimizers.Adam(),
                           metrics=['categorical_accuracy'])

Lreg=[
"Auvergne-Rhône-Alpes",
```

```
"Bourgogne-Franche-Comté",
"Bretagne",
"Centre-Val de Loire",
"Grand Est",
"Hauts-de-France",
"Normandy",
"Nouvelle-Aquitaine",
"Occitanie",
"Pays de la Loire",
"Provence-Alpes-Côte d'Azur",
"Île-de-France"]

def getReg(file):
    """obtient la région d'un file InfoX.txt"""
    K= 2
    lines=file.readlines()
    file.close()
    Goodlines=[]
    for line in lines:
            if "formatted" in line:
                Goodlines.append(line)

    Good=Goodlines[-K].split(":")[1].split(",")[0][2:]
    return Good

def transfo(i,k):
    L=[0]*k
    L[i]=1
    return L

geoModel = Geoguessr().model
print(geoModel.summary)

step=200
nbmax=19999

pathimg="../input/imagesfrance/"
pathinfo="../input/infosfrance/Infos/"

Lx=[]
Ly=[]
  for batch in range(0,nbmax,step):
```

```
80
81      for i in range(batch,batch+step):
82          try:
83              img0=Image.open(pathimg+"img{}.png".format(i))
84              img=np.reshape(img0,[300,600,3])
85              img0.close()
86              info=open(pathinfo+"Info{}.txt".format(i))
87              reg=getReg(info)
88              info.close()
89              indice=Lreg.index(reg)
90
91              Lx.append(np.array(img))
92              Ly.append(transfo(indice,12))
93
94          except: # il manque des images aores nettoyage mais c'est
            ↪ parfaitement normal
95              pass
96
97      if len(Ly)!=0:
98            Lx=np.array(Lx,dtype="float64")
99            Ly=np.array(Ly)
100           geoModel.fit(Lx,Ly,batch_size=4)
101           del Lx
102           del Ly
103           gc.collect()
104           Lx=[]
105           Ly=[]
106
107
108
109 !pip install tensorflowjs
110 import tensorflowjs as tfjs
111 tfjs.converters.save_keras_model(geoModel,"./kaggle/working/ModelJS")
112
113
114 def getCorr():
115     dico=dict()
116     for i in range(19999):
117         try:
118             info=open(pathinfo+"Info{}.txt".format(i))
119             reg=getReg(info)
120             info.close()
```

```
121             dico[i]=reg
122         except (FileNotFoundError, ValueError) as e:
123             pass
124     return dico
125 
126 dico=getCorr()
127 
128 
129 #evaluation des perfs
130 
131 Lx=[]
132 Ly=[]
133 for i in range(19999-2000,19999):
134     try:
135                 img0=Image.open(pathimg+"img{}.png".format(i))
136                 img=np.reshape(img0,[300,600,3])
137                 img0.close()
138                 info=open(pathinfo+"Info{}.txt".format(i))
139                 reg=getReg(info)
140                 info.close()
141                 indice=Lreg.index(reg)
142 
143                 Lx.append(np.array(img))
144                 Ly.append(transfo(indice,12))
145 
146             except:
147                 pass # pas de soucis s'il manque une image
148 
149 geoModel.evaluate(Lx,Ly)
```

.7 Programme Master - headlessBrowser.py

```
from time import sleep
from selenium import webdriver

options = webdriver.FirefoxOptions()
options.add_argument("--headless")
# options.add_argument("--disable-dev-shm-usage")
# options.add_argument("--disable-gpu")


path = "/home/nferet/tests/img.png"


def GetRepIA(path):
    """
    - path : String

    Crée un browser en mode headless ( sans interface graphique)
    charge la page et upload l'image située au chemin path
    renvoie la liste contenant les probas de chaque région

    """
    def toFloat(String):
        if "e" in String:
            return 0
        return float(String)
    url = "https://geoimage.000webhostapp.com/AI.html"
    driver = webdriver.Firefox(options=options)
    print("driver created")
    driver.get(url)
    print("url fetched")
    y = driver.execute_script("return res.toString();")
    print("y=", y)
    sleep(1)
    input_element = driver.find_element(by="id", value='IAImage')
    input_element.send_keys(path)
    sleep(1)
    button_element = driver.find_element(by="id", value='startIaButton')
    button_element.click()
    x = driver.execute_script("return res.toString();")
    while x == "[object Promise]":
```

