Qui est-ce ?
Projet de programmation

Groupe *O*

Matteo Faura Behague, Rendra Djunaedi, Alaa Qach, Syrine Mansour

https://gitlab.etu.umontpellier.fr/groupe-o/qui-est-ce

L2 informatique
Faculté des Sciences
Université de Montpellier.

16 avril 2022

Résumé

Ce rapport relate la conception d'une petite application informatique du jeu ***Qui est-ce****. Cette application contient ainsi le jeu de base permettant à l'utilisateur de pouvoir faire une partie et un générateur permettant à ce dernier de créer sa propre grille avec les personnages et caractéristiques qui lui convient. Ce projet sera par la suite presenté lors de la soutenance du Vendredi 22 Avril 2022*

Sommaire

1 Technologies utilisées et organisation

Dans cette section nous parlerons du langage utilisé, des librairies utilisé, des logiciels qui nous ont permis de réaliser notre application mais aussi de l'organisation au sein du groupe.

1.1 Choix du langage

1.1.1 Java Le langage que nous avons décidé d'utiliser :

Nous avons décidé de réaliser notre projet en Java. Etant donné qu'un cours sur la programmation orienté objets a eu lieu au semestre dernier, il nous semblait interessant de mettre en application nos connaissances fraichement acquises ; de plus le Java est un langage que chacun d'entre nous affectionne. Notre projet est réalisé pour une version Java 11 minimum compte tenue des librairies que nous avons décidé d'utiliser qui ne fonctionnent que sur cette version ou les versions supérieur de Java.

Vous pouvez télécharger Java 11 ici : https ://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html ?msclkid=c4265d93bbc911eca6b7eb7624309527 pour pouvoir lancer le logiciel sur les machines et systèmes d'exploitation où il est disponible.

1.2 Choix des bibliothèques

1.2.1 JavaFX une bibiothèque de gestion d'interface graphique :

La bibliothèque qui est utilisé pour gérer les interfaces graphiques est JavaFX version 17.0.2. Nous avons fait ce choix car c'est une librairie récente, mise à jour fréquemment et au goût du jour. De plus, il existe un outil de création et de gestion d'interface graphique suffisament puissant pour aider à gerer des jeux de plateform tel que **Qui est-ce ?**, **UNO** ou des applications tel que des logiciels pour les caisses de supermarchés, ce logiciel nommé SceneBuilder a donc pu nous être d'une grande utilité pour construire les différents onlgets présent dans notre application. Ce logiciel autorise aussi une gestion de la partie **design** à l'aide d'un système équivalent au CSS, certe moins puissant que celui utilisé pour une page internet mais qui reste un bon atout.
Pour finir, cette librairie est très connus, ce qui implique des réponses à des problèmes que d'autre auront déjà rencontré et possède un site complet avec une bonne documentation. Pour le petit apartè, la version java du jeu bien connue Minecraft est conçue à l'aide de JavaFX. Vous pouvez trouver ici cette librairie pour les versions windows, linux ou macOS :
https ://openjfx.io/ ?msclkid=eeded834bbce11ecb8230284675b6d78

1.2.2 Json simple pour gèrer les fichiers json :

Pour gérer les fichiers JSON la bibliothèque utilisé est Json Simple version 1.1.1. Comme l'indique son nom, cette librairie est simple d'utilisation avec notamment peu de classes, un simple analyser de json et quelques methodes seulement. Vous pouvez trouver Json Simple version 1.1.1 ici :
http ://www.java2s.com/Code/Jar/j/Downloadjsonsimple11jar.htm ?msclkid=38b9dfd7bbcf11ec9fbef3a3aa910602

1.3 Organisation du travail

1.3.1 Répartition du travail au sein du groupe

La répartition des tâches et du travail a été faite avant le commencement du projet. Aprés avoir réalisé différents UML, nous avons remarqué que le groupe avait pour chacun d'entre nous une vision différente de l'application, c'est pourquoi nous avons regroupé nos idées les plus pertinentes pour en faire qu'un seul UML.Voici comment nous avons fini par répartir les tâches :

- Matteo : Organisation de la structure du jeu et tout ce qui touche au back-end de l'application.
- Rendra : GUI de l'application et front-end avec implémentation des methodes back-end.
- Alaa : l'aspect esthétique (CSS).
- Syrine : extention.

Mais ce n'est pas une répartition fixe, on s'entraidait et on alternait lorqu'une personne bloquait pour pouvoir d'avantage avancer sur le projet.

