



INSTITUT  
POLYTECHNIQUE  
DE PARIS

*Projet Informatique de 1<sup>ère</sup> Année à Télécom SudParis*

*PRO-3600*

---

**Sujet**

Fredo Image GeoGuessing

**Membres**

Nathan Féret

Pierre Domachowski

Nour Rammal

Charles Meyer

Théophile Schmutz

**Enseignante responsable** Elisabeth Brunet

**Date de soutenance** 30 Mai 2022

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Cahier des charges</b>	<b>3</b>
<b>3</b>	<b>Développement</b>	<b>5</b>
3.1	Analyse du problème et spécification fonctionnelle . . . . .	5
3.2	Conception préliminaire . . . . .	5
3.2.1	Collecte des données . . . . .	5
3.2.2	Conception et entraînement du CNN . . . . .	6
3.2.3	Mise à disposition de l'outil sur une interface Web . . . . .	7
3.2.4	Interaction avec l'IA . . . . .	8
3.2.5	Tests unitaires . . . . .	8
3.3	Conception détaillée . . . . .	8
3.4	Codage . . . . .	9
3.5	Tests d'intégration . . . . .	9
<b>4</b>	<b>Manuel d'utilisateur</b>	<b>10</b>
<b>5</b>	<b>Conclusion</b>	<b>11</b>
	<b>Appendices</b>	<b>13</b>
	<b>Annexe A : Code source</b>	<b>13</b>
.1	Programme Master - GetDataBaseColab.py . . . . .	13
.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

# Chapter 1

## Introduction

Notre projet a été très librement inspiré par le jeu *Geoguessr*<sup>1</sup>. 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 ou, dans certains cas, 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<sup>2</sup> 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. Boulder 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

---

<sup>1</sup><https://www.geoguessr.com/>

<sup>2</sup> *Convolutional Neural Networks* - réseau de neurones convolutifs

cette technologie pourrait être d'aider à identifier un emplacement avec une connexion faible 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.

## Chapter 2

# Cahier des charges

Le projet devra comporter les points suivantes :

1. la première étape implique la carte et la définition des grilles géographiques à partir desquelles collecter les données. Pour ce projet, la carte de la France sera choisie. Elle sera divisée en 13 parties en suivant le découpage des régions actuelles.
2. les données (images et métadonnées) sont extraites de *Google Street View*. Cela se fera à l'aide de l'API *Google Street View*.
3. une fois le pipeline de collecte de données prêt, les données seront extraites puis triées afin d'éliminer les photos peu pertinentes pour l'entraînement de l'IA.
4. Puis, l'entraînement sera effectué à l'aide des techniques d'apprentissage automatique CNN couplé à des modèle de *computer vision* (*ResNet 50*).
5. l'entraînement de ce réseau sera lourd en calculs. Pour rendre la formation efficace, le réseau sera formé par lots, les images étant récupérées pendant la formation.
6. afin qu'Image *GeoGuessing* soit utilisable à plus large échelle, une application sera développée. Celle-ci permettra de prendre une photo, puis de l'analyser via notre IA pour au final retourner la région dans laquelle l'utilisateur a pris la photo.

Mise à jour du cahier des charges : suite aux divers problèmes que nous avons rencontré, nous avons dû procéder à une mise à jour de notre cahier des charges.

1. l'application sera remplacée par un bot *Discord* qui interagira avec l'IA via un site web. Le bot lancera un jeu, où un joueur affrontera l'IA. Il affichera une image demandera la région au joueur et à l'IA, comparera les réponses avec la vraie localisation de l'image et modifiera les scores en conséquence. Il aura aussi la possibilité de stocker les meilleurs joueurs dans un tableau des scores pour les afficher. Enfin, pour les utilisateurs moins avertis, une fonction *Help* sera disponible pour récapituler les principales fonctions à utiliser.
2. Nous avons donc aussi développé un site web permettant à l'utilisateur de rentrer une image et d'obtenir en sortie la région correspondant au lieu où l'image a été prise. Ce site

aura aussi des pages réservées à la présentation de l'équipe et du projet ainsi qu'une page d'accueil pour rediriger rapidement les utilisateurs.