```
41          x = driver.execute_script("return res.toString();")
42          sleep(1)
43      print(x)
44      x = str(x)
45      Lreps = x.split("[")[2][:-3]
46      Lreps = Lreps.split(",")
47      L = [toFloat(x) for x in Lreps]
48      print(L)
49      return L
50
51  #TESTS
52  assert len(L)==12
53  assert x != None
```

.8 Programme Master - DiscordBot39.py

```
from headless_browser_vm import GetRepIA
from config import TOKEN, ListNum
import discord
import random
import math
import numpy as np

Lregions=[
"Auvergne-Rhône-Alpes",
"Bourgogne-Franche-Comté",
"Bretagne",
"Centre-Val de Loire",
"Grand Est",
"Hauts-de-France",
"Normandy",
"Nouvelle-Aquitaine",
"Occitanie",
"Pays de la Loire",
"Provence-Alpes-Côte d'Azur",
"Île-de-France"]

def get_score(path, user):
    """
    - path : String
    - user : discord.user.User

    renvoie le score de "user" dans la base de données placée en "path"
    """
    file = open(path, 'r')
    lines = file.readlines()
    file.close()

    for i in range(len(lines)):
        line = lines[i]
        line = line.strip()
        line = line.split(" ")
        name = line[0]
        if name == str(user):
            return int(line[1])

```

```
41      # is user not in database we add said user
42
43      file = open(path, "a")
44      file.write(str(user) + " 0\n")
45      file.close()
46      return 0
47
48
49  def set_score(path, user, score):
50      """
51
52      - path : String
53      - user : discord.user.User
54      - score : int
55
56      fixe le score de 'user' à la valeur 'score' dans la bdd placée en 'path'
57
58      """
59
60      file = open(path, 'r')
61      lines = file.readlines()
62      file.close()
63
64      for i in range(len(lines)):
65          line = lines[i]
66          line = line.strip()
67          line = line.split(" ")
68          name = line[0]
69
70          if name == str(user):
71              indiceuser = i
72
73      file = open(path, "w")
74      for k in range(indiceuser):
75          line = lines[k]
76          file.write(line)
77
78      file.write(str(user) + " {} \n".format(score))
79
80      for k in range(indiceuser+1, len(lines)):
81          line = lines[k]
82          file.write(line)
```

```
83
84      file.close()
85
86
87  def update(path, user, add):
88      """
89      - path : String
90      - user : discord.user.User
91      - add : int
92
93      ajoute "add" au score de "user" dans la BDD placée en "path"
94      appelle get_score et set_score
95      """
96      score = get_score(path, user)
97      score += add
98      set_score(path, user, score)
99
100
101 def getloc(num):
102     """
103     num : int
104     num doit être compris entre 0 et 99
105
106     renvoie l'indice de la région associé au numéro num"""
107
108     file = open("./reg99.txt")
109     lines = file.readlines()
110     line = lines[num].strip()
111     return Lregions.index(line)
112
113
114 def get_REG_IA(path):
115     """
116     - path : String
117     path contient le chemin d'accès a l'image qu'on soumet a l'IA
118
119     renvoie la région correspondant au choix de l'IA
120
121     """
122     # renvoie la région prédite par L'IA
123     L = GetRepIA(path)
124     i = np.argmax(np.array(L))
```

```
125     return Lregions[i]
126
127
128 async def geoguess(msg, stage):
129
130     """
131     msg : discord.message.Message
132     stage : int (0 ou 1)
133
134
135     lance une partie de geoGuess
136     """
137
138     content = msg.content[8:]
139
140     if stage == 0:
141         # si le stage vaut 0 on montre l'image et invite le joueur a répondre
142         global numImg
143         numImg = random.choice(ListNum)
144         print(numImg)
145         path_img = "/home/nferet/DiscordBot/Images/img{}.png".format(numImg)
146
147         await Confirm(msg, "Lancement de la partie, veuillez patienter
            ↪ Quelques instants")
148         global region_ia
149         region_ia = get_REG_IA(path_img)
150         await msg.channel.send("Dans quelle région de France a été prise cette
            ↪ photo ?", file=discord.File(path_img))
151         await Confirm(msg, "??guess NomRegion pour répondre")
152         await Confirm(msg, "??stop pour arreter")
153         return 1
154
155     if stage == 1:
156         # a l'étape 1 on attend encore la réponse du joueur
157
158         print(content)
159
160         try:
161             indice = Lregions.index(content)
162
163         except ValueError:
164             # si la région proposée n'est pas valide
```