1.3.2 Rythme de travail, mode de fonctionnement ...

En ce qui concerne le Rythme de travail et le mode de fonctionnement, on se regroupe évidemment chaque lundi pour profiter de la présence physique du groupe, nous permettant de mieux communiquer les changements effectué sur l'application, faire le point, mais aussi de se motiver. Hors lundi, on travaille chez-soi le soir en salon de reunion vocal sur discord, mais aussi à la bibliothéque universitaire (BU) les jeudi aprés-midi dans une salle de projet pour encore une fois profiter du fait d'être en groupe.

2 Étape 1 : Permettre à l'utilisateur de jouer

*T*out d'abord il faut mettre au clair quelques points :

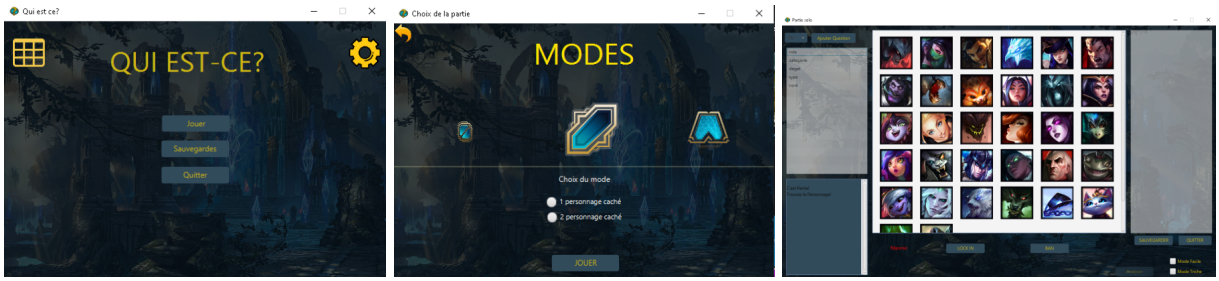
Nous parlerons très souvent d'attributs et arguments. Nous avons décidé d'utiliser ces termes pour parler des caracteristiques d'un personnage, les attributs étant les points ou critères sur lequels on compare les entités et les arguments sont les valeurs de ces critères et qui font la différences entre chaque entité, par exemple :
Pierro parle le français et l'espagnol et porte un chapeau, et Lisa parle l'anglais et ne porte pas de chapeau.
Alors on dira que cette ensemble de personnages a pour liste d'attribut : le nom, les languages qu'ils connaissent et le port d'un chapeau ; il existera donc pour chaque attribut plusieurs arguments : pour le nom nous aurons Pierro et Lisa, pour les languages parlé nous aurons le français l'espagnol et l'anglais, et nous avons le port du chapeau ou pas. Ainsi, nous parlerons de paires d'attributs et arguments pour chaque personnage.

2.1 Fonctionnalités de l'application : interactions possibles de l'utilisateur

2.1.1 Information sur l'installation et les compatibilités

L'application est fonctionnel sous Windows 10 et les machines de la faculté et normalement toutes machines possédant une version 11 de Java superieur. Il suffit de se rendre sur le lien gitlab enoncé plus haut et de télécharger le code source de la branche **main** au format qui vous convient le mieux puis de suivre les instructions du README.txt qui l'accompagne.
Vous découvrerez ainsi un fichier **Quiestce-JeuetGenerateur[Windows/Linux].bat** selon votre OS qui quand il est exécuté lancera une commande dans un terminal pour pouvoir lancer le **quiestce.jar** à l'aide des librairies dont il a besoin.
Les bibliothèques pour windows et pour linux sont déjà dans le dossier **quiestcelib**, mais si vous souhaitez le lancer sur macOS vous pouvez télécharger JavaFX pour macOS grace au lien vu plus haut et mettre le dossier téléchargé dans le dossier **quiestcelib**. Il ne vous reste plus qu'à executer à l'aide d'un fichier batch ou directement dans un terminal ouvert depuis le dossier **qui-est-ce**, la commande qui suit le paterne suivant :
<lien depuis la racine jusqu'à un fichier **java** d'une version 11 ou supérieur> –module-path **./quiestcelib/[nom du dossier JavaFX]/lib/** –add-modules javafx.controls,javafx.fxml -jar quiestce.jar
Ainsi vous pourrez lancer l'application et l'utiliser sur n'importe quel type de machine.

