

# Qui est-ce ?

## Projet de programmation

Groupe AA

*Elliot MAZERAND, Loris GARCIA-PENA,*

*MEDOUT-MARERE Paul, ROMANO Tom*

*<https://github.com/GarciaPena-Loris/ProjetProgS4-ETPL>*

L2 informatique

Faculté des Sciences

Université de Montpellier.

8 avril 2022



### Résumé

Tout au long du semestre, il nous a été demandé de réaliser un projet de programmation ayant pour sujet la conception d'un jeu de « Qui est-ce ? ». Pour ce faire, plusieurs étapes ont été nécessaires avant l'aboutissement de ce travail. Dans un premier temps, nous avons effectué des modélisations de l'application afin de mieux anticiper et visualiser les différentes parties qui la composent comme les menus ou le jeu. Ensuite, nous avons programmé un jeu de « qui est-ce » générique, ce dernier nous permettant d'approcher de manière plus précise l'application finale en lui servant de base. Par la suite, nous avons réalisé un générateur permettant aux utilisateurs de concevoir leurs propres grilles de « Qui est-ce ? » qu'il sera possible de sélectionner comme grille dans l'application finale. Enfin, nous avons créé notre application finale en se basant sur le « Qui est-ce ? » générique et utilisant une grille JSON créer via le générateur ainsi que l'ajout d'extensions telles qu'un mode multijoueur.

# Table des matières

<b>1 Technologies utilisées et organisation du Travail</b>	<b>3</b>
1.1 Présentation de la modélisation . . . . .	3
1.2 Organisation du travail . . . . .	5
1.3 Description des technologies utilisées . . . . .	6
<b>2 Présentation du “Qui est-ce” générique</b>	<b>8</b>
2.1 Introduction à l’application . . . . .	8
2.2 Fonctionnement interne . . . . .	10
2.2.1 La difficulté “facile” . . . . .	15
2.2.2 Le système de sauvegarde . . . . .	16
<b>3 Description du Générateur pour Grille de “Qui est-ce”</b>	<b>18</b>
3.1 Présentation du générateur et notice d’utilisation . . . . .	18
3.2 Description technique du générateur . . . . .	25
<b>4 Développement des Extensions “Qui est-ce”</b>	<b>31</b>
4.1 Présentation du mode multijoueur . . . . .	31
4.2 Explication du fonctionnement de l’extension . . . . .	33
<b>5 Conclusion et recul sur le travail effectué</b>	<b>38</b>

# 1 Technologies utilisées et organisation du Travail

## 1.1 Présentation de la modélisation

Pour mener à bien ce projet, nous avons décidé d'utiliser le langage de programmation JAVA. Nous l'avons choisi, car nous le connaissons bien et qu'il nous paraît des plus adaptés afin de réaliser une application comme c'est un langage objet, ce qui nous permet une plus simple et plus compréhensible utilisation des fonctions.

Dans le but de mieux comprendre et anticiper le travail à réaliser, nous avons commencé par modéliser notre projet. Nous avons effectué une série de diagrammes composée d'un diagramme de classe, un diagramme d'utilisation et un de séquence.

Le diagramme de classe présenté ci-dessous nous a permis de mieux visualiser notre classe correspondant au "Qui est-ce". En effet, il contient des informations précieuses pour le jeu comme ses méthodes qui constituent les différentes étapes dans la réalisation du jeu :

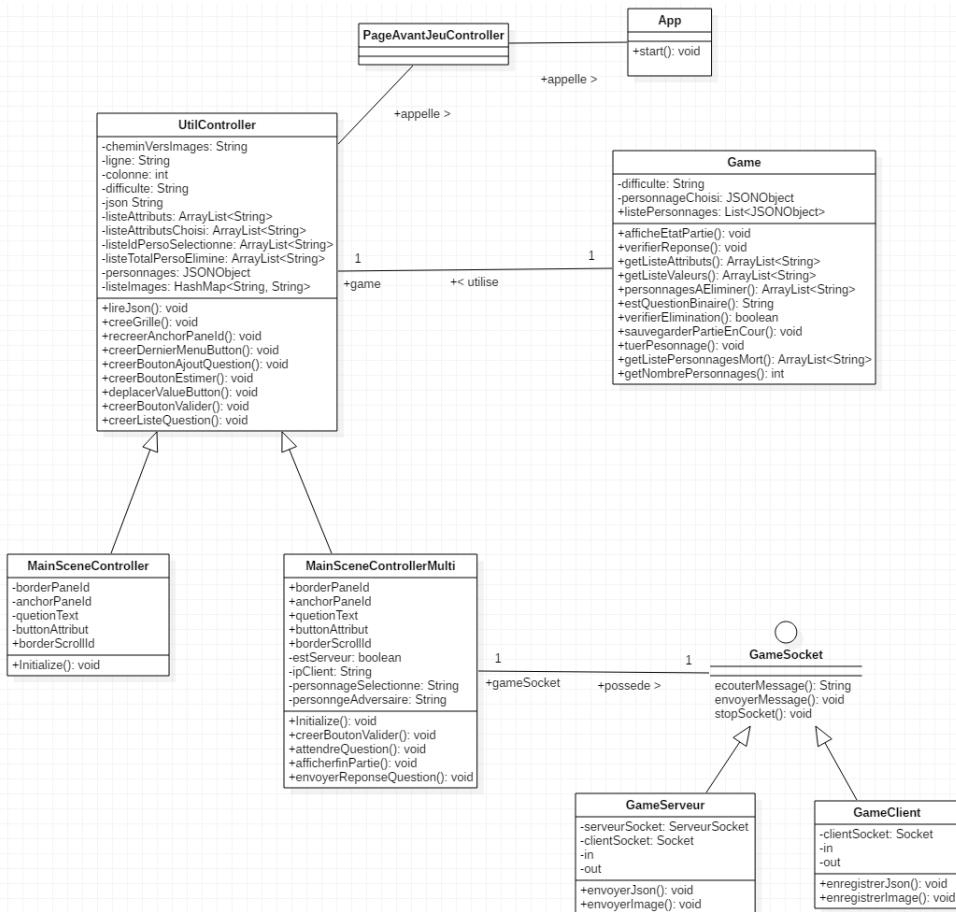
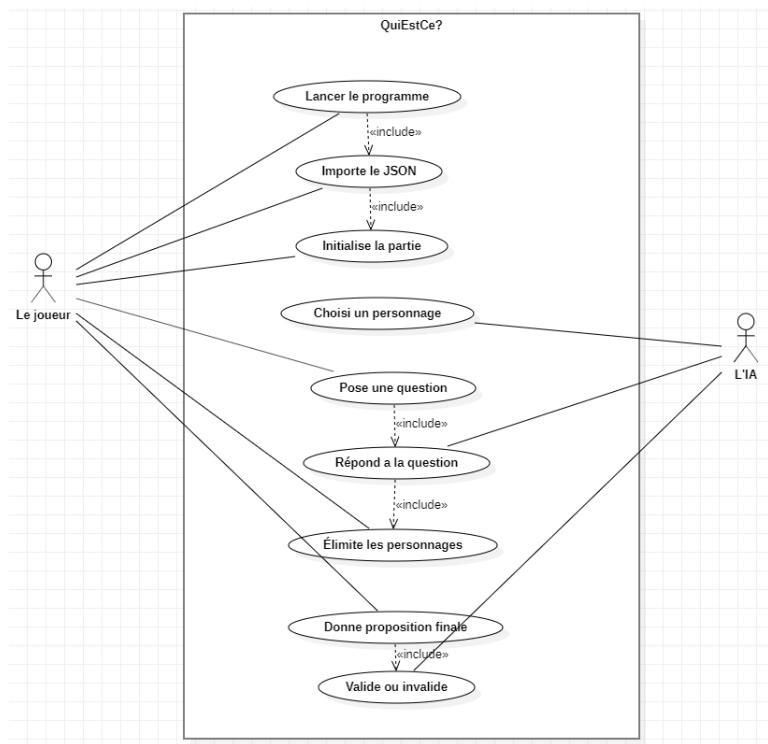


Diagramme de classe

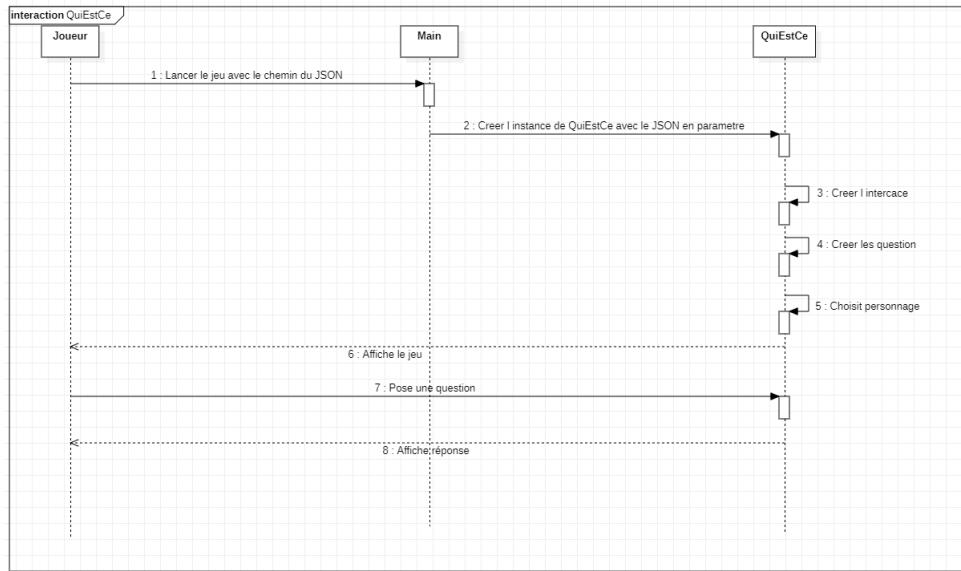
Le deuxième diagramme est un diagramme de cas d'utilisation. Ce dernier nous permet de mieux comprendre les interactions possibles entre l'utilisateur et la machine par le biais de notre jeu bien qu'il n'inclut pas de notion temporelle. En effet, on peut distinguer le jeu en trois grandes parties. La première sera une phase de présentation, où l'on retrouve des étapes comme lancer le programme, importer le json et initialiser la partie. Dans un second temps nous retrouvons toutes les étapes nécessaires pendant que la partie est en cours comme choisir un personnage, poser une question, répondre à la question et éliminer les personnages. Enfin la dernière partie du diagramme présente la fin du jeu avec la proposition finale et enfin la validation de la réponse et la fermeture de l'application. À la différence du diagramme de classe précédent qui été plus concentré sur le jeu, celui-ci nous permet d'intégrer la notion d'utilisateur. Noter que ce diagramme est partiellement correct car nous avons tout de même tenté de représenter le temps de haut en bas ce qui nous a poussé à réaliser un troisième diagramme, prenant cette fois-ci en compte le temps, un diagramme de séquence :



