UNIVERSITÉ
DE MONTPELLIER

FDS Montpellier

L2 Informatique
Rapport

Projet de programmation - Groupe B

Élèves :
Michel BE
Matthias BLANC
Adam DAIA
Paul FONTAINE

Enseignants :
Philippe JANSSENN
Sylvain DAUDÉ
Anne-Élisabeth BAERT
Clémentine NEBUT
Nicolas POMPIDOR
Julien DESTOMBES

16 avril 2022

Table des matières

1 Description brève du projet et des étapes de sa réalisation

Le projet consiste à développer une interface utilisateur graphique (couramment appelée GUI) permettant de jouer au Qui Est-ce ?
-Nous avons d'abord réalisé un jeu de Qui-est-ce ? simple en java.swing, en prenant pour personnages des sportifs internationaux.
-Nous avons ensuite rapidement ajouté une fonctionnalité permettant de sauvegarder une partie en cours (sur demande explicite du joueur) et de la reprendre.
-Nous avons ensuite travaillé sur un générateur permettant de créer ses propres sportifs pour jouer.

Comme le générateur demandé devait être davantage générique nous avons travaillé sur deux améliorations parallèles du programme :
-Une version comportant un générateur et la possibilité de jouer avec les personnages générés et de sauvegarder.
-Une version ne permettant ni de sauvegarder ni de jouer avec des personnages générés par le joueur, mais permettant de jouer avec deux personnages cachés et des questions avec un connecteur logique ("et","ou").

2 Technologies utilisées et organisation

Après discussion, nous avons décidé d'utiliser Java, avec les bibliothèques Swing pour le rendu graphique et Gson pour pouvoir décoder et encoder les différents fichiers JSON.

2.1 Choix du langage

2.1.1 Java

Nous avons choisi de développer entièrement le projet en langage Java. Java était un langage simple et multi-plateformes que nous connaissions déjà, et son typage fort nous apparaissait comme une qualité au moment du choix.

2.1.2 Swing, Gson

Pour l'interface graphique notre choix s'est porté sur l'ancienne bibliothèque Swing, car elle est disponible dans la version 8 du JDK, donc installée par défaut sur les ordinateurs de la faculté.

Pour la manipulation des personnages, Gson est la bibliothèque la plus adaptée car elle est simple mais puissante, elle permet de produire et parser du JSON aisément.

3 Organisation du travail

3.1 Rythme de travail, mode de fonctionnement

Nous travaillions trois heures ensemble par semaine, le lundi lors des séances de TP, en restant souvent après 16h30, et chacun travaillait de son côté à son rythme durant la

semaine.

3.2 Organisation du travail au sein du groupe

Au début de chaque séance, nous nous fixions des objectifs à atteindre avant la séance prochaine,puis nous trouvions des solutions pour les atteindre. Ensuite, nous nous répartissions les tâches en fonction des compétences de chacun. Nous avons commencé à programmer sans modélisation préalable, ce qui a permis des cycles de développement courts au début, et une production rapide de la première partie du projet. Cependant, cela causait une difficulté pour mettre à jour le code au moment d'enrichir l'application.

4 Étape 1 : démarrage du jeu de base (concerne toutes les versions finales)

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

Quand l'utilisateur démarre le jeu, il arrive sur une page d'accueil. Il suffit à ce moment là de cliquer pour lancer une partie.

Après le clic sur démarrer, les personnages sont chargés depuis un fichier JSON, et l'un d'eux est choisi au hasard. Une nouvelle fenêtre s'ouvre où l'on peut voir en haut à gauche trois onglets. L'onglet "Fichier" permet de sauvegarder une partie, d'en redémarrer une ou encore reprendre la dernière sauvegarde. Ensuite on trouve "Modes de jeu" qui permet de jouer au mode "2 personnages" ou "2 personnages complexe" (nous y reviendrons plus tard). Au milieu on retrouve l'onglet "Thème", comme son nom l'indique il permet de choisir entre différents thèmes (par défaut, jaune, orange, rose, vert et bleu). Pour finir le bouton "Aide" permet d'avoir quelques indications sur le jeu et d'afficher les crédits, mais aussi d'afficher un tableau avec les caractéristiques de chaque personnage au cas où l'utilisateur ne les connaît pas tous.

