Projet JAVA – Module encadré par Gaël Pegliasco

creation du jeu : FRANCLE



BARAKZOY Guiti & HIVIN Clément

Année universitaire 2021-2022

Table des matières

[I- Description du projet 2](#_Toc112765076)

[II- Notice d’installation : Comment le jeu fonctionne 4](#_Toc112765077)

[III- Choix des librairies 5](#_Toc112765078)

[IV - Algorithmes utilisés 6](#_Toc112765079)

[V- Structures de données 11](#_Toc112765080)

[VI- Manuel utilisateur : Le guide d’utilisation 12](#_Toc112765081)

[*A)* *Les avancés du projet :* 12](#_Toc112765082)

[VII - Documentation du code 14](#_Toc112765083)

# 

# **I- Description du projet**

La spécificité du master MIMO – certains diront sa force – réside dans son inclination à accueillir des étudiants en voie de réorientation. De fait, dans un monde de plus en plus spécialisé, les approches pluridisciplinaires et pédagogiques des compétences sont essentielles. Au sein de notre binôme, nous partageons par le prisme de notre formation initiale notre volonté première, qui était d’enseigner, de transmettre, partager. Notre connaissance fine du monde de l’éducation nous permet d’en cibler les points perfectibles. Notre réorientation nous permet aujourd’hui, à travers la programmation en Java, de tâcher d’y répondre, en développant des outils pédagogiques. Ainsi, le projet Java constituait l’occasion parfaite de proposer un « jeu sérieux », pouvant être utilisé par des élèves en fonction des objectifs établis par l’enseignant. Nous pourrions l’imaginer en ouverture d’un cours, pour permettre une mise en route en mobilisant les connaissances et les compétences des élèves. Nous pourrions aussi l’imaginer manier par des utilisateurs quelconques, en quête de savoir, pour le plaisir d’apprendre.

Avant de décrire le projet Java, expliquons ce qu’est un jeu sérieux. La définition proposée par le Centre de Ressource et d’Informations sur le Multimédia pour l’Enseignement Supérieur (CERIMES) indique qu’il s’agit *«****d’applications développées*** *à partir des* ***technologies avancées du jeu vidéo****, faisant appel aux mêmes approches de design et savoir-faire que le jeu classique mais qui* ***dépassent la seule dimension du divertissement****».* Deux types d’activités en alternance ou en association se rejoignent : des phases ludiques et des moments d’apprentissage. Les thématiques peuvent être variées.

Dans notre cas, il s’agirait d’un jeu sérieux sur le thème de la géographie. Notre jeu, nous l’avons nommé Francle, et il tire son inspiration du jeu Worldle (https://worldle.teuteuf.fr/).

Francle propose la carte d’une région française. L’objectif pour l’utilisateur est de trouver en cinq tentatives à partir de la carte, de quelle région il s’agit. Après chaque tentative échouée, des indices telles que la distance ou des spécificités de la région en question seront donnés. L’envie de recommencer une nouvelle carte après avoir nommé la précédente trouvera sa source dans la propension de l’utilisateur à rechercher le plaisir de la réussite, dans la satisfaction de connaissances en cours de consolidation. Nous avons conscience que la méconnaissance totale des cartes régionales peut mener à une frustration de l’utilisateur. C’est pour cela que nous avons pensé ce jeu principalement pour une utilisation pédagogique.

Nous avons fait le choix de représenter dans un premier temps les régions dans leur configuration géographique depuis la réforme de 2016. C’est seulement dans un deuxième temps que l’utilisateur pourra deviner le nom des anciennes régions, plus nombreuses.

Nous avons utilisé GitHub pour réaliser ce projet. Il s’agit d’un service de *cloud computing* permettant aux développeurs de stocker et gérer l’évolution du code. Cet outil illustre la puissance du *cloud* pour travailler à plusieurs sur un même projet en passant outre la limitation géographique – chacun peut contribuer au projet d’où qu’il soit à condition qu’il ait un accès à internet. Dans notre cas, GitHub nous a permis de travailler sur le fichier que nous avons nommé *main.java*, dans lequel nous avons pu, par nos efforts combinés, suivre le code, le modifier, en observer l’avancement au travers les mises à jour du fichier. Ainsi, nous nous sommes familiarisés progressivement avec ce nouvel outil, et ce n’est que sur la fin du projet que nous nous sommes rendu compte qu’il y avait des fonctionnalités plus agiles telles que « *pull request* » et « *forks* » ; permettant de repérer exactement les changements dans le code. Nous pourrons optimiser notre temps en utilisant ces fonctionnalités sur un futur projet.