*Diagramme de cas d'utilisation.*

Le diagramme de séquence apporte une valeur temporelle essentielle dans la réalisation d'une telle application. En effet le jeu réalisé étant un "Qui est-ce", il est nécessaire de le décomposer en étapes succinctes permettant l'aboutissement du jeu. Parmi ces étapes on retrouve celles effectuées par l'utilisateur à savoir lancer le jeu, poser les questions. Du côté de l'application

d'autres étapes sont à réaliser comme créer l'interface, créer les questions, choisir les personnages, afficher le jeu et enfin afficher les réponses :



*Diagramme de séquence.*

Ces trois diagrammes nous ont ensuite permis de nous lancer dans la programmation du jeu, car ils compossait nos axes de programmation, primordiale si l'on souhaite ne pas se retrouver en désaccord et hors sujet vis-à-vis du travail à réaliser.

Notez toutefois que ces diagrammes ont été susceptibles d'être modifiés légèrement au cours de notre travail, nous permettant une plus simple compréhension et réalisation du travail demandé.

## 1.2 Organisation du travail

Pour ce qui est de l'organisation du travail nous avons choisi d'utiliser Github, afin de construire un projet de programmation tout en partageant des informations et ceci sans se gêner dans notre travail. En effet, Github offre la possibilité de travailler sur des branches différentes sur lesquelles nous pouvions écrire du code sans que celui-ci ne soit écrit pour les autres, ce qui évite des erreurs de conflits, d'exécution voire de compréhension pour les autres. Nous communiquons également fréquemment sur Discord pour discuter autour des concepts du projet, de sa réalisation mais également effectuer du travail ensemble. Nous avons réparti assez équitablement les tâches au sein du groupe, nous nous retrouvions tous les lundis après-midi durant le créneau horaire prévu à cet effet, ainsi que certain soir de la semaine ou nous convenions d'une heure pour nous rassembler sur un serveur discord afin de faire le point, d'avancer ensemble sur le travail et de discuter de la répartition des nouvelles tâches.

### 1.3 Description des technologies utilisées

Dans le but de mener à bien ce projet, il était important de choisir soigneusement les technologies que nous allions utiliser. Premièrement, le choix du langage, nous avions plusieurs choix bien évidemment et le nôtre s'est tourné vers le langage Java. En effet c'est le langage avec lequel nous étions le plus à l'aise (car étudier dans le cadre de la licence) et de plus nous avions conscience de l'existence de certaines bibliothèques utiles au bon développement du projet. La première bibliothèque notable que nous avons utilisée est JavaFX. JavaFX est une bibliothèque open source permettant la manipulation d'interface utilisateur, nous permettant de créer plusieurs interfaces graphiques de bureau pour le "Qui est-ce ?" (ex. Capture 1 : Page d'accueil de l'application.). Afin de nous faciliter la création de ces susdites interfaces nous avons utilisé un logiciel : SceneBuilder. Ce logiciel permet de créer son interface en faisant glisser des "objets" sur un aperçu de la page que l'on est en train de créer. Ces interfaces sont alors enregistrés avec l'extension de fichier ".FXML" qui est un langage à balise. On peut alors se demander comment faire communiquer le programme et ces interfaces. Cela est très simple, la bibliothèque JavaFx le permet. Précisément, nous pouvons définir un "controller" (voir : Capture : définition d'un controller).

```
fx:controller="pageAvantJeuController">
```

*Capture : définition d'un controller.*

Les "controller" sont des classes java qui ont pour rôles de définir les méthodes liées aux interfaces graphiques (ex. Les boutons, les champs de textes...). A savoir que les "objets" de l'interface graphique sont eux aussi liés à ces méthodes dans le fichier ".FXML" de leurs interfaces (voir : Capture : définition des méthodes d'un bouton).

```
<Button fx:id="buttonchargerpartie" layoutX="448.0" layoutY="50.0" mnemonicParsing="false" onAction="#chargerpartie">
```

*Capture : définition des méthodes d'un bouton.*

La deuxième bibliothèque notable que nous avons utilisée est "JSONObject". Cette fois ci utilisé à la gestion des grilles de personnages. Il s'avère que les descriptifs que le jeu prend en entrée sont au format de fichier ".JSON" il était donc impératif de pouvoir manipuler ce type de fichier avec Java. C'est donc grâce à la bibliothèque "JSONObject" que nous avons pu y parvenir. Cela nous permet donc dans un premier temps de lire le JSON (voir Capture : lecture du fichier ".JSON").

```

JSONObject js = (JSONObject) new JSONParser().parse(new FileReader(json));

cheminVersImages = (String) js.get("images");
ligne = Integer.parseInt((String) js.get("ligne"));
colonne = Integer.parseInt((String) js.get("colonne"));
personnages = (JSONObject) js.get("personnages");

```

*Capture : lecture du fichier “.JSON”.*

Dans un second temps de faire des listes de “JSONObject” qui fonctionne comme des hashmap avec des clé et des valeurs(voir Capture : Crédation d’un JSONObject).

```

JSONObject personnage = (JSONObject) JSONPersonnages.get(String.valueOf(i));

```

*Capture : Crédation d’un JSONObject.*

Et dans un dernier temps à créer un système de sauvegarde que l’on a décidé de faire sous forme de fichier “JSON” (on écrit un fichier grâce à la bibliothèque, voir Capture : création d’une sauvegarde).

```

// sauvegarde de la partie
JSONObject partieSave = new JSONObject();
partieSave.put("images", String.valueOf(images));
partieSave.put("ligne", String.valueOf(ligne));
partieSave.put("colonne", String.valueOf(colonne));
partieSave.put("difficulte", difficulte);
partieSave.put("personnagesChoisi", personnageChoisi);

```

*Capture : création d’une sauvegarde.*

## 2 Présentation du “Qui est-ce” générique

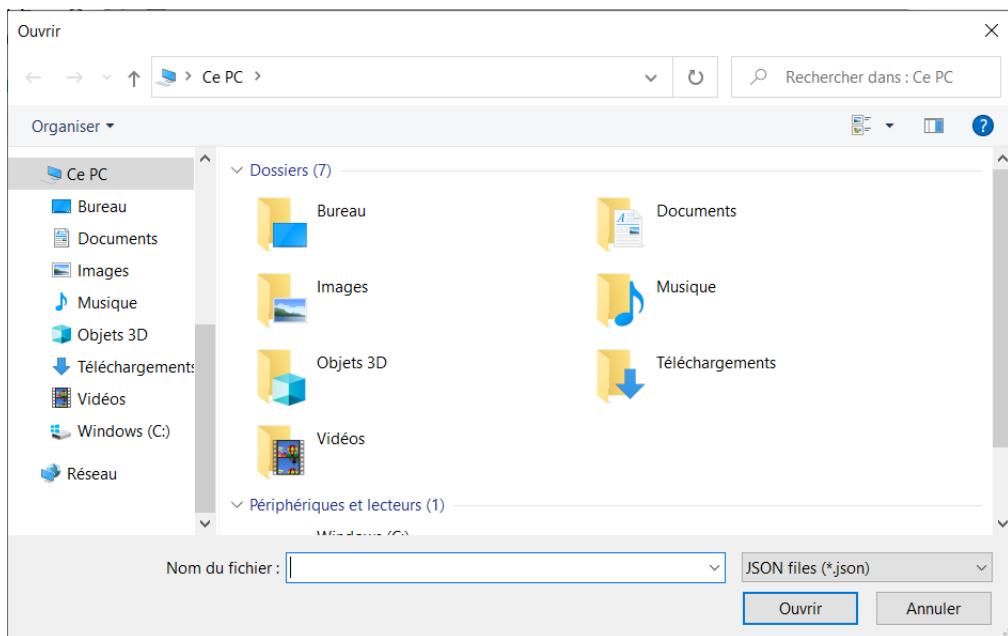
### 2.1 Introduction à l’application

Nous allons tout d’abord vous présenter le “qui est-ce” en lui même, nous commencerons par parler des fonctionnalités de l’application. Nous parlerons ensuite de la forme des requêtes traitées. Suite à ça, nous décrirons au mieux les structures de données, les classes, et les variables qui ont été utilisées. Enfin, nous détaillerons le traitement effectué lors d’une requête. L’application propose à l’utilisateur de nombreuses fonctionnalités, pour commencer il faut exécuter le fichier app.java pour lancer l’application.