2.1.2 Aperçue de l'interface utilisateur



Vous arrivez alors sur le menu principal de l'application intégrant le jeu et le générateur, vous avez alors plusieurs interactions possibles, vous pouvez cliquer sur **Jouer** pour lancer une nouvelle partie, sur **Sauvegardes** pour reprendre une partie sauvegardé ou encore sur l'engrenage qui est un onglet **Parametre** en haut à droite du menu principale, vous laissant ainsi la possibilité de selectionner la grille que vous souhaitez utiliser pour votre partie (Une grille est selectionné par default si vous êtes pressé de jouer). Vous pouvez aussi cliquer sur la grille en haut à gauche pour entamer ou reprendre la création d'une grille.

Vous avez donc 2 choix possibles pour jouer, commencer une nouvelle partie ou en continuer une préalablement sauvegardé :

Pour entamer une nouvelle partie vous pouvez cliquer sur **Jouer** puis selectionner votre mode de jeu (seulement le mode de jeu solo est fonctionnel) puis choisir si vous voulez jouer avec un ou deux personnages à trouver.
Pour charger une partie antérieurement sauvegardé il suffit de cliquer sur **Sauvegardes**, vous aurez donc la liste avec les noms des sauvegardes que vous avez effectué -veuillez noter que le nom des parties sauvegardés en mode de jeu 2 personnages cachés commencent par **2c-**.

Vous atterrisez ensuite en partie -on parlera ici uniquement du mode de jeu solo avec un unique personnage à trouver-, où vous pouvez poser des questions simple ou complexe sur le personnage à trouver, obtenir des réponses à ces questions et supprimer des personnages de la liste en fonction des réponses obtenue. Ainsi quand vous arrivez au stade où il ne vous reste qu'un seul personnage le jeu vous indique si c'est bien le personnage à trouver ou non et vous indique si vous avez gagné ou perdu. Comme mentionné, vous pouvez poser des questions simple ou complexe.

En partie, vous pourrez expérimenter le mode facile qui va enlever les personnages en fonction de la reponse à la question automatiquement, ainsi que le mode **Triche** où l'utilisateur peut choisir de l'analyser, cette action enclenche la methode Analyser de la classe ControlleurEnJeu/ControlleurEnJeu2 qui affiche dans le ChatBox le nombre potentiel de personnage à éliminer avec la question posée par le joueur. Ainsi, en cliquant sur le boutton **Bannir** l'application élimine les personnages automatiquement pour le joueur, ce dernier n'a pas à bannir personnage par personnage. qui vous donnera un nombre de personnage que vous pourriez eliminer en posant la question choisi et vous permet d'élimer directement ce groupe de personnage en cliquant sur **Ban**. Il y a bien entendu un chat pour communiquer la réponse à chaque question du joueur et aussi pour indiquer où en est le joueur dans ça question.

2.2 Format du fichier JSON, contraintes éventuelles.

Le **.json** contenant les informations d'une grille est structuré de la façon suivante :

```
{
    "personnages" : [
        {
            "id" : 0,
            "photo": "chatminion.png",
            "Nom" : "Chat",
            "Couleur" : ["Noir","Blanc"],
            "Taille" : "Petit"
        },
        {
            "id" : 1,
            "photo": "Goldenretriever.png",
            "Nom" : "Chien",
            "Couleur" : "Roux",
            "Taille" : "Grand"
        }
    ]
}
```

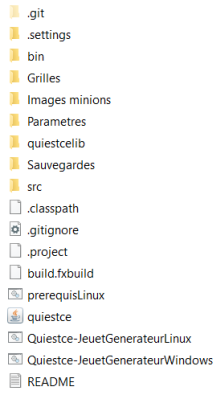
Le fichier json est composé d'une unique liste de **personnages** chacun possédant au moins une **id** unique et un

nom de **photo**. Les arguments des personnages s'ils sont uniques pour un attribut donné peuvent simplement s'écrire entre guillemet, mais s'ils sont multiple doivent apparaître entre crochets pour signifier la multitude.
Une grille est complètement operationnel lorsqu'elle est sous la forme d'un dossier nommé par le nom de la grille, ce dossier étant lui même composé d'un fichier au format JSON nommé par le nom de la grille suivit de l'extension **.json** et ce dossier comporte aussi toutes les images des personnages de cette grille. Ensuite il suffit de placer ce dossier contenant le **.json** et les photos dans le dossier **Grilles**. Toute cette construction est bien sûr gèrer et réalisé par le générateur.
De plus, comme vous pouvez le constater le **.json** ne contient pas de nombre de ligne ni de colone, car notre jeu faisant reference à League of Legends affichera vos images par ligne de 6 comme la salle de selection des champions de League of Legends.