## Chapter 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 *trainset* 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*<sup>1</sup> ainsi que via un site web où 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 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 une carte 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

---

<sup>1</sup>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* <sup>2</sup>.

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

### 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*<sup>3</sup> à l'aide de la fonction *tf.keras.preprocessing.image.loading*. Le tableau est composé de valeurs RVB et a la forme suivante (300, 600, 3). Pour l'entraînement, les numéros de grille 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 *MNIST*<sup>4</sup> CNN pouvant être entraîné,

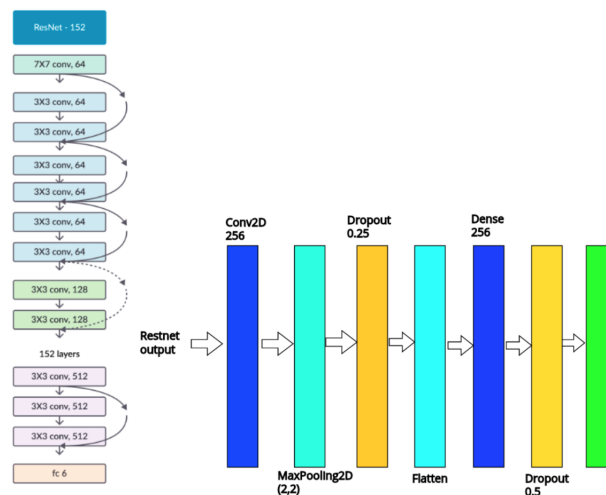
La première partie du modèle est le modèle pré-entraîné *ResNet*. *ResNet*, abréviation de *Residual Networks*, est un réseau de neurones utilisé comme modèle pour de nombreuses tâches de vision par ordinateur. Ce modèle a remporté le défi *ImageNet* en 2015. La propriété fondamentale du *ResNet* est 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*<sup>5</sup> *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 2 ci-dessous.

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

<sup>3</sup><https://numpy.org/> - bibliothèque pour langage de programmation Python

<sup>4</sup>Mixed National Institute of Standards and Technology - jeu de données très utilisé en apprentissage automatique

<sup>5</sup>bibliothèque open source écrite en python, dépend de *TensorFlow*



**Figure. 3.1:** Figure 2 : 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*<sup>6</sup>. 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).

### 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 télécharger une image issue de l'ordinateur, 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.

<sup>6</sup>le modèle connaît par coeur les données d'entraînement plutôt que de les comprendre

### 3.2.4 Interaction 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*<sup>7</sup>, 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'implémentation 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.

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.

---

<sup>7</sup> Association de réseau de TSP - <https://www.minet.net>

## 3.4 Codage

La documentation<sup>8</sup> est détaillée sur notre *GitHub* et site :

[cliquer ici pour accéder au \*GitHub\*](#)

[cliquer ici pour accéder au site](#)

## 3.5 Tests d'intégration

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

---

<sup>8</sup>La documentation a été générée sur le site <https://www.sphinx.com>

## Chapter 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,
  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 l'IA est présente)
  1. "play" pour lancer une partie de *GeoGuessing* : Après quelques instants de chargement, une image s'affiche et le bot vous incite à deviner la région où la photo a été prise,
  2. "guess Nom Region" 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

## Chapter 5

# Conclusion

Finalement, ce projet a été très formateur pour nous. Il nous a permis de découvrir de nombreuses nouvelles technologies et de gagner en compétence. Cela à aussi été l'occasion de consolider nos capacité à travailler en groupe déjà développée grâce au PRO3100.

Faire communiquer les différents aspects de ce projet a été un défi plus grand que ce qu'on aurait pu anticiper et nous sommes fiers d'avoir su résoudre problèmes après problèmes jusqu'à mener à bien ce projet.