*Capture : Page d'accueil de l'application.*

Une fois l’application lancée, il est possible de choisir la difficulté de la partie que l’on souhaite jouer en cliquant sur “choix difficulté”, il en existe deux : facile et normal. Lorsqu’on joue en mode facile, une fois que l’utilisateur a choisi une question, un bouton “estimation” est disponible et permet d’informer l’utilisateur du nombre de personnages concerné par sa question et les sélectionne. Il peut ensuite cliquer sur valider pour avoir la réponse à sa question. Les personnages à éliminer sont alors automatiquement sélectionnés selon la bonne réponse. Il n’y a plus qu’à cliquer sur “valider” pour que les personnages à éliminer le soit automatiquement. Pour la difficulté plus classique, le mode normal, il n’y a pas de bouton “estimation” et il faut cliquer soit même sur les personnages que l’on souhaite éliminer après qu’on ait eu la réponse à notre question au risque de se tromper. La difficulté étant choisie, il faut sélectionner un fichier JSON, en cliquant sur “choix du json”, une fenêtre s’ouvre dans vos dossiers pour vous permettre de choisir le fichier que vous souhaitez utiliser.



*Capture : Fenêtre sélection du fichier json.*

Après avoir sélectionné votre json, il est possible de cliquer sur le bouton “nouvelle partie” qui lancera alors une nouvelle partie avec la difficulté et la grille de personnage que vous aurez choisie. Vous pouvez également voir un bouton “charger partie” sur cette première fenêtre (capture 1), en effet, si vous avez fermé l’application en cour de jeu, il est possible de récupérer votre partie en relançant l’application, le bouton “charger partie” sera alors disponible et vous n’auriez qu’à cliquer dessus pour revenir là où vous en étiez.



Une fois qu’on est dans une partie, il faut poser des questions pour pouvoir éliminer des personnages et avancer dans le jeu. Pour cela, il y a toujours la phrase “le personnage est-il ou a t-il :”. Il faut alors compléter cette phrase en cliquant sur un petit cadre qui propose tous les attributs possibles comme cheveux, genre, prénom et autre. On choisit donc un attribut, et on sélectionne ensuite la valeur de cet attribut, par exemple après avoir sélectionné “cheveux”, on a le choix parmi “blond”, “roux” etc. Une fois la valeur choisie, la question est terminée et on peut cliquer sur “validé” pour que l’application renvoie la réponse à cette question. Cependant, certains attributs ont des valeurs binaires comme l’attribut “lunettes” ou “chapeau”, la question se termine alors directement après avoir sélectionné l’attribut.

Le personnage est-il ou a-t-il : cheveux noir

et le personnage est-il ou a-t-il : chapeau

ou le personnage est-il ou a-t-il : lunettes

**Ajouter question**

Il existe encore une autre fonctionnalité, quand on a terminé une question, nous ne sommes pas obligé de cliquer sur "valider", en effet, un cadre apparaît en dessous de notre question "ajouter question", si l'on clique dessus, on peut ajouter une autre question dans la même requête. On doit alors choisir comment connecter les deux questions, on peut sélectionner "et" ou "ou". On peut donc poser plusieurs questions en une requête et choisir si on les connecte. Il est possible de supprimer une de ces questions avant de cliquer sur "valider".

La réponse est : VRAI.  
Vos critères étaient : genre homme  
et chauve

**Valider**

Veuillez cliquer sur les personnages à éliminer.

Quand on valide une requête, l'application indique si la réponse est vraie ou fausse, et nous rappelle la question que l'on vient de poser. Elle nous demande ensuite de cliquer sur les personnages que l'on souhaite éliminer en conséquence, il faut encore cliquer sur "valider" cette fois pour éliminer définitivement les personnages sélectionnés. On peut alors recommencer le processus et soumettre une nouvelle requête.

La partie s'arrête toute seule et nous annonce la fin de la partie une fois qu'on a éliminé tous les personnages sauf celui qu'il fallait trouver, on a alors gagné la partie. Toutefois, si on choisit d'éliminer le personnage à trouver, la partie s'arrête également, cette fois-ci pour nous annoncer qu'on a éliminé la mauvaise personne et que nous avons donc perdu.

## 2.2 Fonctionnement interne

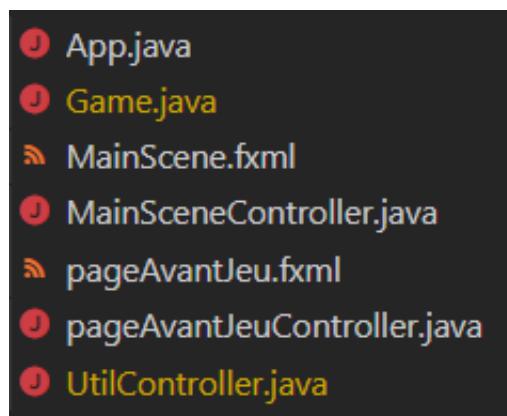
Tout d'abord pour fonctionner le jeu utilise un fichier d'extension JSON comme dans la partie 2.c. Le JSON est écrit d'une manière particulière afin d'être facilement utilisable et compatible avec le programme. Le fichier est écrit pratiquement comme celui présenté lors de

la présentation du projet, une seule différence, chaque personnage dispose d'une valeur "état" pour gérer l'élimination des différents personnages au cours de la partie (voir Capture : Fichier Json compatible).

```
{  
    "images": "../generateur/images/personnages",  
    "ligne": "4",  
    "colonne": "6",  
    "personnages": {  
        "0": {  
            "image": "samuel.png",  
            "prenom": "Samuel",  
            "genre": "homme",  
            "etat": "vivant",  
            "cheveux": "blanc",  
            "lunettes": "oui",  
            "chauve": "oui",  
            "moustache/barbe": "non",  
            "chapeau": "non"  
        },  
        "1": {  
            "image": "leon.png",  
            "prenom": "Leon",  
            "genre": "homme",  
            "etat": "vivant",  
            "cheveux": "blond",  
            "lunettes": "oui",  
            "chauve": "non",  
            "moustache/barbe": "non",  
            "chapeau": "non"  
        }  
    }  
},  
EXTRAIT
```

*Capture : Fichier json compatible.*

Nous allons maintenant découvrir le fonctionnement interne de ce "qui est-ce?" Dans un premier temps, nous allons voir la structure du programme de l'application. Elle se compose de 5 classes et de deux fichiers "fxml" comme on peut le voir sur l'image ci-dessous.



Nous allons commencer par décrire la Classe “App”. C'est la classe qui contient le main, quand le programme est lancé elle s'occupe uniquement de lancer

l'application avec comme interface graphique “pageAvantJeu.fxml”, de mettre un nom et une icône à la fenêtre.

Deuxièmement, nous avons la classe “pageAvantJeuController”. Cette classe est un "contrôleur", donc ,comme expliqué auparavant, est constitué des méthodes liées au boutons de l'interface (visible sur Capture1 : Page d'accueil de l'application). C'est dans cette classe que l'on retrouve notamment les méthodes pour choisir sa difficulté ou encore son Json pour la partie (voir Capture : Méthodes difficulté choixJson). Une fois que ces deux conditions sont remplis le bouton “nouvelle partie” se débloque, il appelle la fonction charger partie dans cette classe.

```
@FXML  
void choixdifficulte(ActionEvent event) {  
    difficulte = ((MenuItem) event.getSource()).getText();  
    estNouvellePartiePossible();  
    difficulteName.setText(difficulte);  
}  
  
@FXML  
void choixjson(ActionEvent event) {  
    FileChooser fc = new FileChooser();  
    FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("JSON files (*.json)", "*.json");  
    fc.getExtensionFilters().add(extFilter);  
    File selectedFile = fc.showOpenDialog(null);  
  
    if (selectedFile != null) {  
        jsonPath = selectedFile.getAbsolutePath();  
        String jsonNom = selectedFile.getName();  
        estNouvellePartiePossible();  
        jsonNameLabel.setText(jsonNom);  
    }  
}
```

*Capture : Méthode difficulté choixJson.*

Pour suivre, nous allons nous pencher sur la Classe “Game”. C'est cette classe qui contient les méthodes relatives aux règles et au bon fonctionnement d'une partie de “Qui est-ce ?”. Une partie prend en paramètre une difficulté (ici un string) et un Json compatible avec le jeu (voir Capture : Attribut de la classe Game), Json qui va permettre le choix du personnage par l'ordinateur et le déroulement du jeu.

```
private String difficulte;  
private JSONObject personnageChoisi;  
private ArrayList<JSONObject> listePersonnages;
```

*Capture : Attributs de la classe Game.*

“Game” est donc la classe du programme qui va gérer le Json pour le rendre jouable, par le biais de plusieurs méthodes comme par exemple “getListAttributs” qui retourne dans une liste tous les attributs des personnages du Json (de même pour les valeurs avec “getListValeurs”) (voir Capture : Méthode getListAttribut).

```
public ArrayList<String> getListAttributs() {
    ArrayList<String> attributs = new ArrayList<String>(personnageChoisi.keySet());
    attributs.remove(o: "image");
    attributs.remove(o: "etat");
    return attributs;
}
```

*Capture : getListAttributs.*

La classe vas aussi est à l'origine de la gestion des question posées par l'utilisateur pour y répondre en fonction des personnages du Json, on peut notamment citer la méthode “verifierReponse” qui vérifie si les attributs associées à leurs valeurs dans la question correspondent au personnage à deviner (voir Capture : Méthode verifierReponse).

```
public boolean verifierReponse(ArrayList<String> listeAttribut, ArrayList<String> listeValeurs,
    ArrayList<String> listConnecteurs) {
    boolean correspondPersonnage = personnageChoisi.get(listeAttribut.get(index: 0)).equals(listeValeurs.get(index: 0));
    for (int i = 1; i < listeAttribut.size(); i++) {
        if (listConnecteurs.get(i - 1).equals(anObject: "et")) {
            correspondPersonnage &= personnageChoisi.get(listeAttribut.get(i)).equals(listeValeurs.get(i));
        } else {
            correspondPersonnage |= personnageChoisi.get(listeAttribut.get(i)).equals(listeValeurs.get(i));
        }
    }
    return correspondPersonnage;
}
```

*Capture : Méthode verifierReponse.*

Enfin il est notable que cette classe régit la liste des personnages au cours la partie, s'ils sont en vie ou éliminés. Pour cela on édite le fichier Json afin de changer la valeur de “état” des personnages dans la méthode “tuerPersonnage” (voir Capture : Méthode tuerPersonnage).

```
public void tuerPersonnage(String nomPersonnage) {
    for (JSONObject personnage : listePersonnages) {
        if ((personnage.get(key: "image").toString()).equals(nomPersonnage)) {
            JSONObject personnageATuer = (JSONObject) personnage;
            personnageATuer.replace(key: "etat", value: "mort");
        }
    }
}
```

*Capture : Méthode tuerPersonnage.*

Dans un quatrième temps, nous allons regarder les classes “MainSceneController” et “UtilController” car elles fonctionnent ensemble. Ces deux classes contiennent les méthodes qui vont être utiles au déroulement de la partie de jeu côté interface graphique. Plus précisément c'est “MainSceneController” qui va utiliser les méthodes de l'autre classe. Regardons quelques-unes des méthodes les plus importantes.