Pour jouer, l'utilisateur est en présence de 16 personnages au plus. Via le panneau en dessous de leurs images il choisit une question à poser (nationalité, âge etc.) et une réponse apparaît. L'utilisateur peut continuer jusqu'à deviner le personnage choisi, pour s'aider il peut cliquer sur une ou plusieurs images pour éliminer des personnages (les images en questions sont remplacées par une croix rouge), l'action étant réversible. Lorsqu'il pense avoir trouvé, il sélectionne le nom du personnage dans le panneau du bas et une fenêtre s'ouvre avec le message "Dommage, veuillez recommencer" ou "Félicitations vous avez trouvé le personnage !".

4.2 Représentation JSON des personnages, contraintes éventuelles

Le fichier JSON chargé au démarrage d'une nouvelle partie, comporte un tableau Java, dont chaque élément est un objet JSON sans sous-objet. Le nom des personnages, comme le nom de l'image associée ou les caractéristiques concernées par les questions du joueur, sont représentées sous la forme "clé" :"valeur".
Dans le programme, les personnages étaient représentés dans des objets de types Personnage où chaque caractéristique (type de sport, classe d'âge, genre, longueur des cheveux

...) correspondait à un attribut en Java. De plus, chaque caractéristique et ses différentes valeurs possibles étaient représentées par une énumération Java.
Deux avantages constatés pour cette première approche :
-Les listes déroulantes du jeu pouvaient chacune être associées à une classe d'attribut, (il y avait une JComboBox<Cheveux>, une JComboBox<Genre> ...) et le compilateur pouvait ainsi faire d'avantage de vérifications de type.
-L'import des personnages via la bibliothèque Gson était alors trivial (ou presque) grâce à l'introspection. Il suffisait d'utiliser l'appel de méthode *readJson(input,Personnage[].class)* où "input" est le tableau json sous forme de String.