# Bibliography

<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

```
1  """ Signs a URL using a URL signing secret """
2
3  import hashlib
4  import hmac
5  import base64
6  import urllib.parse as urlparse
7
8
9  def sign_url(input_url=None, secret=None):
10     """ Sign a request URL with a URL signing secret.
11         Usage:
12         from urlsigner import sign_url
13         signed_url = sign_url(input_url=my_url, secret=SECRET)
14         Args:
15         input_url - The URL to sign
16         secret    - Your URL signing secret
17         Returns:
18         The signed request URL
19     """
20
21     if not input_url or not secret:
22         raise Exception("Both input_url and secret are required")
23
24     url = urlparse.urlparse(input_url)
25
26     # We only need to sign the path+query part of the string
27     url_to_sign = url.path + "?" + url.query
28
29     # Decode the private key into its binary format
30     # We need to decode the URL-encoded private key
```



```

31     decoded_key = base64.urlsafe_b64decode(secret)
32
33     # Create a signature using the private key and the URL-encoded
34     # string using HMAC SHA1. This signature will be binary.
35     signature = hmac.new(decoded_key, str.encode(url_to_sign), hashlib.sha1)
36
37     # Encode the binary signature into base64 for use within a URL
38     encoded_signature = base64.urlsafe_b64encode(signature.digest())
39
40     original_url = url.scheme + "://" + url.netloc + url.path + "?" +
41     ↪ url.query
42
43     # Return signed URL
44     return original_url + "&signature=" + encoded_signature.decode()
45
46     # Import PyDrive and associated libraries.
47
48     # This only needs to be done once in a notebook.
49     from pydrive.auth import GoogleAuth
50     from pydrive.drive import GoogleDrive
51     from google.colab import auth
52     from oauth2client.client import GoogleCredentials
53
54     # Authenticate and create the PyDrive client.
55     # This only needs to be done once in a notebook.
56     auth.authenticate_user()
57     gauth = GoogleAuth()
58     gauth.credentials = GoogleCredentials.get_application_default()
59     drive = GoogleDrive(gauth)
60
61     !mkdir /content/PRO3600_DATA
62     !mkdir /content/PRO3600_DATA/Images
63     !mkdir /content/PRO3600_DATA/Infos
64
65     from shapely import geometry
66
67     import urllib.parse as parse
68
69     import requests
70     import random
71

```

```

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

```

```

114
115     param = arg1=value1&arg2=value2     """
116
117
118     ↪ url="https://maps.googleapis.com/maps/api/streetview/metadata?{}&location={}".format
119
120     url=url+"&key="+API_KEY # ajout de la 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 def GetImage(req):
129     """
130     req étant composé de bits, on traduit en un array 'usuel'
131     renvoie l'array
132     """
133     bits=req.content # on prend le contenu de la requete
134     b=bits.strip()
135     c=io.BytesIO(b) # on décode les bits
136     img=Image.open(c)
137     return np.array(img) # pour ensuite récupérer sous forme de matrice
138
139
140 def GetRequeteReverseGeo(lat,lng):
141     """
142     - lat : float
143     - lng : float
144
145     envoie une requete a l'api de reverse geocoding pour obtenir les infos sur
146     ↪ l'endroit auquel correspondent les coordonnées
147     """
148
149     ↪ url="https://maps.googleapis.com/maps/api/geocode/json?latlng={},{}".format(lat,lng)
150     url=url+"&key="+API_KEY
151
152     req=requests.get(url) # pas besoin de signer l'url
153     return req