Tout d'abord la fonction “lireJson” qui va lire le Json et récupérer des données tels que le chemins vers les images liées au personnages, les personnages en eux mêmes, et le nombre de ligne et de colonne de la future grille à l'aide de la fonction parse de la bibliothèque JSONObject (voir Capture : Méthode lireJson) :

```
protected void lireJson() {
    try {
        FileReader fr = new FileReader(json);
        JSONObject js = (JSONObject) new JSONParser().parse(fr);

        cheminVersImages = (String) js.get(key: "images");
        ligne = Integer.parseInt((String) js.get(key: "ligne"));
        colonne = Integer.parseInt((String) js.get(key: "colonne"));
        personnages = (JSONObject) js.get(key: "personnages");

        if ((String) js.get(key: "difficulte") != null) {
            // partie chargée
            difficulte = ((String) js.get(key: "difficulte"));
            partieEnCour = new Game(difficulte, personnages, ligne, colonne,
                (JSONObject) js.get(key: "personnagesChoisi"));
        } else {
            // nouvelle partie
            partieEnCour = new Game(difficulte, personnages, ligne,
                colonne);
        }
        fr.close();
    }
```

*Capture : Méthode lireJson.*

Ensuite la méthode “creerGrille” qui elle s'occupe de l'affichage graphique de la grille des personnages à l'endroit spécifié en paramètre. C'est aussi elle qui permet l'affichage de cible, quand l'on clique sur un personnage lors de l'élimination et l'affichage de la tête de mort si le personnage a été éliminé précédemment (voir Capture : Méthode creerGrille) .

Cette méthode consiste à créer une grille qui va contenir les différentes images puis pour chaque image contenue dans le fichier JSON de les ajouter à la grille. De plus, si le personnage actuellement ajouté est censé être un personnage éliminé, une tête de mort est placé sur lui.

Enfin nous allons revenir sur certains points abordés plus tôt, les différentes difficultés et la fonction de sauvegarde.

### 2.2.1 La difficulté “facile”

Le jeu propose donc deux difficultés, mais comment sont-elles implémentées. La difficulté choisie par le joueur est récupérée au choix de la difficulté sous forme d'un String dont nous avons parlé précédemment. Ce String est alors utilisé dans le constructeur de la classe “Game”. La difficulté normale ne change rien au jeu de base du “Qui est-ce ?”.

Nous allons ainsi nous pencher sur la difficulté facile. Cette dernière permet le déverrouillage lors de la partie du bouton “estimer” et sa fonction est, d'avant de valider une question, avoir une idée de combien de personnages sont concernés par la question et lors de la validation éliminée automatiquement en fonction de la réponse à la question. Cette estimation est définie par la méthode “estimerElimination”. Cette fonction permet de définir le nombre de personnages potentiellement éliminé par la question posé(en faisant appel à une fonction de la classe “game”) mais également de placer des cibles sur ce dernier pour permettre à l'utilisateur de facilement voir quels personnages vont être éliminé :

```
label estimationLabel = new Label("Elimination de "
    + listePersoAEliminer.size()
    + " sur " + ((partieEnCour.getNombrePersonnages() - listeTotalPersoElimine.size())));
```

*Capture : affichage du nombre de personnages a éliminé par rapport au nombre de personnages restant.*

```
File f = new File(pathname: "images/ciblepng.png");
Image imageCible = new Image(file:/// + f.getAbsolutePath());
ImageView imageViewCible = new ImageView(imageCible);
imageViewCible.setFitHeight(125);
imageViewCible.setFitWidth(90);
imageViewCible.setId("cible_" + idSplit[1] + "_" + idSplit[2] + "_" + idSplit[0]);
grillePerso.add(imageViewCible, Integer.parseInt(idSplit[1]),
    Integer.parseInt(idSplit[2]));
```

*Capture : affiche une cible sur les personnages potentiellement éliminés.*

### 2.2.2 Le système de sauvegarde

Comme requis, ce jeu dispose d'un système de sauvegarde qui fonctionne de la manière suivante : entre chaque élimination, le programme créer/modifier un fichier "save.json" et si l'utilisateur ferme la fenêtre en pleine partie, en rouvrant le jeu, il débloque le bouton "charger partie" (voir Capture1 : Page d'accueil de l'application). Bouton qui lui permet de reprendre la partie là où l'avait arrêté.