Le lien ci-dessous permet d’accéder au GitHub sur lequel nous avons travaillé. Il présente différents formats de documentation, les plus intéressants étant les suivants :

* L’ébauche du code, expliquant dans les grands axes l’algorithme utilisé : test d’algorithme.java
* La totalité du code : main.java
* Un dossier contenant les images au format PNG de chaque région
* Un document contenant les indices cités pour chaque région
* Le diagramme UML
* Le présent document

https://github.com/deIbine/francle

# **II- Notice d’installation : Comment le jeu fonctionne**

Il n’y a pas d’installation spécifique à prendre en compte pour ce jeu. Le seul élément à prendre en compte est la ligne de code sur le chargement des images. En effet, l’utilisateur devra, pour bien faire fonctionner le programme, changer la ligne de code 1420 en fonction de l’endroit où il a mis les fichiers sur sa machine. L’idée de demander le chemin d’accès à l’utilisateur à chaque fois a été explorée, mais abandonnée par volonté d’ergonomie. Dans le futur, nous améliorerons le code pour empêcher cette configuration demandée à l’utilisateur.

Lorsque l’utilisateur accède à Francle, un menu s’affiche. Il devra choisir le numéro 1 pour accéder au jeu, dans ce cas, il se trouve devant une carte de la région de France (les autres fonctionnalités du menu sont expliquées ci-dessous dans la partie 6).

Un message lui indique qu’il doit écrire le nom de la région correspondant, si le nom indiqué est incorrect, un message d’erreur apparaît. Sinon, plusieurs cas de figure se présentent :

* Si la réponse est fausse, un message lui indique qu’il peut recommencer. Il a alors plusieurs indices :
  + La distance, qui est calculée en fonction de la latitude et la longitude des deux préfectures de la région. Plus l’utilisateur est proche de la réponse, plus la distance sera courte. Elle est indiquée en kilomètre.
  + Un pourcentage permettant d’indiquer la proximité en fonction de la distance de la région choisi par l’utilisateur et la région cible. Plus l’utilisateur est proche de la réponse, plus le pourcentage est élevé.
  + Des indices supplémentaires sont apportés seulement pour les 3 dernières tentatives :
    - 3ème tentative : Il s’agit de la spécialité culinaire
    - 4ème tentative : La capitale régionale
    - 5ème tentative : Personnalité provenant de la région
* Si au bout de 5 tentatives, la réponse est toujours fausse, un message lui indiquant qu’il n’y a pas la possibilité d’effectuer une supposition supplémentaire apparaît en plus de la réponse.
* Si la réponse est correcte, en plus du message le félicitant, les messages suivants apparaissent :
  + La distance est de 0km
  + Le pourcentage indique 100%

# **III- Choix des librairies**

Pour le codage de Francle, nous utilisons la librairie standard appelé JAVA SE (Standard Edition). Cette librairie est organisée en packages dans lesquels on trouve des classes souvent utilisées dans ce langage de programmation telles que « *java.lang* », qui contient des classes de bases comme String, System, etc... ou par exemple « *java.math* » permettant d’effectuer des calculs.

L’utilisation de ces packages se fait à travers la déclaration d’import au début du programme, à l’extérieur de toutes les classes.