```

```

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

```

```

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

```

```

233
234     print("Image : ",compteur , "Adresse : ", address)
235     req=GetRequeteFromAddress(address) # on prend la street view
236     # (potentiels bugs ? nan tkt)
237
238     img=GetImage(req) # on récup l'image
239
240     # on enregistre sur le drive l'image et le fichier d'infos associé
241
242     nameimg="img{}.png".format(compteur)
243     nameinfo="Info{}.txt".format(compteur)
244
245     saveimg(img,nameimg)
246     saveInfo(geodata,nameinfo)
247
248     ImageFolderID='1hPhwEse70iL1-Q511pqwnJZYgDJsKpdE'
249     saveToDrive(path+"/Images/"+nameimg,ImageFolderID)
250
251     InfoFolderID = '1T_UlkbKyxbu6skvkb1AjhL_jEuLnCW7T'
252     saveToDrive(path+"/Infos/"+nameinfo,InfoFolderID)
253     print("Done")
254     compteur+=1
255     print(compteur)
256
257
258
259
260
261 def createPoly():
262     """renvoie le polygone représentant la france"""
263     Lpoints=[[43.525093, -1.134750],[42.741205, 2.447694],[43.888962,
264     ↪ 2.987746],[43.894815, 6.840072],[46.605900, 5.612790],[48.774532,
265     ↪ 7.390237],[50.630174, 2.315648],[49.137655, -0.284608],[49.585080,
266     ↪ -1.555339],[48.608597, -1.188893],[48.608597, -1.188893],[46.346987,
267     ↪ -0.413643]]
268     PointList=[]
269     for pt in Lpoints:
270         PointList.append((pt[0],pt[1])) #on remplace les listes par des tuples
271
272     poly=geometry.Polygon(PointList)
273     return poly # on renvoie le polygone généré
274

```

```
271 France=createPoly()  
272 main()  
273  
274  
275 assert getcos()[0]<=51
```

---

## .2 Programme DataCleaning - main.py

```

1 import ProduitConvolution as pc
2 import FiltragePhoto as fp
3 import DisplayFunctions as df
4
5 import matplotlib.pyplot as plt
6 import numpy as np
7 from skimage.io import imread, imshow
8
9 import shutil
10
11
12
13 plt.close()
14
15 # ----- Constantes -----
16 chemin = "../ImagesBDD/"
17 cheminBDD = 'C:\\Users\\utilisateur\\Documents\\Télécom SudParis\\Cours S6\\DEV
   ↪ Info\\ImagesBDD'
18 fig = 0
19 L=[]
20 Lvar=[]
21
22 # ----- Variables -----
23
24 seuil = 0.7 # Seuil à partir duquel on considère que l'image est un mur
25 seuilVar = 0.03
26 eps = 20 # Marge d'acceptabilité des couleurs
27 iterationsFin = 20000
28 # Afficher ou non l'image et histogramme de chaque mur
29 cond = False
30
31 # ----- Tests | Cellule 1 -----
32 for i in range(iterationsFin):
33     string = chemin + 'img' + str(i) + '.png'
34
35     try :
36         image = imread(string, as_gray=True)
37         Lvar.append(np.var(image))
38
39         if fp.detectWall(image,seuil,eps):

```



```

40
41     if cond : # Afficher les images et histogrammes 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 histogrammes de chaque mur ?
70                     image = imread(string, as_gray=True)
71                     df.imageHist(image)
72
73                 cheminBinImage = str(cheminBDD+ '\\Murs\\img'+ str(i) +
74                     ↪ '.png')
75                 cheminBDDImage = str(cheminBDD+ '\\img'+ str(i) + '.png')
76                 shutil.move(cheminBDDImage, cheminBinImage)
77                 L[i] = True
78             except : # Si la photo n'existe pas
79                 pass