Pour créer le fichier de sauvegarde, nous utilisons la méthode "sauvegarderPartieEnCour" qui crée un JSONObject au format lisible par le jeu pour ensuite écrire dans un fichier le JSONObject transformé en String (voir Capture : Méthode sauvegarderPartieEnCour).

```
public void sauvegarderPartieEnCour(String images, int ligne, int colonne) {  
    // sauvegarde de la partie  
    JSONObject partieSave = new JSONObject();  
    partieSave.put("images", String.valueOf(images));  
    partieSave.put("ligne", String.valueOf(ligne));  
    partieSave.put("colonne", String.valueOf(colonne));  
    partieSave.put("difficulte", difficulte);  
    partieSave.put("personnagesChoisi", personnageChoisi);  
  
    JSONObject listePerso = new JSONObject();  
    int i = 0;  
    for (JSONObject personnage : listePersonnages) {  
        listePerso.put(String.valueOf(i), personnage);  
        i++;  
    }  
    partieSave.put("personnages", listePerso);  
  
    try (FileWriter file = new FileWriter(new File(pathname: "CestQuiGame/bin/sav  
        file.write(partieSave.toJSONString());  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

*Capture : Méthode sauvegardePartieEnCour.*

De plus, dans le JSON de sauvegarde, une variable supplémentaire est ajouté par rapport au JSON de base. Cette variable est le personnage sélectionné par l'ordinateur, en effet il est nécessaire de stocker le personnage qui a été choisi pour pouvoir reprendre la partie correctement.

### 3 Description du Générateur pour Grille de “Qui est-ce”

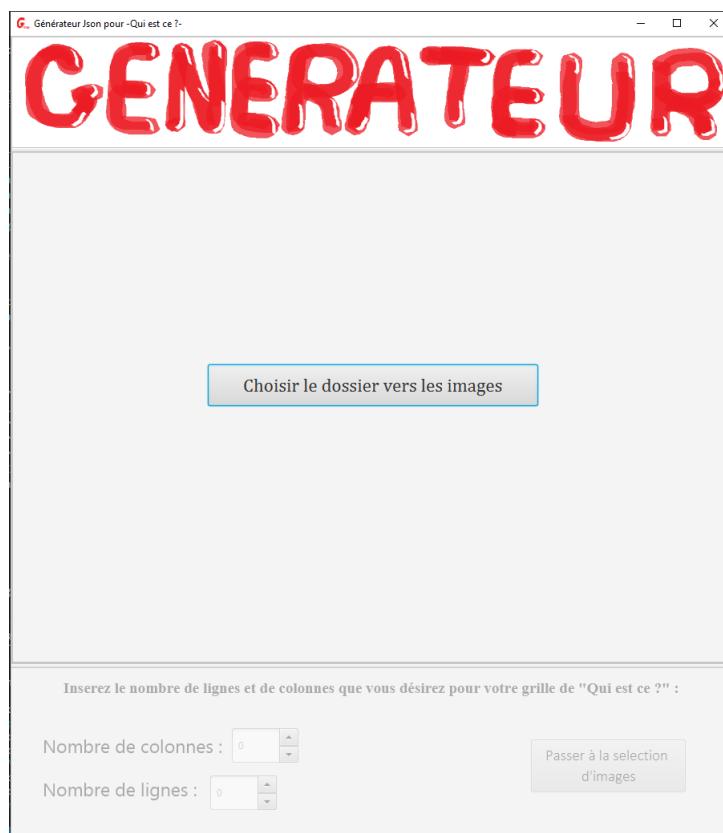
Le générateur pour grille de “Qui est-ce” est un outil qui offre la possibilité aux utilisateurs de créer leurs propres grilles de “Qui est-ce” avec les images de leur choix, les attributs et le nom qu’ils souhaitent. Cet outil consiste à réaliser une petite série d’étapes qui vous seront présentées ci-dessous. Notez que la grille de “Qui est-ce” une fois créée sera conservée au format JSON permettant, une fois l’application de “Qui est-ce” lancée, d’être sélectionnée lors de la demande de grille de jeu. De plus, le générateur est codé sur la même base que le “Qui est-ce” générique, c’est-à-dire qu’il utilise comme langage le Java et la bibliothèque javafx.

À présent, nous allons vous présenter dans un premier temps comment l’application fonctionne pour les utilisateurs et ensuite nous présenterons la conception de l’application d’un point de vue interne. Cette première partie explique quelles sont les interactions que peut effectuer l’utilisateur avec le générateur, sans traiter le côté programmation qui sera abordé ultérieurement. Tout au long de cette présentation de l’application, il sera également montré comment réaliser son propre fichier JSON tout simplement en réalisant les étapes présentées par ordre chronologique.

#### 3.1 Présentation du générateur et notice d'utilisation

Tout d’abord en exécutant le fichier App.java l’application se lancera et il sera alors affiché une première fenêtre d’accueil, point de départ dans la manipulation du générateur. Comme on peut le remarquer sur la capture d’écran ci-dessous, il sera impossible d’effectuer une autre action que “Choisir le dossier vers les images”. En effet avant toutes choses, nous avons préféré demander à l’utilisateur de sélectionner ces images car ce sont sur ces dernières que nous choisirons les attributs permettant aux utilisateurs de “Qui est-ce” de faire la différence entre deux personnages. De plus, cette étape est l’une des plus importantes et doit être forcément placée avant la validation des attributs, ce qui en fait par défaut notre première étape dans l’application.

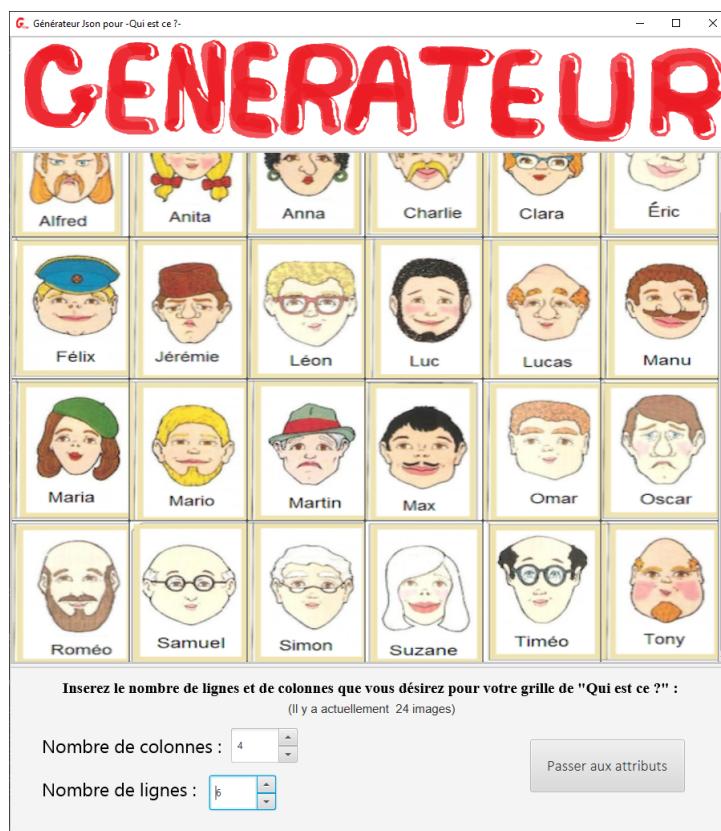
Ce générateur dispose également d’un système de sauvegarde lui permettant lors du lancement de l’application de reprendre à l’étape où nous nous trouvions la dernière fois que nous avions lancé le générateur.



*Capture : Page d'accueil du générateur.*

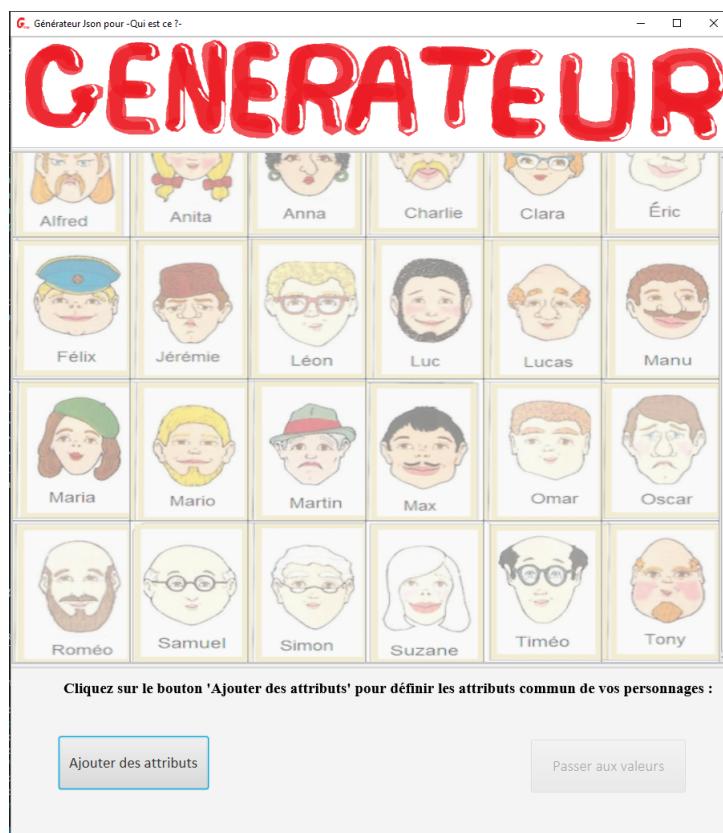
Ensuite lorsque l'utilisateur réalisera un clic sur “Choisir le dossier vers les images”, l'application ouvrira une fenêtre dans laquelle il suffira de sélectionner le dossier d'images sur lesquels vous souhaitez créer une grille de “Qui est-ce”.

Une fois le dossier image choisi, ce qui était impossible à réaliser à la page d'accueil du générateur deviendra accessible et vous pourrez alors choisir le nombres de lignes et de colonnes que vous souhaitez à condition que la multiplication du nombres de lignes et de colonnes n'excède pas le nombres d'image sélectionné précédemment.Toutefois, si le nombres de lignes et de colonnes choisies ne permettent pas de sélectionner toutes les images, Il sera demandé lorsque vous “Passer aux attributs” de choisir manuellement quelles images seront pris en charge par le générateur. A noter également qu'une fois sur cette fenêtre vous verrez en fond les images de votre dossier sans pouvoir encore interagir avec ces dernières (voir Capture : Fenêtre nombres de colonnes et lignes.). Enfin, lorsque le nombre de lignes et de colonnes sera adéquat avec le nombre d'images, vous pourrez alors choisir de passer à l'étape suivante en cliquant sur “Passer aux attributs”.

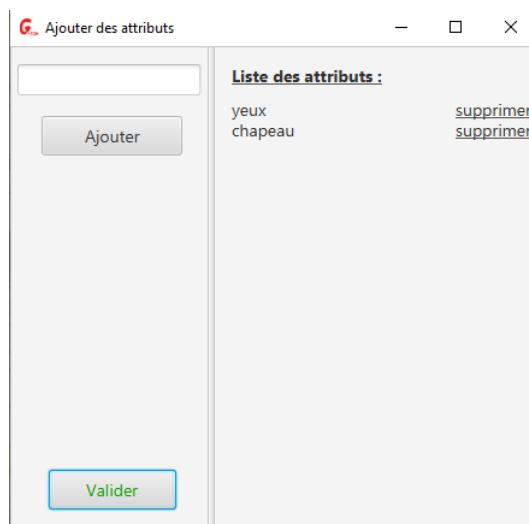


*Capture : Fenêtre nombre de colonnes et lignes.*

La prochaine fenêtre permet de saisir les attributs que l'utilisateur souhaite utiliser dans son "Qui est-ce". Pour ce faire, il suffit de cliquer sur "Ajouter des attributs" ( Capture : Fenêtre d'ajout des attributs.). Une fois fait, une nouvelle fenêtre s'ouvrira dans laquelle il suffit d'entrer le nom de l'attribut voulu, faire un clic sur "Ajouter". Il peut aussi les retirer en cliquant sur "Supprimer"(voir Capture : Fenêtre attributs.).

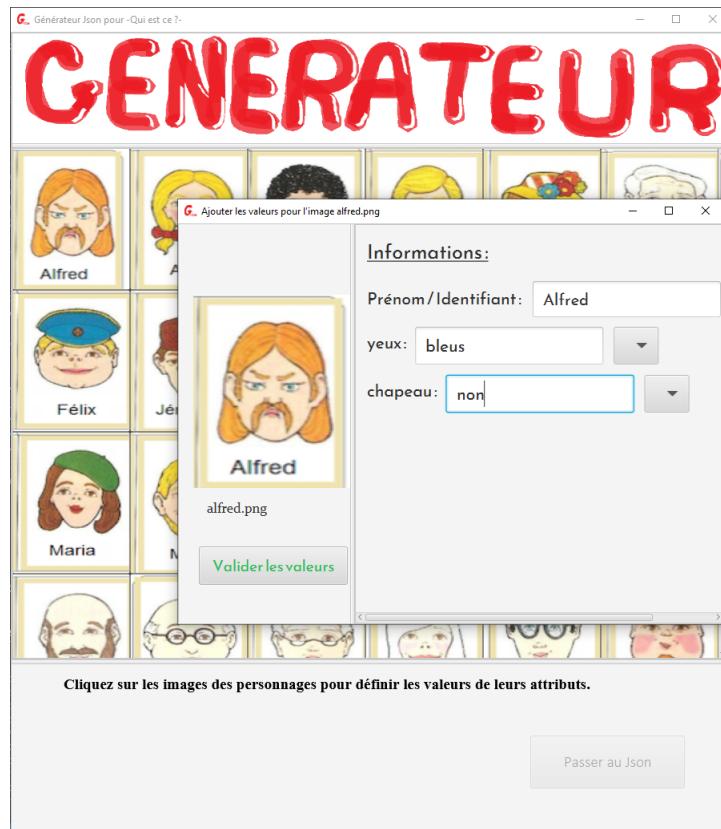


*Capture : Fenêtre d'ajout des attributs.*



*Capture : Fenêtre attributs.*

À présent, l'utilisateur dispose d'une grille de personnages et de ses attributs, il doit donc remplir pour chaque personnage les attributs qui lui correspondent. Pour ce faire, il suffit de cliquer sur une image pour accéder aux différents attributs du personnage (Capture : Attributs pour les personnages). Notez qu'il existe un système de suggestion qui propose à l'utilisateur de remplir des attributs avec des valeurs d'autres personnages saisies précédemment.



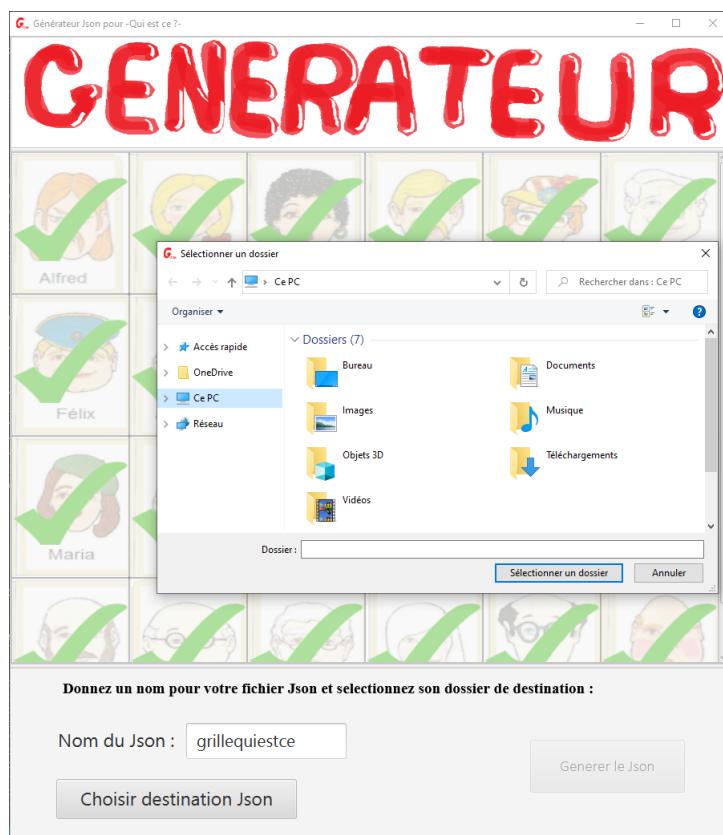
*Capture : Attributs pour les personnages*

Lorsque toutes les valeurs d'un personnage sont saisies et que l'utilisateur clique sur "Valider les valeurs" la case du personnage se valide. Pour passer à l'étape suivante, il faut que tous les personnages soient validés comme sur la capture ci-dessous.



*Capture : Tous les personnages ont leurs attributs remplis.*

Enfin, il reste une ultime étape avant de générer votre grille pour “Qui est-ce” en JSON. en effet, l’utilisateur doit simplement indiquer un nom et un emplacement pour son fichier JSON et cliquer sur “Générer Le Json” ( Capture : Générer le JSON.). Le fichier JSON obtenu sera donc compatible avec le “Qui est-ce” et sera stocké par l’utilisateur comme ci-dessous (Capture : Fichier JSON obtenu.).



*Capture : Générer le JSON.*

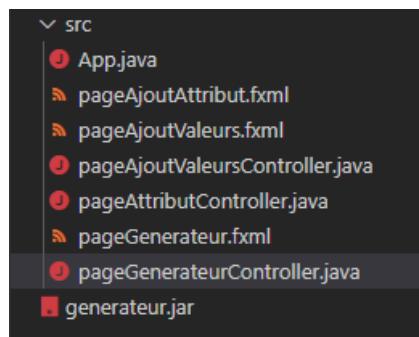
```
grillequistce - Bloc-notes
Fichier Edition Format Affichage Aide
[{"personnages": {"11": {"image": "manu.png", "chapeau": "non", "etat": "vivant", "prenom": "manu", "yeux": "marrons"}, "22": {"image": "timeo.png", "chapeau": "non", "etat": "vivant", "prenom": "timeo", "yeux": "bleus"}, "0": {"image": "alfred.png", "chapeau": "non", "etat": "vivant", "prenom": "Alfred", "yeux": "bleus"}, "1": {"image": "anita.png", "chapeau": "non", "etat": "vivant", "prenom": "anita", "yeux": "marrons"}, "21": {"image": "suzane.png", "chapeau": "non", "etat": "vivant", "prenom": "suzane", "yeux": "marrons"}, "2": {"image": "lucas.png", "chapeau": "non", "etat": "vivant", "prenom": "lucas", "yeux": "marrons"}, "10": {"image": "julien.png", "chapeau": "non", "etat": "vivant", "prenom": "julien", "yeux": "marrons"}, "19": {"image": "antoine.png", "chapeau": "non", "etat": "vivant", "prenom": "antoine", "yeux": "marrons"}, "3": {"image": "amelie.png", "chapeau": "non", "etat": "vivant", "prenom": "amelie", "yeux": "marrons"}, "18": {"image": "camille.png", "chapeau": "non", "etat": "vivant", "prenom": "camille", "yeux": "marrons"}, "4": {"image": "elodie.png", "chapeau": "non", "etat": "vivant", "prenom": "elodie", "yeux": "marrons"}, "17": {"image": "louise.png", "chapeau": "non", "etat": "vivant", "prenom": "louise", "yeux": "marrons"}, "5": {"image": "marielle.png", "chapeau": "non", "etat": "vivant", "prenom": "marielle", "yeux": "marrons"}, "16": {"image": "lucie.png", "chapeau": "non", "etat": "vivant", "prenom": "lucie", "yeux": "marrons"}, "6": {"image": "mirella.png", "chapeau": "non", "etat": "vivant", "prenom": "mirella", "yeux": "marrons"}, "15": {"image": "louise2.png", "chapeau": "non", "etat": "vivant", "prenom": "louise2", "yeux": "marrons"}, "7": {"image": "mirella2.png", "chapeau": "non", "etat": "vivant", "prenom": "mirella2", "yeux": "marrons"}, "14": {"image": "louise3.png", "chapeau": "non", "etat": "vivant", "prenom": "louise3", "yeux": "marrons"}, "8": {"image": "mirella3.png", "chapeau": "non", "etat": "vivant", "prenom": "mirella3", "yeux": "marrons"}, "13": {"image": "louise4.png", "chapeau": "non", "etat": "vivant", "prenom": "louise4", "yeux": "marrons"}, "9": {"image": "mirella4.png", "chapeau": "non", "etat": "vivant", "prenom": "mirella4", "yeux": "marrons"}, "12": {"image": "louise5.png", "chapeau": "non", "etat": "vivant", "prenom": "louise5", "yeux": "marrons"}, "11": {"image": "mirella5.png", "chapeau": "non", "etat": "vivant", "prenom": "mirella5", "yeux": "marrons"}]
```

*Capture : fichier JSON obtenu.*

## 3.2 Description technique du générateur

Dans cette partie, nous nous concentrerons sur la partie technique du générateur. Nous verrons comment fonctionne l'application et comment les actions de l'utilisateur se réalisent. Pour bien aborder cette notion, nous proposons en premier lieu d'expliquer la structure de ce projet, comprendre les différentes classes et leurs utilités. Ensuite, nous verrons plus précisément les fonctionnalités les plus importantes, à savoir choisir les images, définir les attributs, donner les valeurs des attributs à chaque personnage et générer le JSON et de bien comprendre comment elles fonctionnent. Enfin, nous reviendrons sur quelques fonctions essentielles du programme comme la sauvegarde, la vérification des attributs et la suggestion des valeurs d'attributs.

Commençons par la structure du programme. Comme montré sur la capture ci-dessous ( Capture : Les classes du générateur.), le générateur dispose de 4 classes.



*Capture : Les classes du générateur.*

La première nommée “App.java” permet de lancer l'application, c'est l'exécution de cette dernière qui ouvre le générateur. Son code est plutôt facile à comprendre puisqu'elle ne fait que charger la “pageGenerateur.fxml” et l'icône ( Capture : App.java.) grâce à la fonction FXMLLoader.load.