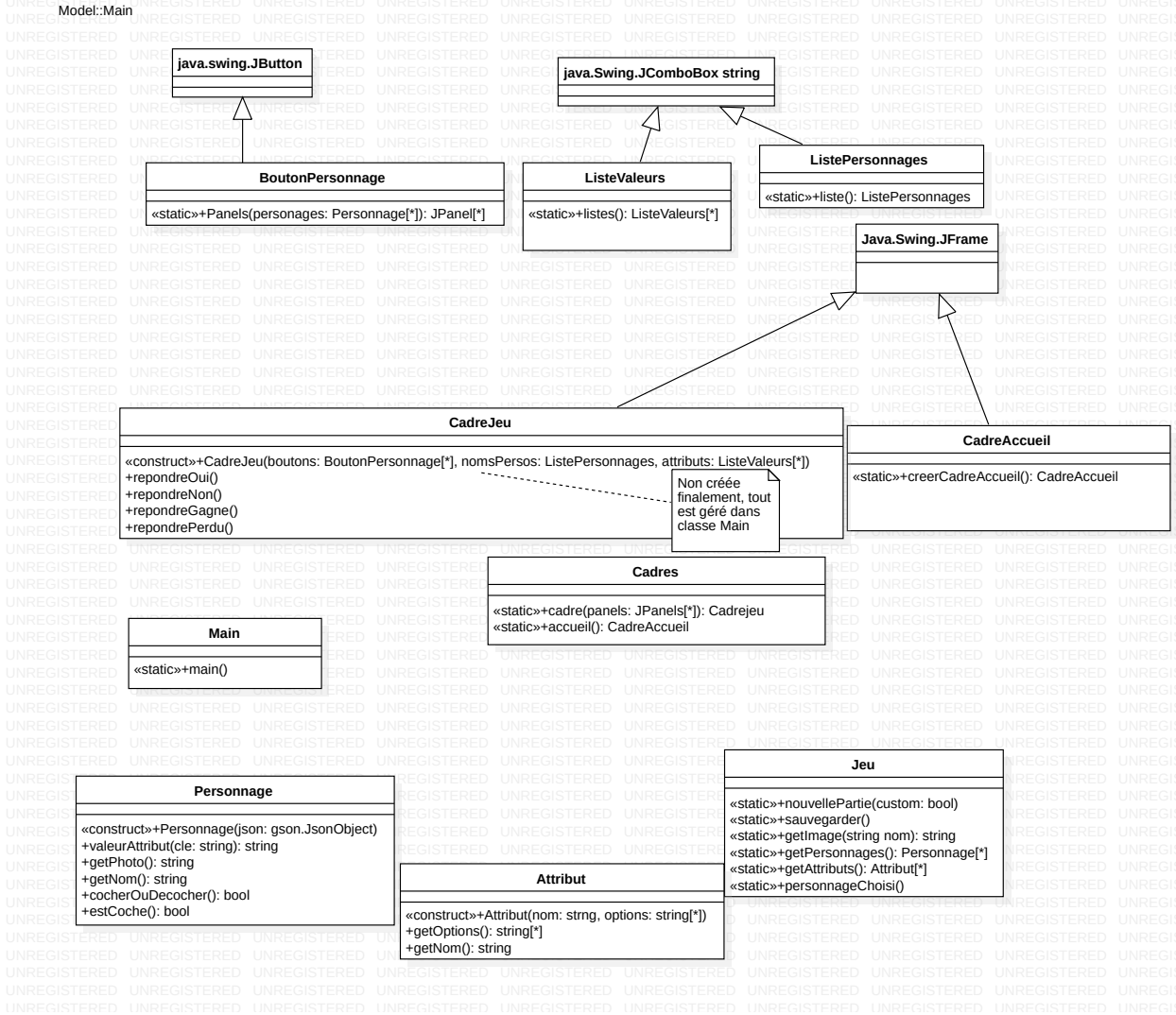
Cependant cette représentation en Java des personnages a du être drastiquement repensée pour intégrer un type de personnage personnalisé par l'utilisateur via le générateur. En définitive la classe Personnage comportait un unique attribut représentant l'objet Json associé (de type gson.JsonObject), et les JComboBox sont toutes devenues des JComboBox<String>.

4.3 Représentation JSON des parties sauvegardées

Une partie sauvegardée est une représentation Json d'un objet Java de type Jeu, qui comporte pour attributs :
-Le tableau des personnages, chacun augmentés de l'attribut "coché" indiquant si l'utilisateur avait éliminé le personnageen question.
-La position dans le tableau du personnage à deviner.

4.4 Description des structures de données, classes, variables utilisées

Ci-dessous le modèle UML de l'application finale (hors générateur qui tient dans un seul .java) en ne représentant que les méthodes publiques (une classe OuvrirFichier qui expose différentes variables et méthodes statiques a également été crée)



4.5 Requetes traitées en cours de partie

-Une fois que le joueur a choisi une valeur d' attribut de la liste déroulante on vérifie la correspondance par rapport à l'instance choisie au hasard , chaque nombre correspondant à un personnage) la classe personnage, puis on affiche "oui" ou "non" (le personnage choisi état stocké en permanence en mémoire centrale)
-Si le joueur coche ou décoche un personnage, l'attribut coché ou décoché change en mémoire centrale pour le personnage en question, et l'image de croix, préalablement chargée vient se substitué à l'image du personnage en question (en pratique l'ensemble des attributs de tous les personnages est chargé au moment du parsing du fichier JSON, bien que certaines données ne soient pas utilisées)
-Si le joueur sélectionne le nom d'un personnage, la comparaison avec le nom du personnage choisi est effectuée, et la fenêtre de victoire ou d'erreur est affichée
Dans tous les cas, on a pris soin de comparer les String avec la méthode equals et non l'opérateur ==.

5 Étape 2 : Générateur et création manuelle de personnages

5.1 Pourquoi le développement du projet est devenu plus difficile au moment d'ajouter le générateur

A ce moment là, intégrer au jeux existant un générateur aussi générique que voulu devenait difficile sans changer radicalement le code du prototype. Nous avons donc pris du retard, puis fini par nous partager le travail en deux : certains étendant le prototype pour enrichir le "gameplay" du jeu, et d'autres travaillant à développer le générateur et à retravailler le code du jeu de base pour le rendre compatible avec les personnages générés.