Dans notre cas, nous utilisons plusieurs packages :

* ***Java.lang.Math*** va permettre l’utilisation de fonctions mathématiques au sein du programme. Nous l’utilisons notamment à travers les fonctions trigonométriques.
* ***Java.util.Scanner*** est un outil permettant la simplification d’expression utilisée régulièrement dans le programme. Dans notre code, on associe le scanner à la méthode *nextInt,* pour extraire un nombre entier. On l’associe également à la méthode *nextLine* pour lire la ligne du texte jusqu’au prochain retour à la ligne.
* ***Javax.swing.\****et***Java.awt.\**** : sont des outils semblables permettant la création d’interface graphique. Ils peuvent servir pour faire apparaître la fenêtre par exemple.
* ***Java.awt.image.BufferedImage*** joue un rôle dans le traitement des images. Dans notre cas, il s’agit des différentes cartes régionales. Cet outil va gérer l’image qui se situe dans la mémoire, fournir des méthodes de stockage et permettre l’obtention des données en pixels.
* ***Java.io.File***permet la manipulation du fichier. Aussi, c’est un outil qui peut représenter un chemin sur le système.
* ***Java.io.IoException***comme son nom l’indique, il s’agit d’exception d’entrée et de sortie. Elle se produit lorsqu’une opération d’entrée ou de sortie échoue. Lorsqu’un fichier n’est pas reconnu lors de sa lecture une exception d’E/S est alors lancé.
* ***Java.util.Objects*** permet d’opérer sur des objets**:** pour calculer le code de hachage d’un objet, comparer 2 objets ou encore renvoyer une chaîne de caractères pour un objet.
* ***Java.awt.image.ImageObserver*** et ***Java.awt.image.ImageProducer*** : permettent le suivi de la procédure de chargement de l’image. Ils permettent de manipuler l’état de chargement de l’image en fonction de la demande.

# **IV - Algorithmes utilisés**

Comme cité plus haut, le document appelé « *test d’algorithme.java* » sur le GitHub compile les premières idées de codage pour ce projet. Nous tâcherons dans cette partie de fournir l’explication de ce code en justifiant le choix de l’algorithme utilisé avec le principal code que l’on retrouve dans le fichier « *main.java* ».

L’objectif de « test d’algorithme.java » était de bien délimiter le rôle de chacune des classes. La première classe que l’on trouve sur les 2 codes est la **classe main**. Elle correspond principalement à notre menu dans Francle et permet d’obtenir l’interface graphique de l’image. C’est ce qui réfère vers les méthodes qui font fonctionner le code.

**Classe Parametres** : Correspond à une classe Java classique dans laquelle on trouve des variables en privée, des méthodes, des getters et des setters. Rappelons que les getters et les setters sont des méthodes utilisées dans des classes afin de lire (getter), ou d’affecter/modifier (setter) des valeurs d’une autre classe.

Les getters permettent d’avoir un accès aux champs d’une classe ou d’une méthode en *private*, tout en gardant la protection. Cela permet de lire des attributs protégés dans des classes ou des méthodes. C’est pratique dans le cas où l’on a juste besoin de lire l’information en question et pas besoin de l’accès direct à cette information via une protection en public.

Les setters sont pour l’affectation des données. Ils sont utiles pour affecter des valeurs qui sont en *private*, garder la protection, et comme c’est une méthode, on peut tout à fait mettre des conditions aux affectations tentées par l’utilisateur (par exemple, empêcher une valeur négative pour un poids). En plus de cela, cela permet de garder les affectations d’un objet dans la même méthode que l’objet en question dans les deux cas, et donc de respecter l’encapsulation.

3 types de paramètres ont été implémentés dans cette classe :

* Le paramètre permettant d’afficher ou non la liste des noms de Région.
* Le paramètre permettant de choisir les régions en fonction de la réforme de 2016 : le choix 0 correspond aux régions avant la réforme, le choix 1 correspond aux régions actuelles, le choix 2 rassemble toutes les régions.
* Le paramètre permettant de cacher ou non l’image de la carte

Dans le fichier test d’algorithme.java, on peut remarquer la présence d’autres fonctionnalités comme le fait de remplacer les km en miles, le fait de changer le thème de l’interface, changer le langage, changer le sens de l’image. Toutes ces fonctionnalités n’ont pas été codées sur la version actuelle. Mais participe aux avancés que l’on souhaite apporter prochainement sur ce projet.