```

### .3 Programme DataCleaning - DisplayFunctions.py

```

1 import ProduitConvolution as pc
2 import FiltragePhoto as fp
3 import DisplayFunctions as df
4
5 import matplotlib.pyplot as plt
6 import numpy as np
7 from skimage.io import imread, imshow
8 from skimage import exposure
9
10
11
12 def imageHist(image):
13     """
14     - image : np.array
15
16     Génère et affiche l'histogramme d'une image, qu'elle soit en nuance de
17     ↪ gris ou en couleurs
18
19     Est appelé par : showGrey
20     """
21     _, axis = plt.subplots(ncols=2, figsize=(12, 3))
22     if (image.ndim == 2):
23         # Grayscale image
24
25         axis[0].imshow(image, cmap=plt.get_cmap('gray'))
26         axis[1].set_title('Histogram')
27         axis[0].set_title('Grayscale Image')
28         hist = exposure.histogram(image)
29         axis[1].plot(hist[0])
30
31     else:
32         # Color image
33
34         axis[0].imshow(image, cmap='gray')
35         axis[1].set_title('Histogram')
36         axis[0].set_title('Colored Image')
37         rgbcolors = ['red', 'green', 'blue']
38         for i, mycolor in enumerate(rgbcolors):
39             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
49     ↪ filtrée
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

```

1 import ProduitConvolution as pc
2
3 from skimage.io import imread
4 from skimage.io import imshow
5 from skimage import exposure
6 import matplotlib.pyplot as plt
7 import numpy as np
8 import PIL
9 from skimage import io
10
11
12
13 def detectWall(image,seuil,eps):
14     """
15     - image : np.ndarray
16     - seuil : float
17     - eps : float
18
19     renvoie un booleen valant True si l'image est un mur, selon un critère
    ↪ basé sur la prédominance d'une couleur sur les autres
20     """
21     n,p = np.shape(image)
22
23     # Compte le nombre de pixels qu'il y a pour chaque nuance de gris
24     count = [0]*256
25     for i in range(n):
26         for j in range(p):
27             count[int(image[i,j]*255)] += 1
28
29     # Détermine s'il y a une prédominance d'une couleur sur le mur
30     iMax = count.index(max(count)) # Nuance de gris du pixel le plus présent
    ↪ sur l'image en nuance de gris
31     compteur = 0
32     borneSup = min(iMax+eps,256)
33     if iMax-eps < 0 :
34         borneInf = 0
35     else :
36         borneInf = iMax - eps
37
38     # 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) # Calcul de la moyenne
42
43     # Critère de prédominance
44     if compteur > seuil: # Les pixels sont quasi tous à peu près de la même
45         ↪ couleur
46         return True
47     else :
48         return False
49
50
51 def detectVar(image, seuil):
52     """
53     - image : np.ndarray
54     - seuil : float
55
56     renvoie un booléen valant True si l'image est un mur, selon un critère
57     ↪ basé sur la variance de l'image
58
59     """
60     # Critère de variance
61     if np.var(image) < seuil:
62         return True # On considère que l'image est un mur
63     else :
64         return False
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

```

1  # Produit de Convultion
2
3  from skimage.io import imread, imshow
4  from skimage import exposure
5  import matplotlib.pyplot as plt
6  import numpy as np
7  import PIL
8  from skimage import io
9
10
11
12 def convuLocal(portionImage,filtre):
13     """
14     - portionImage : np.ndarray
15     - filtre : np.ndarray
16
17     - Génère ValueError si les matrices ne sont pas de même taille
18
19     Renvoie le produit le convolution entre les deux matrices
20     """
21
22
23     if np.shape(portionImage)!=np.shape(filtre):
24         raise ValueError("Matrices de dimensions différentes")
25
26     else:
27         n,p = np.shape(portionImage)
28         somme = 0
29         for i in range(n):
30             for j in range(p):
31                 somme += filtre[i][j]*portionImage[i][j]
32
33         return somme
34
35
36
37 def convuProduct(image):
38     """
39     - image : np.ndarray de dimension n,p
40