2.3 Description des structures de données, classes, variables utilisées.

2.3.1 Structure du dossier principale

Tout d'abord commençons par la structure de notre dossier principale **qui-est-ce**, voici un petit aperçu de celui-ci après téléchargement sur le lien gitlab :



Le dossier **qui-est-ce** est composé comme suit :

1. Grilles : C'est le dossier qui permet de stocker toutes les grilles terminer ou non. Lors du choix d'une grille sur le menu principale, une methode est appelé pour analyser ce dossier et répertorier tous les fichiers existant, seul les dossier ayant la structure d'un dossier de grille sont à déposer dans ce dossier.
2. Parametres : N'étant pas sûr du nombre de fichier paramètre que notre jeu aurait besoin nous avons choisi de créer un dossier exprès qui actuellement ne contient qu'un fichier **parametres.json**. Ce fichier **.json** analysé à multiple reprise lors des différentes requêtes effectué par l'utilisateur contient les valeurs suivantes :
   (a) **grille** indicant le nom de la grille que vous voulez utiliser pour votre prochaine partie, cette valeur peut etre changer en cliquant sur l'engrenage du menu principale de l'application.
   (b) **partie** qui est mise à jour par une methode du logiciel lorsque vous commencez une nouvelle partie ou lorsque vous chargez une partie sauvegardé -est seulement utile pour savoir si le jeu dois charger les informations d'une partie sauvegardé au pas-.
   (c) **grilleGenerateur** est utilisé de la même manière que **partie** mais pour le generateur.
   (d) **volume** est là pour une potentiel implementation d'une musique d'ambience.
3. quiestcelib : contient les libairies utile au logiciel pour s'executer et fonctionner correctement.

4. Sauvegardes : contient la liste des fichiers sauvegardes au format json. Une sauvegarde contient simplement la liste des personnages déjà bannis, l'id du personnage à trouver, la grille que cette sauvegarde utilise, le nom de cette sauvegarde ainsi que son etat pour savoir si la partie est finie ou pas.
5. prerequisLinux.bat : sert à changer le nom du dossier principale car télécharger depuis gitLab donne au dossier le nom du projet suivit du nom de la branche depuis laquel il est téléchargé, il faut donc le renommer et enlever le nom de la branche.
6. quiestce.jar : est le fichier contenant le code compilé pouvant être lus par le Java Runtime Environment (aka JRE).
7. Quiestce-JeuetGenerateur[OS].bat : sont les fichiers utilisé pour lancer l'application.

2.3.2 Structure du fichier jar, organisation des classes

Le code est organisé en 3 paquets, le paquet **quiestce** qui contient les classes et methodes qui servent à gèrer les requêtes et à adapter les json au format d'une structure de donner en java, le paquet **application** où il y a les differentes classes pour gèrer les onglets ainsi que les methodes qui relient les boutons au méthodes du paquet quiestce et enfin le paquet **resources** sur lequel nous ne nous attarderons pas qui ne contient que les photos.

Le paquet **quiestce** est lui même composé d'un paquet **games** qui contient les classes des differents modes de jeu, le paquet **generateur** qui contient le code necessaire au générateur et enfin le paquet **associations** qui contients des classes necessaire au mode de jeu ainsi qu'au générateur.
Commençont donc avec les 2 paquets principaux qui permettraient au jeu d'etre joué dans un terminal, les paquet **games** et **associations**.
Tous d'abord le paquet **association** :

1. Il y a la classe **Personnage**, une entité quand elle est extraite du json est instancié en un Personnage caracterisé par ces **paires d'attributs et d'argument**, de sont **id** ainsi que de la **grille** à laquel il appartientt. Lors de son instance les caracteristiques d'une entité sont enregistré dans un tableau de paires d'attributs et arguments et sont id est stoqué. La methode clef de cette application ce trouve dans cette classe c'est la methode **verifierArguments** qui prend en paramètres une tableau de paires d'attributs et arguments représentant la question ainsi que le constructeur logique à utilisé dans la question. Elle va vérifier pour le personnage qui fait appel à cette methode si celui-ci posséde les arguments de cette question ou pas et va retourner une valeur de vérité **true** ou **false** en fonction du résultat.
2. En suite il y a la classe **grille** qui est une représentation de la grille du format json mais en format java grace à la librairie Json Simple. Elle sert à extraire les information du **.json** et va analyser ces informations pour stocker les differents attributs et arguments présent sur la grille analysé pour pouvoir ensuite les representer sur l'interface graphique et ainsi permettre à l'utilisateur de poser ces questions. Cette classe est caracterisé par un **nom** -celui de la grille sans l'extion **.json-**, une **taille** qui indique le nombre d'entité présentes dans la grille, et un tableau de **paires d'attributs et arguments** stockant la totalité des arguments de chaque personnage dans leurs attributs respectif.
3. Et en dernier la classe **Sauvegarde** qui aurait dû avoir un role plus important mais qui finalement nous sert principalement à gerer la liste des personnages bannis lors d'un partie.