On se trouve ensuite devant la **classe Region**. Plusieurs choix de modélisation s’offraient à nous pour cette classe. Comme on peut le remarquer sur « test d’algorithme.java », cette classe aurait pu être codé avec la méthode classique en java dans laquelle on aurait un switch avec les longitudes et latitudes de chaque région qui seraient initialisés ensuite dans la classe main. Or, nous pouvons remarquer pour des questions de malléabilité du code l’absence de variable dite « *private*». Cela induit l’absence de « *getter* » et de « *setter* », qui ont pour rôle d’associer une action chaque fois qu’ils sont appelés. Ainsi, il n’y a pas de variable à manipuler. En revanche, le nombre de méthodes dans cette classe est plus importante. Cette architecture est un choix que nous avons fait. Nous avons conscience qu’on aurait pu effectuer une classe jeu qui aurait récupéré certaine méthode présente dans la classe région. Les méthodes sont utiles pour le bon fonctionnement du code.

* La **méthode aleatoire**: En fonction du mode choisi par l’utilisateur, un chiffre aléatoire est choisi et renvoie ensuite vers la méthode choixpossible qui donne la région.
* La **méthode choixpossible** : Dans le *switch*, on récupère la région en fonction du nombre aléatoire, si le mode est à 1, on obtient alors les régions depuis la réforme de 2016.
* La **méthode jeu** : On demande la supposition auprès de l’utilisateur. Si l’utilisateur note 1, que le mode de jeu est à 1 comme ci-dessus, mais que le paramètre de la liste n’est pas paramétré, alors la liste lui sera proposé. On retourne ensuite le compteur pour éviter que ce soit compté comme un essai de l’utilisateur. La saisie n’a pas été reconnue lorsqu’il y a une erreur dans la syntaxe de la région. Si l’utilisateur ne parvient pas à trouver les réponses, des indices telles que la distance et le pourcentage lui sont indiqués. On incrémente le compteur, ce qui permet le renvoie la tentative suivante, si le compteur est à 5, le jeu se termine.
* Les **méthodes touteslesregions et lesregions :** permetd’envoyer toutes les régions quand l’utilisateur le demande. Le codage se fait sur 2 méthodes.