```

```

41     renvoie le produit de convolution de image par le filtre f de dim(3x3)
42
43     f = -1, 0, 1,
44         -1, 0, 1,
45         -1, 0, 1,
46
47     """
48     n,p = np.shape(image)
49
50     dimFiltre = 3                                # Filtre est de la forme :
51     filtre = np.array([[ -1,0,1]]*dimFiltre) #      -1 0 1
52     matrice = np.zeros((n,p))                  #      -1 0 1
53     halfDimFiltre = int(dimFiltre/2)          #      -1 0 1
54
55     # On applique le produit de convolution à chaque matrice extraite 3x3
56     for i in range(1,n-1): # On parcourt toute la matrice sauf les bords
57         for j in range(1,p-1):
58             portionImage=np.zeros([dimFiltre,dimFiltre])
59
60             # Création de la matrice extraite
61             for l in range(dimFiltre):
62                 L = image[i+halfDimFiltre-1][j-halfDimFiltre :j+halfDimFiltre
63                     ↪ +1]
64                 portionImage[l]=L
65
66             # On calcule le produit de convolution entre la matrice extraite
67             ↪ et le filtre
68             resij = convuLocal(portionImage,filtre)
69             matrice[i][j]+=resij
70
71     # On enlève tous les bords pour avoir une matrice de dimension (n-1)x(p-1)
72     L=[]
73     for i in range(1,n-1):
74         L.append(matrice[i][1:-1])
75     return np.array(L)

```



## .6 Programme Master - TrainIA.py

```

1 import gc
2 import numpy as np
3 from PIL import Image
4 import tensorflow as tf
5
6 class Geoguessr:
7     def __init__(self,
8                 inputShape=(300,600,3), gridCount=12,
9                 hidden1=256, hidden2=512):
10
11
12     # load resnet model
13     restnet = tf.keras.applications.resnet50.ResNet50(include_top=False,
14
15                                                         ↪ weights='imagenet',
16
17                                                         ↪ input_shape=inputShape)
18
19     self.model = tf.keras.models.Sequential()
20     self.model.add(restnet)
21
22     # freeze resnet model
23
24     self.model.layers[0].trainable = False
25
26     self.model.add(tf.keras.layers.Conv2D(hidden1, (3, 3),
27     ↪ activation='relu',
28
29                                     input_shape=inputShape))
30     self.model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
31     self.model.add(tf.keras.layers.Dropout(0.25))
32     self.model.add(tf.keras.layers.Flatten())
33     self.model.add(tf.keras.layers.Dense(hidden2, activation='relu'))
34     self.model.add(tf.keras.layers.Dropout(0.5))
35     self.model.add(tf.keras.layers.Dense(gridCount, activation="softmax"))
36
37     self.model.compile(loss=tf.keras.losses.categorical_crossentropy,
38                       optimizer=tf.keras.optimizers.Adam(),
39                       metrics=['categorical_accuracy'])
40
41     Lreg=[
42         "Auvergne-Rhône-Alpes",

```

```

38 "Bourgogne-Franche-Comté",
39 "Bretagne",
40 "Centre-Val de Loire",
41 "Grand Est",
42 "Hauts-de-France",
43 "Normandy",
44 "Nouvelle-Aquitaine",
45 "Occitanie",
46 "Pays de la Loire",
47 "Provence-Alpes-Côte d'Azur",
48 "Île-de-France"]
49
50 def getReg(file):
51     """obtient la région d'un file InfoX.txt"""
52     K= 2
53     lines=file.readlines()
54     file.close()
55     Goodlines=[]
56     for line in lines:
57         if "formatted" in line:
58             Goodlines.append(line)
59
60     Good=Goodlines[-K].split(":")[1].split(",")[0][2:]
61     return Good
62
63 def transfo(i,k):
64     L=[0]*k
65     L[i]=1
66     return L
67
68 geoModel = Geoguessr().model
69 print(geoModel.summary)
70
71 step=200
72 nbmax=19999
73
74 pathimg="../input/imagesfrance/"
75 pathinfo="../input/infosfrance/Infos/"
76
77
78 for batch in range(0,nbmax,step):
79     Lx=[]