Maintenant passons au paquet **games** qui contient les classes **GameSolo** qui représente le mode de jeu seul avec un personnage caché et la classe **GameSolo2** que nous verrons dans l'extension. La classe **GameSolo** est caraterisé par une **Grille**, un **Personnage** qui est le personnage à trouver, une **Sauvegarde** pour la liste des bannis, et une liste de **Personnage** qui représente la totalité des entités que la grille contient. Ici nous allons regrouper ces 4 éléments afin de créer une partie c'est pourquoi nous allons seulement instancier la grille puis instancier la liste des personnage, générer un personnage caché de manière aléatoire et enfin instancier une sauvegarde.
Ensuite dans ce paquet **quiestce**, il a le packet **generateur** qui ne contient que la classe **preGrille** sur laquel

nous reviendrons.

Enfin il y a le paquet **application** qui contient toute la gestion des interfaces graphiques. Avec JavaFX nous avons pus créer des fichier **fxml** qui représentent les onglets et lier ces fichiers à des classes pour pouvoir attribuer des méthodes à des boutons et gérer les informations visuellement.

2.4 Forme et traitement des requêtes utilisateur

2.4.1 Description de la forme des requêtes traitées.

Pour effectuer une demande d'information sur le personnage à trouver, si c'est une question simple, il suffit de selectionner un attribut et un argument puis de cliquer sur **Lock In**. Pour effectuer une question complexe il suffit d'appuyer sur **ajouter question** et de selectionner un autre argument d'un autre attribut ou du même attribut, puis choisir si vous souhaitez utiliser des **OU** ou des **ET** entre chaque caracteristiques que vous souhaitez verifier. Enfin, cliquer sur **Lock In** pour obtenir une reponse.

Les question sont donc stocké à l'aide de tableau de paires (en Java les HashMap). Chaque paire de ce tableau est composé d'un mot d'une part (l'attribut) et d'une liste de mot d'une autre part (les arguments) stocké à l'aide d'une ArrayList ; en Java on nommera les mots des String, ce qui nous donne donc le format suivant pour une question : Tableau[Attribut+Tableau[Argument]] en Java HashMap< String attribut, ArrayList<String arguments> >.

2.4.2 Description du traitement effectué lors d'une requête.

Il faut savoir que nous avons choisi de restreindre l'utilisateur à poser des questions qui altérne entre le constructeur **OU** et **ET**. C'est à dire que ce type de question "..OU..ET..ET..OU.." n'est pas pris en charge pas notre application. Néanmoins, l'utilisateur reste libre de poser des questions avec un grand nombre de **OU** ou de **ET**.
La méthode va regarder le constructeur à utiliser, si c'est un **OU** ou un **ET** puis va verifier pour le personnage à trouver si celui-ci possède au moins une caracteristique demandée dans la question ou toutes les caracteristique selon le constructeur. Le bouton **Lock In** va donc appeler la fameuse méthode **verifierArguments** de la classe Personnage sur le **Personnage** caché de notre partie et ainsi l'utilisateur obtiendra une réponse à ça question. Malgré toutes les methodes qui sont appelées pour gérer et afficher les données de l'application du coté du jeu, la méthode la plus coûteuse en temps a seulement une complexité en O(NbPersos* nbAttributs* Max(nbArgumentspourAtt1, ...,nbArgumentspourAttX), c'est la methode lié au bouton **Lock In** quand le mode facile est activé. Ce qui, dans le pire des cas pourrait s'arrondir à une complexité en O(n^3), avec n le max(nbPersonnages, nbAttributs, nbArgumentMax). Comme cette derniére va comparer le resultat de verifierArguments pour tous les personnages restant avec celle de notre personnage caché, elle réalise une première boucle pour comparer tous les personnages restant avec le personnage caché à l'aide de **verifierArguments** qui a besoin d'une double boucle imbriquée afin de vérifier pour chaque attribut de la question si les arguments de cette question correspondent à ce personnage et ainsi éliminer les personnages différents du personnage caché.