```
root = FXMLLoader.load(getClass().getResource("pageGenerateur.fxml"));
```

*Capture : App.java.*

Une fois la page du générateur chargée, cette classe n'effectue plus aucune action et laisse la main à une autre classe, “pageGenerateurController.java”.

La deuxième classe que nous vous présentons s'appelle "pageGenerateurController". C'est dans cette dernière que seront présentes toutes les fonctions permettant les interactions avec l'utilisateur, à l'exception de celles pour les ajouts d'attributs et de valeurs. On y retrouve notamment les fonctions associées au bouton pour passer d'une étape à une autre ou des fonctions plus spécifiques comme celle de sauvegarde ou celle pour générer le json qui seront abordées ultérieurement ou encore celle qui permet de lancer les fenêtres d'ajout d'attributs ou d'ajout de valeurs. Cela en fait donc la classe la plus importante du générateur, mais aussi la plus complète car elle contient plusieurs fonctions avec des utilités très différentes les unes des autres allant parfois de simples Getter et Setter au chargement de fenêtre fxml.

En troisième classe nous retrouvons "pageAttributController.java". Cette classe intervient lorsque l'utilisateur saisit les attributs. En effet, lorsque l'utilisateur clique comme vu lors de la présentation du générateur sur "Ajouter attribut" une nouvelle fenêtre s'ouvre dans laquelle vous pouvez saisir un nom pour un attribut ou le supprimer. Cette classe est celle qui gère les interactions entre l'utilisateur et cette nouvelle fenêtre elle est composée exclusivement des méthodes réalisant ces interactions.

Enfin La quatrième classe se nomme "pageAjoutValeursController.java". A l'instar de la classe "pageAttributController.java", cette classe s'occupera de toutes les interactions entre l'utilisateur et la fenêtre qui s'ouvre lorsque ce dernier cliquera sur les images dont il souhaitera entrez les valeurs pour les attributs précédemment choisis.

Voyons à présent plus précisément les 4 fonctions les plus importantes du générateur. En premier lieu nous trouvons la fonction permettant de choisir les images. Cette fonction intervient au début de l'application lorsque l'utilisateur doit choisir un chemin pour récupérer un dossier d'image. Son code est donné sur les Capture : Fonction choisir image 1. Cette fonction est l'une des plus importantes du générateur car elle constitue le premier point clé dans la génération du JSON.