```

```

80     Ly=[]
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:
95             z=0
96
97     Lx=np.array(Lx, dtype="float64")
98     Ly=np.array(Ly)
99
100    print(Ly.shape)
101    if len(Ly)>0:
102        geoModel.fit(Lx, Ly, batch_size=5)
103        del Lx
104        del Ly
105        gc.collect()
106
107
108    !pip install tensorflowjs
109    import tensorflowjs as tfjs
110    tfjs.converters.save_keras_model(geoModel, "./kaggle/working/ModelJS")
111
112
113    def getCorr():
114        dico=dict()
115        for i in range(19999):
116            try:
117                info=open(pathinfo+"Info{}.txt".format(i))
118                reg=getReg(info)
119                info.close()
120                dico[i]=reg
121            except (FileNotFoundError, ValueError) as e:

```

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

## .7 Programme Master - headlessBrowser.py

```

1 from time import sleep
2 from selenium import webdriver
3
4 options = webdriver.FirefoxOptions()
5 options.add_argument("--headless")
6 # options.add_argument("--disable-dev-shm-usage")
7 # options.add_argument("--disable-gpu")
8
9
10 path = "/home/nferet/tests/img.png"
11
12
13 def GetRepIA(path):
14     """
15     - path : String
16
17     Crée un browser en mode headless ( sans interface graphique)
18     charge la page et upload l'image située au chemin path
19     renvoie la liste contenant les probas de chaque région
20
21     """
22     def toFloat(String):
23         if "e" in String:
24             return 0
25         return float(String)
26     url = "https://geoimage.000webhostapp.com/AI.html"
27     driver = webdriver.Firefox(options=options)
28     print("driver created")
29     driver.get(url)
30     print("url fetched")
31     y = driver.execute_script("return res.toString();")
32     print("y=", y)
33     sleep(1)
34     input_element = driver.find_element(by="id", value='IAImage')
35     input_element.send_keys(path)
36     sleep(1)
37     button_element = driver.find_element(by="id", value='startIaButton')
38     button_element.click()
39     x = driver.execute_script("return res.toString();")
40     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

```
1 from headless_browser_vm import GetRepIA
2 from config import TOKEN, ListNum
3 import discord
4 import random
5 import math
6 import numpy as np
7
8 Lregions=[
9     "Auvergne-Rhône-Alpes",
10    "Bourgogne-Franche-Comté",
11    "Bretagne",
12    "Centre-Val de Loire",
13    "Grand Est",
14    "Hauts-de-France",
15    "Normandy",
16    "Nouvelle-Aquitaine",
17    "Occitanie",
18    "Pays de la Loire",
19    "Provence-Alpes-Côte d'Azur",
20    "Île-de-France"]
21
22 def get_score(path, user):
23     """
24     - path : String
25     - user : discord.user.User
26
27     renvoie le score de "user" dans la base de données placée en "path"
28     """
29     file = open(path, 'r')
30     lines = file.readlines()
31     file.close()
32
33     for i in range(len(lines)):
34         line = lines[i]
35         line = line.strip()
36         line = line.split(" ")
37         name = line[0]
38         if name == str(user):
39             return int(line[1])
40
```

```
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
148             ↳ Quelques instants")
149         global region_ia
150         region_ia = get_REG_IA(path_img)
151         await msg.channel.send("Dans quelle région de France a été prise cette
152             ↳ photo ?", file=discord.File(path_img))
153         await Confirm(msg, "??guess NomRegion pour répondre")
154         await Confirm(msg, "??stop pour arreter")
155         return 1
156
157     if stage == 1:
158         # a l'étape 1 on attend encore la réponse du joueur
159
160         print(content)
161
162         try:
163             indice = Lregions.index(content)
164
165         except ValueError:
166             # 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 reçu
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 ordre 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
238         ↪ content==--sensitive
239         tmp = content.split(" ")
240         command = tmp[0]
241         global compteur
242
243         # 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)
286         ↪ as e:
287         await Confirm(msg, "User not found")
288
289     if content == "??help":
290         # affiche la liste des commandes possibles
291         txt = "Liste des commandes : score, showleader, play, help, ping"
292         await Confirm(msg, txt)
293
294 client.run(TOKEN)
```