3 Étape 2 : Aider à la saisie des personnages

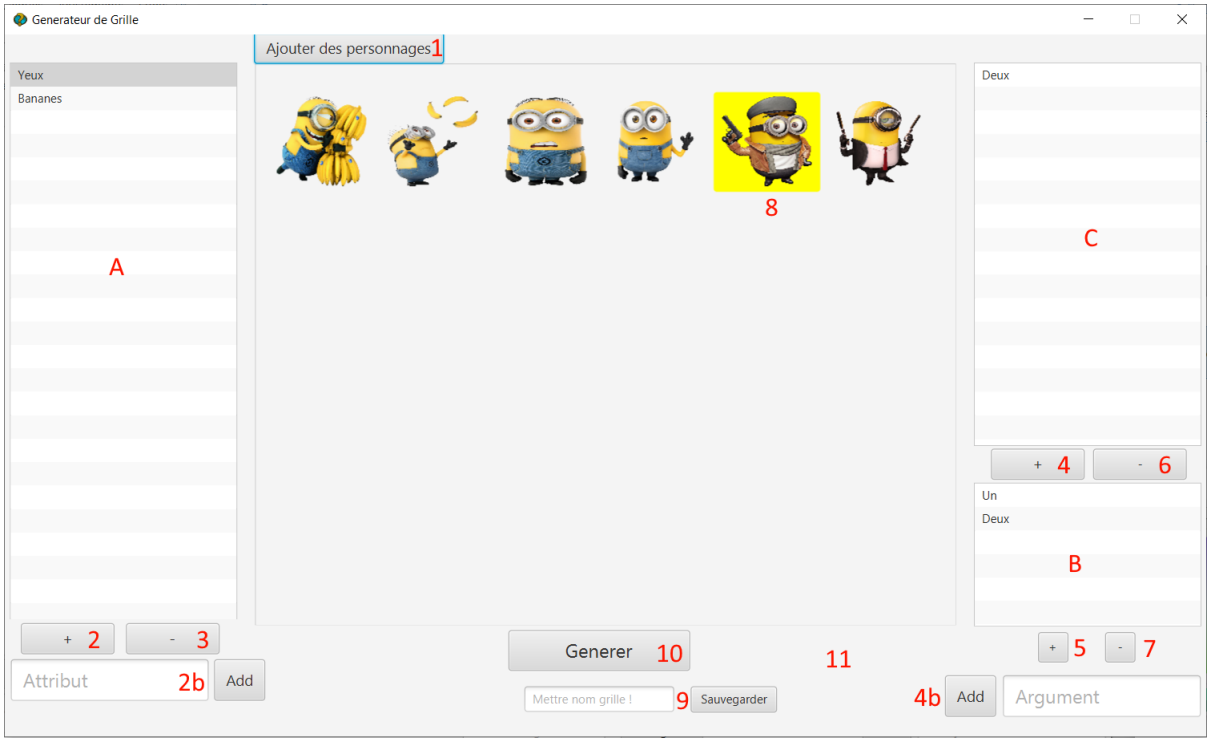
Le choix d'integrer le générateur directement à l'application principal n'est pas anodin. En effet comme nous le verrons, ce choix est purement strategique est nous a permis d'éviter les certaines redondance dans notre code, en plus d'être extrêmement pratique pour l'utilisateur.

3.1 Description du problème : format des données et du résultat.

On veut pouvoir dans cette partie créer des grilles que notre jeu pourra utiliser, c'est à dire des grilles que l'utilisateur pourra jouer. On a gardé le même format que pour une partie c'est à dire un tableau de class.Personnage, un personnage possédant sa liste d'attribut et pour chacun une liste d'arguments, il suffit de cliquer sur l'image d'un personnage sur la grille ainsi que de cliquer sur un attribut pour faire apparaitre ça liste d'argument pour cette attribut. Une class.preGrille est enfaite un regroupement d'une liste d'attribut, une liste d'argument pour chaque attribut ainsi qu'une liste de class.Personnage. On peut ajouter des attributs auquel cas on ajoute cet attribut à tous les personnages déjà presents et qui arriveront, ainsi qu'une liste d'arguments vide. Pour ajouter un argument on peut le voir sous 2 angles, le premier on ajoute un argument à l'historique de cet attribut ou alors on peut ajouter un argument à un personnage pour un attribut donné.