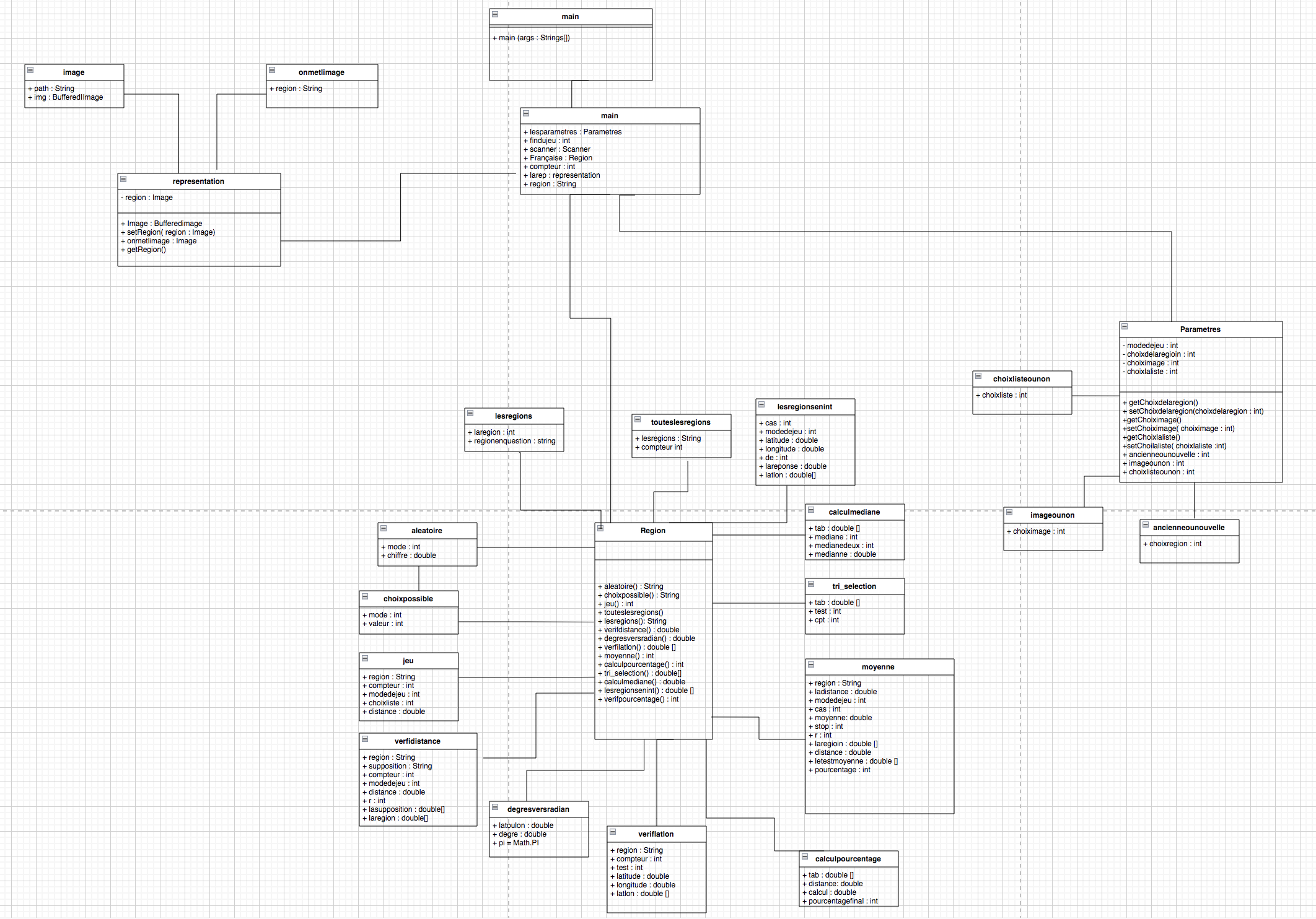
* Les méthodes **verifdistance**, **degreversradian et calculmediane**: Ce sont des méthodes dans lesquelles on retrouve des calculs mathématiques. La méthode degreversradian permet d’être utilisé dans la méthode verifdistance.
* La méthode **veriflatlon** : Permet de récupérer la latitude et longitude qu’on utilisera dans la méthode aléatoire. Aussi, en fonction du compteur qui représente le nombre d’essai de l’utilisateur, pour les 3 dernières tentatives, on lui envoie un indice. La variable test permet de vérifier qu’il s’agit des bonnes régions. A la fin de cette méthode, on renvoie un tableau de nombre réel correspondant à la latitude et longitude qui sera utilisé pour faire les calculs de distance.
* Les **méthodes pourcentage** et **calculpourcentage :** En fonction du mode de jeu choisi, on calcul l’ensemble des distances possible par rapport à la région cherchée. On obtient un tableau de l’ensemble des distances qui permettra de définir la distance la plus longue. C’est à partir de cette distant qu’on calcule le pourcentage qui se calcul par un produit en croix.
* La **méthode triselection** : Prend l’ensemble des distances qui ont été calculé et les range dans un tableau. On obtient un tableau dans lequel les distances sont rangées dans l’ordre croissant. On trouve ainsi la distance maximale dans la dernière case du tableau. Cette distance est utilisée pour le calcul du pourcentage.
* La méthode **lesregionsenint**: Pour tester l’ensemble des distances dans les pourcentages, il a fallu recréer un ensemble de longitude et latitudes, pour ensuite les calculer un par un par rapport à la région aléatoire. Cette méthode se base sur le même principe que la méthode veriflatlon cependant à la place d’avoir des régions dans les « cases » on a des chiffres.

On remarque dans ce code, la présence de calcul de moyenne et médiane pour être utilisé dans le pourcentage de distance, les résultats n’étant pas satisfaisants, on a utilisé la distance maximale.

Finalement, comme prévu initialement, la classe régions est principalement composé de calculs. Cette classe ressemble à un codage utilisé en langage C.

La **classe representation** sera détaillée dans la partie Structure de données ci-dessous.

Pour des raisons de lisibilités de l’algorithme, ci-dessous un diagramme UML :

Diagramme UML représentant l’algorithme de Francle.

# **V- Structures de données**

Rappelons que Francle est un jeu dont le but pour l’utilisateur est de reconnaître une image, plus précisément la région de France à laquelle correspond la carte présentée. il a donc fallu inclure des images dans le code.