```
DirectoryChooser directoryChooser = new DirectoryChooser();
selectedDirectory = directoryChooser.showDialog(MainAnchorPane.getScene().getWindow());
```

*Capture : Fonction choisir image 1.*

La première étape de la fonction consiste à récupérer un dossier d'image grâce à la fonction DirectoryChooser.

```

if (selectedDirectory.isDirectory()) {
    int x = 0; // colonne
    int y = 0; // lignes
    grillePerso = new GridPane();
    grillePerso.setId("grillePerso");
    grillePerso.setGridLinesVisible(true);
    grillePerso.setMaxSize(900, 10000);
    grillePerso.setHgap(2);
    grillePerso.setVgap(2);

```

Ensuite nous créons une grille dans laquelle vont venir se loger les images. Et, à l'aide d'une boucle qui parcourt tous les fichiers du dossier, les images sont ajoutées à la grille, avec la fonction add, en fonction de leur format (png et jpg) triée grâce à la fonction ImageFiltre.

```

for (File image : selectedDirectory.listFiles(imageFiltre)) {
    String nomImage = image.getName();
    String urlImage = image.getAbsolutePath();
    Image imagePerso = new Image("file://" + urlImage);
    ImageView imageViewPerso = new ImageView(imagePerso);
    listeImages.add(imageViewPerso);
    imageViewPerso.setFitHeight(175);
    imageViewPerso.setFitWidth(145);
    imageViewPerso
        .setId(nomImage + "*" + x + "*" + y + "*" + urlImage);
    imageViewPerso.setOnMouseClicked(selectionnerPersonnageEvent);
    grillePerso.add(imageViewPerso, x, y);
    compteurImage++;
}

```

Une autre fonction importante dans le générateur est celle de l'ajout des attributs. Cette fonction joue un rôle clé pour le “Qui est-ce” puisque les questions des joueurs se basent sur le principe des attributs et des valeurs associées à chaque image. Comme on peut le remarquer sur la Capture 14 : Fonction ajout attribut., une fois créé un attribut sera placé dans une liste dans laquelle il pourra ensuite être supprimé et dans laquelle il pourra apparaître sur la fenêtre.

```

// ajout des suppr
Label suppr = new Label("supprimer");
listSuprLabel.add(suppr);
suppr.setUnderline(true);
suppr.setOnMouseClicked(supprimerAttribut);
suppr.setId("suppr" + numeroID);
anchorPaneAttributs.getChildren().add(suppr);

```

De plus, une fois créé, il est ajouté à une autre liste appelée “ListAttributsString” qui sera la liste de tous nos attributs et qui sera transmise au JSON lors de la validation des attributs par la fonction “submit”.

```
// ajout des attributs
listAttributsString.add(champDeTexte.getText());
Label attribut = new Label(champDeTexte.getText());
listAttributsLabel.add(attribut);
anchorPaneAttributs.getChildren().add(attribut);
```

*Capture : Fonction ajout attribut.*

Au même titre que la fonction précédente, les fonctions permettant de saisir la valeur des attributs pour chaque personnage sont importantes. On y trouve de quoi récupérer les informations passées par l'utilisateur dans les zones de textes.

```
((TextField)anchorPaneId.getScene().lookup("#field"+source)).setText(((MenuItem)event.getSource()).getText());
```

Mais aussi de vérifier que toutes les valeurs correspondantes aux attributs soient bien remplies par l'utilisateur ( Capture : Fonctions ajouts valeurs.) avant de transmettre le JSON correspondant à la page principale via la fonction “validerButtonEvent”.

```
private void verifierAttributsTousRemplis() {
    boolean estComplet = true;
    for (int i = 0; i < listeAttributs.size(); i++) {
        TextField textField = (TextField) informationsPaneId.getScene().lookup("#field" + i);
        if (textField.getText().equals("")) {
            estComplet = false;
            break;
        }
    }
}
```

*Capture : Fonctions ajouts valeurs.*

Enfin, nous vous proposons la fonction pour générer le JSON, “genererJsonEvent”. Cette fonction va à la fois charger la page finale et générer le fichier JSON final. La Capture : Fonction générant le JSON. montre le code permettant de charger la page de fin du générateur, on y voit plusieurs actions comme le remplacement de plusieurs éléments fxml par un bouton final “fermefenetre”. Dans un second temps, la fonction va se charger de créer le JSON correspondant avec le code de la Capture : code pour générer le JSON. où l'on retrouve les éléments permettant de créer le JSON des lignes 753 à 770 et enfin une partie nous signalant si notre fichier JSON s'est bien généré ou non.

```
// supprime tous les éléments
Label textLabel = (Label) bottomAnchorPane.getScene().lookup("#nomJsonLabel");
TextField textField = (TextField) bottomAnchorPane.getScene().lookup("#nomJsonField");
Button buttonChoixDestination = (Button) bottomAnchorPane.getScene()
    .lookup("#choixDestinationButton");
```

*Capture : Fonction générant le JSON.*

```

JSONObject jsonFinal = new JSONObject();
jsonFinal.put("images", String.valueOf(cheminVersImage));
jsonFinal.put("ligne", String.valueOf(ligne));
jsonFinal.put("colonne", String.valueOf(colonne));

JSONObject listePerso = new JSONObject();
int i = 0;
for (JSONObject personnage : listePersonnages) {
    listePerso.put(String.valueOf(i), personnage);
    i++;
}
jsonFinal.put("personnages", listePerso);

```

*Capture : code pour générer le JSON.*

Pour la dernière partie concernant le générateur, nous parlerons de quelques fonctions singulières mais toutefois très utiles, à commencer par la vérification des valeurs des attributs qui permet d'éviter qu'un utilisateur ne transmette 2 personnages ayant les mêmes valeurs d'attributs auquel cas il serait impossible de les différencier. Cette fonction s'appuie sur la comparaison des valeurs des personnages comme le montre le code ci-dessous.

```

for (JSONObject personnage2 : listePersonnages) {
    if (!personnage.equals(personnage2)) {
        if (p.equals(personnage2.get("prenom"))) {
            for (String attribut : attributs) {
                if (personnage.get(attribut).equals(personnage2.get(attribut))) {
                    return false;
                }
            }
        }
    }
}

```

*Capture : Fonction vérification.*

Il existe également une fonction de sauvegarde de partie en cours, celle-ci permet en cas d'interruption du programme de reprendre où nous en étions grâce à un système de sauvegarde. Ce dernier est simplement constitué d'un fichier JSON qui est remplie étape après étape lors de chaque action de l'utilisateur :

```

public static void sauvegarderPartieEnCour() {
    // sauvegarde du générateur
    JSONObject generateurSave = new JSONObject();
    if (etape >= 1) {
        generateurSave.put("cheminVersImage", String.valueOf(cheminVersImage));
        generateurSave.put("etape", 1);
    }
}

```

*Capture : Fonction sauvegarde de partie 1.*

Puis est enregistré dans un endroit prédéfini à l'aide d'un File Writer :

```

854
855     try (FileWriter file = new FileWriter(new File("bin/save.json"))) {
856         file.write(generateurSave.toJSONString());

```

*Capture : Fonction sauvegarde de partie 2.*

Pour conclure avec le générateur, nous retrouvons une fonction permettant la suggestion des valeurs pour les attributs lors du remplissage pour chaque personnage. Ce code crée un élément “menuitem” pour chaque attribut valider précédemment dans lequel vont venir se stocker les valeurs déjà saisies par l’utilisateur à ces mêmes attributs.