Il est possible de mettre un terme à la saisie de 3 façons soit en sauvegardant, auquel cas on va sans discuter créer un **.json** et écrire dans celui-ci notre liste de Personnage avec leur paires d'attribut et arguments -veuillez noter que sauvegarder une partie la met en mode **SAVE** ne permettant pas de l'utiliser pour lancer une nouvelle partie. La seconde manière de quitter le générateur est de demander de générer la grille, lors de cette demande une methode va s'assurer que chaque personnage est unique et identifiable en une simple serie de question, ainsi qu'aucun personnage possède une liste d'arguments vide pour un attribut donné. Enfin la troisième est de tout simplement la fenêtre du générateur, ne sauvegardant et ne générant en aucun cas la grille en cours de création ou de modification.

3.2 Scénario des interactions avec l'utilisateur.



A l'aide de la photo du générateur on va pouvoir expliquer les différentes interaction possible entre l'utilisateur et l'interface du générateur.

1. Un bouton "ajouter des personnages" vous permet de rajouter des personnages à votre grille quand vous le souhaitez. Ce bouton va ouvrir un explorateur de fichier vous permettant de rajouter des images depuis n'importe quel dossier de votre ordinateur tant que celle-ci soit au format **.png** -veuillez noter que vous pouvez selectionner plusieurs images à la fois en restant appuyer sur controle lors de la sélection dans l'explorateur de fichier-. -Veuillez aussi noter qu'il est impossible d'importer la même image c'est à dire, impossible d'avoir 2 même personnages dans une grille.
2. Ce bouton sert simplement à montrer ou cacher la section 2b.

   — 2b- Grace à ce bouton et la barre de saisie de texte vous pouvez ajouter des attributs pour votre ensemble de personnage, il sera à la suite afficher de façon permanente dans la liste A -veuillez noter que vous ne pourrez pas créer d'attributs vide ni d'attribut déjà existant-.

3. Vous pouvez selectionner un attribut de la liste A et cliquer sur ce bouton pour le retirer de la liste si vous ne le voulez plus.

4. Ce bouton sert simplement à montrer ou cacher la section 4b.

— 4b- le boutton **Add** et la barre de saisie de texte permettent lorsqu'un attribut est selectionné d'ajouter un argument à votre historique -liste B-. Pour pouvoir ajouter des argument à un personnage il faut d'abord avoir créer cet argument à l'aide de cette section.

5. Vous venez de créer un argument pour un attribut donné vous pouvez maintenant ajouter celui-ci à un personnage que vous venez de selectionner en cliquant sur ce bouton. L'argument ainsi selectionné sera alors visible dans la liste C qui indique la liste d'argument pour l'attribut que vous avez préalablement selectionné.

6. Vous pouvez à l'aide de ce bouton supprimer un arguments que vous venez de selectionné de la liste C, c'est à dire supprimer un argument d'un personnage.

7. Ici vous pouvez supprimer un argument de votre historique liste B, ainsi vous pouvez supprimer un argument non utilisé !

8. C'est la grille principale, vous pouvez selectionner n'importe quel personnage pour visualiser sa liste d'argument pour un attribut donné.

9. Vous pourrez sauvegarder votre grille en lui donnant un nom qui n'est pas vide -veuillez noter que lorsque vous voulez modifier une grille sauvegardée ou déjà générée le nom de cette grille se met automatiquement dans la barre de saisie mais vous pourrez tout de même le changer-.

10. Pour générer une grille il faut que celle-ci respecte les critéres suivants : - Il faut que tous les personnages aient des caractéristiques qui permettent de les distinguer des autres personnages présents dans la grille, c'est à dire que si au moins un personnages a exactement les mêmes caractéristiques qu'un autre personnage, un message d'érreur sera affiché et la grille ne pourra pas être générée. - Il faut que tous les attributs soient non vide, c'est à dire qu'ils doivent posséder au moins un argument. Ainsi, si ces critéres sont respectés pas l'utilisateur, la grille sera générée et prête à être utilisée en partie.

11. Pour tenir l'utilisateur au courant de ces actions, nous avons mis au point un **Label** affichant les erreurs commises par ce dernier sous forme de texte en s'actualisant à chaque mauvaise manipulation.