```
165             await Confirm(msg, "Votre région n'est pas valide, elle doit être
                ↪  un des suivants : {}".format(Lregions))
166             return 0
167
168         print("loc=", getloc(numImg))
169
170         if indice == int(getloc(numImg)):
171             # si la région propose est la bonne on maj le score et on annonce
                ↪  au joueur qu'il a gagné, on lui affiche aussi la prédiction de
                ↪  l'IA (correcte ou non)
172             update("/home/nferet/DiscordBot/scores.txt", msg.author.id, 1)
173
174             await Confirm(msg, "Bien joué !")
175             await Confirm(msg, "L'ia avait prédit : {}".format(region_ia))
176             return -1  # on renvoie -1 pour revenir a l'état 0 au prochain
                ↪  appel de geoGuess
177
178         await Confirm(msg, "Non, ce n'est pas cette région :( ")
179         # si la region n'est pas la bonne, on reste a la même étape (nouvel
            ↪  essai de joueur)
180         return 0
181
182
183 async def Confirm(msg, txt):
184     """
185     - txt : String
186     - msg : discord.message.Message
187
188     envoie le message 'txt' dans le canal du message recu
189     """
190     await msg.channel.send(txt)
191
192
193 async def showLeaderboard(path, msg, nblines=10):
194     """
195     - path : String, chemin de la BDD
196     - msg : discord.message.Message
197     - nblines : int, nombre de lignes du tableau a afficher
198
199     affiche le Leaderboard par odre décroissant de score
200     (score = nombre de réussite au jeu geoGuess)
201     """
```

```
202
203     def f(line):
204         return -1*int(line.strip().split(" ")[1])
205
206     file = open(path, "r")
207     lines = file.readlines()
208     print(lines)
209     file.close()
210
211     lines.sort(key=f)
212     nblines = min(len(lines), nblines)
213     for i in range(nblines):
214         infos = lines[i].strip().split(" ")
215         id, score = infos[:2]
216         try:
217             username = await client.fetch_user(id)
218             await Confirm(msg, str(username) + " : " + score)
219         except (discord.errors.NotFound, discord.errors.HTTPException) as e:
220             # si le joueur n'est pas sur le serveur
221             await Confirm(msg, "Joueur Inconnu : "+score)
222
223
224 if __name__ == '__main__':
225     compteur = 0
226     path = "./scores.txt"
227     client = discord.Client()
228
229     @client.event
230     async def on_ready():
231         print("Bot is running")
232
233     @client.event
234     async def on_message(msg):
235         userid = msg.author.id
236         username = str(await client.fetch_user(userid))
237         content = msg.content.lower()  # rendre les commandes non if
            ↪ content==-sensitive
238         tmp = content.split(" ")
239         command = tmp[0]
240         global compteur
241
242         # on ne veut pas que le bot interagisse avec ses propres messages
```

```
243         if str(userid) == "967730112999071784":
244             return None
245         # if not command
246         if content[:2] != "??":
247             return None
248         if content=="ping":
249             await Confirm(msg,"Pong !")
250 
251         # geoguess
252         if content == '??play' or (compteur > 0 and content[:7] == "??guess")
            ↪  or content == "??stop":
253             if content == "??stop":
254                 await Confirm(msg, "Dommage !")
255                 compteur = 0
256 
257             else:
258                 rep = await geoguess(msg, compteur)
259                 compteur += rep  # 0 ou 1, ( next step ou non)
260 
261         if command == "??showleader":
262             # appelle showleaderbord avec les arguments correspondants
263             lenght = math.inf
264             if len(tmp) == 2:
265                 try:
266                     lenght = int(tmp[1])
267                 except ValueError:
268                     await Confirm(msg, "Argument invalide ignoré")
269 
270             await showLeaderboard(pathbddGeo, msg, nblines=lenght)
271 
272         if command == "??score":
273             if len(tmp) == 1:
274                 await Confirm(msg, "Ton score est :{}".format(get_score(path,
                    ↪  str(msg.author.id))))
275 
276             else:
277                 id = content.split(" ")[1][2:-1]
278                 print(id)
279                 try:
280                     x = await client.fetch_user(id)
281                     print(x)
282                     score = get_score(path, id)
```

```
283                     await Confirm(msg, "Son score est:" + str(score))
284
285                 except (discord.errors.HTTPException, discord.errors.NotFound)
                    ↪  as e:
286                     await Confirm(msg, "User not found")
287
288         if content == "??help":
289             # affiche la liste des commandes possibles
290             txt = "Liste des commandes : score, showleader, play, help, ping"
291             await Confirm(msg, txt)
292
293     client.run(TOKEN)
```