Pour permettre cela, les images ont dû être conçues. Nous avons dans un premier temps récupéré les images du site « France.comersis.com » qui propose des cartes libres de droit de la France. Dans un second temps, chaque région de la carte a été colorée et séparée à partir d’un logiciel de traitement d’image. Enfin, nous avons exporté les images au format PNG (Portable Network Graphics). Nous avons utilisé ce format pour des questions de qualité d’image. Effectivement, contrairement au format JPEG, l’image en PNG n’est pas compressée.

Même si cela peut paraître évident, notons que Francle inclut les régions d’outre-mer, trop souvent oubliées des cartes de France.

Expliquons maintenant la partie codage sous-jacente à la présentation des différentes cartes régionales à l’utilisateur.

Premièrement, focalisons notre attention sur la « *class »* Representation :

* On se trouve devant une première méthode qui permet de charger l’image, notamment grâce à « *BufferedImage* » qui va créer une variable Image. C’est cette partie du code qui va être utilisée pour faire la représentation d’image avec notre programme.
* On détermine ensuite une variable « *private* » image.
* On se trouve ensuite devant une autre méthode dans laquelle on prend la région qui a été choisie aléatoirement. Prenons l’exemple suivante. Si l’image qui apparaît s’agit de la Corse, alors le code permet de chercher dans le chemin d’accès l’image correspondante à cette région.

Ensuite, dans la « *class* *main* », plus précisément dans la partie du « *case 1* », nous créons une représentation qui va permettre d’accéder à l’interface graphique et ainsi dévoiler l’image de la région correspondante.

# **VI- Manuel utilisateur : Le guide d’utilisation**

Nous avons fait le choix d’une interface graphique simple, intuitive et ergonomique pour l’utilisateur. Comme indiqué dans la partie 2 de ce document, sur l’écran, l’utilisateur se trouve devant un menu avec plusieurs possibilités :

* **0** pour quitter le jeu.
* **1** pour commencer le jeu. Le joueur se trouve devant la carte de la région à deviner.
* **2** pour configurer les paramètres selon ses envies :
  + Il peut paramétrer **les types de régions** : choisir entre les régions avant ou après la réforme, mais aussi toutes les régions confondues (avant et après la réforme de 2016), dans ce cas l’utilisateur clique sur 1.
  + Il peut **modifier la difficulté du jeu** en choisissant par exemple pour les critères de difficulté de cacher l’image de la carte, le joueur clique sur 2, pour les critères de simplicité on peut afficher une liste indiquant le nom des régions il faudra cliquer sur 3.

## *Les avancées du projet :*

Notre ambition nous pousse à penser la suite de Francle. Aujourd’hui notre jeu fonctionne en local, nous voulons le rendre accessible sur internet avec une Interface graphique ludique qui peut être un facteur clé du succès de Francle.

Nous avons la volonté d’améliorer le code dans les prochains mois à travers GitHub et obtenir un jeu parfaitement fonctionnel, pour cela nous avons réfléchi aux avancées suivantes :

* Implémenter une classe comment\_jouer dans laquelle les règles du jeu seront expliquées. Sur l’interface, cela devra être accessible en cliquant sur un point d’interrogation présent sur la page de la carte de la région.
* Implémenter une classe statistique indiquant le nombre de partie à effectuer, le pourcentage de victoire, la moyenne des meilleures distances, ou encore la répartition des victoires en fonction du nombre d’essai.
* Ajouter dans les paramètres une méthode permettant d’obtenir une flèche indiquant la direction de la région par rapport à celle indiquée.
* Comme expliqué dans la partie 4 – Algorithme utilisé, on pourrait ajouter les paramètres initialement pensés au début de ce projet, mais dont la réalisation demandait plus de temps et réflexion : remplacer les km en miles, le fait de changer le thème de l’interface, changer le langage, changer le sens de l’image.
* On peut également apporter des indices dès la première tentative concernant la région en question, actuellement Francle propose des indices seulement à partir du 3ème essai.
* Le code pourrait être retravaillé pour obtenir un code en Java-like.

# **VII - Documentation du code**

Voir, dans le code, la documentation, qui est entre les commentaires suivants : /\* \*/.