4 Étape 3 : Extension, 2 personnages cachés

4.1 Description précise de l'extension réalisée - choix éventuels

Nous avons souvent fait face a un manque d'organisation, c'est pourquoi nous avons opté pour le choix de 2 personnages cachés pour l'extension de notre application. Ainsi, notre jeu propose à l'utilisateur le choix de lancer sa partie avec 2 sous modes : un personnage caché ou deux personnages cachés. Ces deux modes partagent la même interface graphique à l'exception d'une **ChoiceBox** pour le mode 2 cachés qui permet de diriger la question à un des deux personnages, aux deux ou à aucun des deux. Cette ChoiceBox est utile pour la methode **verifierArgumentPersoCache** qui est redefinit dans la classe **ControlleurEnJeu2** et qui est extend de **ControlleurEnJeu**. Cette methode prend en compte à qui l'utilisateur veut poser la question et avec quel constructeur ce dernier à t-il poser cette derniére. Comme la methode de base cette derniére va verifier si un des deux,les deux ou aucun des deux personnages caché repond aux caractéristiques posés dans la question, en allant parcourir leur caracteristiques c'est à dire : pour un certain attribut voir si ce personnage posséde ou non l'argument posé dans la question.

En ce qui concerne le Mode Facile, ce dernier a la même fonctionnalité que celui d'une partie 1 personnage caché mais remanié pour le type de question que peut poser le joueur dans une partie en mode deux cachés.

Lorsque l'utilisateur sauvegarde une partie en mode 2 personnages cachés, les données de la partie sont stockées dans un fichier **.json** different de celui du mode 1 personnage caché. Le json contient une donnée en plus qui est celle de l'ID du deuxiéme personnage caché. De plus le nom de la sauvegarde prend les caractéres **2C** au debut du nom de la partie. C'est ce qui permet à l'application de différencier une sauvegarde 1 personnage caché d'une sauvegarde 2 personnages cachés et de rediriger l'utilisateur sur l'interface graphique d'une partie en mode 2 cachés lors de la reprise de cette sauvegarde. Ce fichier json sera par la suite enregistré dans le dossier **Sauvegardes** des fichiers sources de l'application.

Voici un exemple d'un fichier json d'une partie 2 cachés et d'une partie 1 caché sauvegardée :

4.2 Une seconde mini extension

Nous avons aussi ajouté une sorte de mini extension au jeu (à la demande de notre professeur référent), cette dernière est disponible en mode 1 et 2 personnages cachés. Cette mini extension est le **Mode Facile** qui est activable en jeu. En cochant la CheckBox de ce dernier vous pouvez activé ce mode qui permet de jouer de manière plus posée. En effet après avoir composé sa question, l'utilisateur en appuyant sur **Lock In** verra tous les personnages qu'il aurait pus éliminer, se faire éliminer automatiquement.

5 Bilan et Conclusions

Lors de ce projet, nous avons eu l'occasion de travailler en groupe et d'avoir un avant goût de ce que pourrait être la vie d'un programmeur en entreprise. Ce sont des points importants, car on a pu élargir nos connaissances dans le domaine technique, à la fois en effectuant des tâches individuelles mais aussi grâce à l'échange et à la communication dans le groupe.

Ce projet nous servira pour la suite de notre parcours comme un rappel des choses à faire et à éviter. Le produit final de notre travail se rapproche beaucoup de l'application que nous avions schématisé dans notre premier rendu de maquette en debut de semestre, et pour ça nous somme largement satisfait de l'aspect visuel et interactif de notre jeu Qui est-ce. Nous avons aussi eu l'occasion de manier quelques librairies que java nous mettait à disposition, notamment celle que nous avons le plus utilisé **JavaFX**, que nous avons pu manier par l'intermediaire du logiciel **SceneBuilder** qui nous a été d'une grande aide.

Cependant, Quand on veut aller trop vite, les obstacles se multiplient. Nous avons évidemmment rencontré des difficultés qui ont trés souvent été liées à des mécompréhension et à un manque de maniabilité notamment avec gitlab, qui pour nous, était quelque chose de nouveau et pas forcement facile à utiliser au début. Mais nous avons réussi à tirer profit de ce genre de difficultés et obstacles, car ces derniéres nous ont poussé à mieux nous informer, à faire d'avantage d'efforts et plus de recherches.

Ainsi, tout au long de la préparation de notre projet , nous avons essayé de mettre en pratique les connaissances acquises durant nos études et cela dans le but de réaliser ce jeux. Ce projet a définitivement élargi nos connaissances et nous a apporté beaucoup de plus à notre expérience en groupe et individuelle.