```
110 |         if((valeursDejaDonneesMap.containsKey(attribut))){
111 |             menu.setOpacity(1);
112 |             for (String valeurs:  valeursDejaDonneesMap.get(attribut)) {
113 |                 MenuItem item = new MenuItem();
114 |                 item.setId(""+i);
115 |                 item.setText(valeurs);
116 |                 item.setOnAction(setValeurOnTextField);
117 |                 menu.getItems().add(item);
118 |             }
119 |         }
```

*Capture : Code pour la suggestion des valeurs.*

Cette fonction permet à l’utilisateur d’éviter de ressaisir des valeurs si elles sont récurrentes chez des personnages.

Pour conclure sur le générateur, nous sommes satisfaits du résultat, mais également des différentes fonctionnalités qu’il possède permettant de créer un JSON personnalité assez simplement pour n’importe quel utilisateur.

## 4 Développement des Extensions “Qui est-ce”

Pour l'extension de notre jeu, nous avons pris la décision d'ajouter un mode “multijoueur”. Cette extension réside sur la même base que celle du jeu de base, mais cette fois le joueur n'interagit plus avec un problème, mais avec un autre joueur connecté en celui-ci en réseau. Nous avons fait le choix de cette extension, car elle nous permet de réutiliser de nombreuses compétences vu en cours tel que le principe de “socket client-serveur” ou encore l'utilisation de “thread” mais également parce qu'elle permet de rendre le jeu beaucoup plus ludique avec des interactions entre deux utilisateurs.

### 4.1 Présentation du mode multijoueur

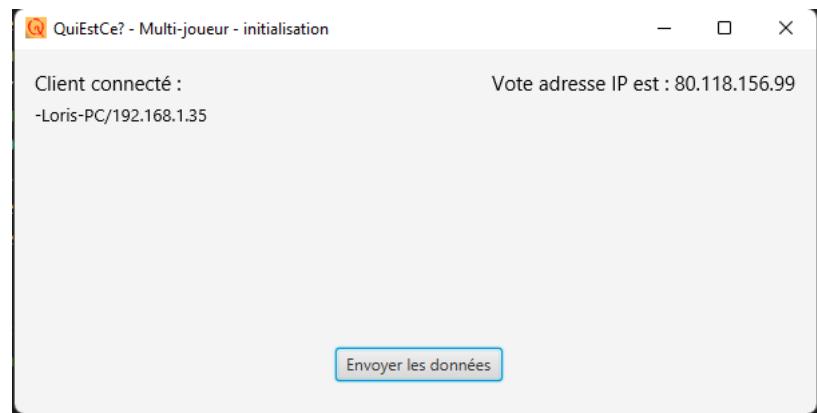
Dans cette première partie, nous allons présenter le fonctionnement de cette extension et expliquer quelles sont les interactions entre les utilisateurs. Premièrement, lorsque l'utilisateur décide de jouer en multijoueur, il doit choisir d'être “serveur” ou “client”, c'est-à-dire il doit faire le choix entre héberger la partie ou simplement la rejoindre.



*Capture : Initialisation mode multijoueur*

L'utilisateur qui se place en tant que “serveur” doit posséder le dossier avec le JSON, les images, mais de plus doit avoir son port 51537 d'ouvert sur son réseau public pour permettre au “client” de se connecter.

Le serveur doit alors choisir le JSON avec lequel la partie va se dérouler et le client doit renseigner l'IP publique du serveur pour se connecter.



*Capture : Client connecté*

Une fois connecté, le serveur peut envoyer toutes les données nécessaires pour jouer au client (le JSON et toutes les images). Finalement, une fois toutes les données transférées, la partie peut commencer.

Dans un premier temps, chaque joueur doit choisir l'image qui souhaite faire deviner à son adversaire :



### *Capture : choix personnage*

Après que les deux adversaires ont choisi leurs personnages, une boucle se met en place pour que, à tour de rôle, les adversaires :

- posent une question
- attendent la réponse
- éliminent le ou les personnages qui corresponde(ent) ou ne corresponde(ent) pas à leur question
- s'ils n'ont ni gagné ni perdu, attendent la question de l'adversaire répondent à la question
- attendent de savoir si l'adversaire a gagné ou perdu, sinon la boucle recommence jusqu'à avoir un gagnant ou un perdant.

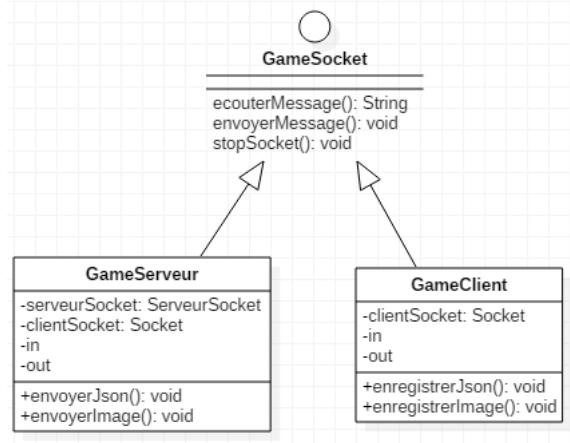


### *Capture : partie gagnée*

## 4.2 Explication du fonctionnement de l'extension

Dans cette ultime partie, nous allons présenter les grands axes de fonctionnement de l'extension, et également décrire quel sont les changements apportés à la structure du projet. En effet, le fonctionnement du jeu en lui-même est assez similaire au fonctionnement du qui est-ce présenté précédemment. Effectivement, les classes utilisées pour le fonctionnement de cette extension héritent des classes utilisées par le jeu. De cette façon, nous avons réutilisé la plupart des fonctions déjà préalablement créées sans devoir redéfinir chacune d'entre-elle. C'est pour cela que nous allons présenter les axes important sans présenter le fonctionnement de base de l'extension.

L'axe principal de l'extension réside dans l'utilisation d'un système client serveur qui utilise des sockets. Un socket est un "connecteur réseau" basé sur le protocole TCP. Il permet d'établir une connexion entre deux ou plusieurs machines pour permettre un échange de données. Pour permettre la création des différents sockets, nous avons ajouté deux classes qui implémentent une même interface :



Ces deux classes utilisent donc des fonctions similaires telles que la fonction d'envois de message, d'écoute de message ou encore une fonction pour fermer les sockets en cas d'arrêt de partie :

```

public interface GameSocket {
    String ecouterMessage() throws IOException;

    void envoyerMessage(String msg) throws IOException;

    void stopSocket() throws IOException;
}
  
```

Mais elles diffèrent dans leur initialisation, en effet le serveur doit être créé en premier sur un port spécifique (en l'occurrence le port : 51537) :

```

serveurSocket = new ServerSocket(port);
  
```

Et dans un second temps, le client est créé, avec l'IP du serveur et sur le même port :

```

clientSocket = new Socket(ip, 51537);
  
```

et enfin le client se connecte au serveur avec la fonction accept :

```

clientSocket = serveurSocket.accept();
  
```

À partir de cet instant, la connexion entre le client et le serveur est établie. Pour permettre les échanges, il faut mettre en place des streams (input et output) pour chacune des sockets, dans chacune des classes. De ce fait, ils pourront communiquer en passant directement pour ces inputs :

```
in = new DataInputStream(clientSocket.getInputStream());
out = new DataOutputStream(clientSocket.getOutputStream());
```

*Capture : création des streams*

Un autre axe de travail de cette extension en lien avec les sockets est l'envoi et la réception des données nécessaire à la partie (JSON et images). En effet, un client qui rejoint une partie ne possède pas le JSON ni les images qui sont utilisées tout au long de la partie par le programme. De ce fait, il faut, avant le lancement de la partie, envoyer tous ces fichiers.

Pour cela, nous avons créé deux paires de fonctions, une pour l'envoi du JSON et une pour l'envoi des images. Nous avons différencié ces fonctions, car leur fonctionnement est différent sur de nombreux points.

## 1. Fonctionnement de l'envoi du JSON

Pour l'envoi, donc côté serveur, on crée dans un premier temps un tableau de byte (1 byte est 8 bit consécutifs dans la mémoire de l'ordinateur) utilisé pour stocker le JSON, puis on l'envoie au serveur via la socket.

```
byte[] mybytearray = new byte[(int) file.length()];
BufferedInputStream bis = new BufferedInputStream(new FileInputStream(file));
bis.read(mybytearray, 0, mybytearray.length);
OutputStream os = clientSocket.getOutputStream();
```

Côté réception, on répète le même processus, mais dans le sens inverse. On crée un tableau de byte d'une taille assez conséquente pour pouvoir stocker le JSON, puis on l'enregistre dans un endroit prédéfini sur la machine pour qu'il puisse être lu plus tard par le programme.

```
byte[] mybytearray = new byte[1000000];
FileOutputStream fos = new FileOutputStream("CestQuiGame/bin/gameTemp/game.json");
BufferedOutputStream bos = new BufferedOutputStream(fos);
int bytesRead = in.read(mybytearray, 0, mybytearray.length);
bos.write(mybytearray, 0, bytesRead);
bos.close();
```

**2. Fonctionnement de l'envoi d'image** Pour les images, nous avions dans un premier temps utilisé le même principe que pour les JSON, mais malheureusement nous avons heurté un problème lorsque la taille de l'image devenait trop importante. Nous avons ainsi décidé de changer de méthode en optant pour un envoi de l'image morceau par morceau.

De plus, pour pouvoir stocker le nom de l'image ainsi que son contenu nous avons utilisé un système de JSON. En effet, dans un premier temps nous transformons l'image en un tableau de byte (compréhensible par la machine), que nous “encodons” en string pour pouvoir l'envoyer et finalement nous créons le JSON avec nos deux attributs :

```
FileInputStream imageInFile = new FileInputStream(image);
byte imageData[] = new byte[(int) image.length()];
imageInFile.read(imageData);

String imageDataString = encodeImage(imageData);
imageInFile.close();

JSONObject obj = new JSONObject();

obj.put("nomFichier", image.getName());
obj.put("image", imageDataString);

String jsonEncode = obj.toJSONString();
```

Dans un second temps, nous découpons l'image en plusieurs parties de 64000 bits (taille maximum autorisée pour envoyer un string) et nous envoyons chacun de ses morceaux un par un au client :

```
int tailleMax = 64000;

String[] chunks = jsonEncode.split("(?=<=\\G.{ " + tailleMax + " })");
for (String string : chunks) {
    out.writeUTF(string);
}
out.writeUTF("endImage");
```

Côté client, le processus inverse est réalisé.

Finalement, une des problématiques majeure de notre système client serveur est l'attente. En effet, les fonctions de connexion ou d'écoute des sockets fonctionne à l'aide boucle créant un blocage du programme au moment de leur appel. Pour palier à ce problème et éviter de faire planter notre partie, nous devons utiliser des “thread”. Un thread est une unité d'exécution qui fonctionne en parallèle du programme principal. Il permet de faire appel à des fonctions qui entraîne des blocages sans arrêter le processus principal.

```
threadServeur = new Thread(() -> {
    while (true) {
        try {
            String ipClient = ((GameServer) gameSocket).connexionClient();
            ipText1.setText("Client connecté :");
        }
    }
})
```

*Capture : utilisation de thread pour attendre la connexion d'un client.*

Le fonctionnement d'un thread est assez simple, il suffit de créer un thread contenant le code que l'on désire exécuter en parallèle est d'appeler la fonction start().

Les threads sont aussi très utilisés pour détecter la déconnexion du client ou du serveur. En effet, au moment de l'attente d'une interaction utilisateur, il peut arriver que le client ou le serveur se déconnecte pour quelconques raisons. Il faut donc à ce moment-là le signaler à l'utilisateur et arrêter le programme.

Finalement, la création de l'extension fut un réel défi à relever, mais après de nombreux essais et modification, nous avons réussis à faire fonctionner le mode multijoueur sans problème.

## **5 Conclusion et recul sur le travail effectué**

Pour conclure, nous sommes satisfaits du résultat, nous avons atteint nos objectifs. Nous avons choisi d'utiliser le langage java et ainsi, nous avons pu réutiliser ce que nous avions appris en cours de modélisation et programmation par objets, mais nous avons également développé des compétences personnelles, notamment concernant l'utilisation de javafx et la création d'interface. C'était une bonne expérience et cela nous a aussi permis de nous exercer à réaliser un travail en équipe. Nous avons rencontré quelques difficultés , mais finalement tout s'est bien terminé.