5.2 Mode d'emploi du générateur

L'utilisateur entre dans la première colonne les noms des personnages, dans la deuxième les noms des images. Les colonnes suivantes servent à renseigner le nom des attributs, c'est à dire l'étiquette/le nom de l'attribut dans la première colonne et pour chaque personnage la valeur de l'attribut.
Après clic sur le bouton d'export (panneau de droite) l'intégrité du jeu de personnage créé est partiellement vérifiée (unicité des noms) et les personnages créés sont exportés dans un emplacement attitré à l'avance

6 Étape 3 : Mode de jeu avec deux personnages à trouver

6.1 Description précise de l'extension réalisée

Nous avons réalisé deux versions étendues du jeu de base, similaires (chacune ayant sa classe principale) mais d'une difficulté différente. La première est un mode avec deux personnages, le principe et le fonctionnement reste le même que pour le mode à un seul joueur. La deuxième extension est un mode à deux joueurs mais avec des connecteurs logiques ("et", "ou" et "non")

6.1.1 Mode deux personnages

Pour le mode classique à deux personnages il suffit au joueur de poser des questions mais il doit trouver des questions qui permettent d'en déduire le personnage, par exemple pour le Genre la réponse peut être féminin et masculin ou encore "à une barbe" et "est imberbe" dans le cas où un homme et une femme sont à trouver. Une fois que l'utilisateur pense avoir trouvé la solution il saisit les deux personnages un après l'autre.

6.1.2 Mode deux personnages complexe

Le mode 'deux personnages complexe' porte bien son nom, le connecteur "!" est le plus simple ; rien ne change du mode "deux personnages" hormis le fait que toutes les réponses sont inversées.

Le mode "et" (double esperluettes) est déjà plus complexe. Par exemple si vous choisissez "Masculin" puis "Imberbe" ; et si la réponse est oui, cela veut dire que les deux personnages sont des hommes imberbes. Un autre exemple, si les personnages sont LeBron James (joueur de basket) et Kimi Räikkönen (pilote de Formula1) et que l'utilisateur choisit "sport collectif" et "sport individuel" la réponse sera négative car ils ne font pas un sport collectif et individuel, par conséquent on ne trouvera jamais une réponse positive en prenant deux attributs du même menu déroulant, ce qui limite les possibilités et augmente la difficulté. Si on reprend notre exemple, une réponse juste serait "Masculin" et "Sport individuel"

Le dernier connecteur est "||". Il renvoie "oui" si une de vos deux propositions est juste. Ce qui fait augmenter la difficulté est le fait qu'on ne sait jamais la quelles de nos deux propositions est juste. Par exemple si les personnages sont Perrine Laffont (skieuse) et Leo Messi (joueur de football) et qu'on sélectionne "Imberbe" || "Sport collectif", il nous est impossible de savoir si un des personnages est imberbe et fait du sport collectif, imberbe et sport individuel ou encore porte la barbe et fait un sport collectif. Ce mode de jeu nous oblige à retenir la majorité des réponses car elles restent inexploitables tant que l'utilisateur n'a pas posé un certain nombre de quittons. Dans certains cas on en vient même à penser qu'il y a une erreur.

7 Bilan et conclusions

7.1 Ce qui a fonctionné

Nous sommes parvenus à implémenter les principales fonctionnalités demandées : chargement des personnages, choix aléatoire d'un personnage et réponse aux questions, questions avec connecteur logique, générateur permettant d'exporter un jeu personnalité de personnage.

7.2 Ce qui n'a pas fonctionné

Par manque de temps et par une mauvaise compréhension au départ des consignes, le générateur ne dispose pas à ce jour (16 avril) de toutes les fonctionnalités de vérifications (vérification que tous les personnages sont différents, et vérification qu'ils sont tous complets).
Les interactions avec le système du fichier sont trop rigides et limitées (démarrage d'une nouvelle partie non-supportée en dehors de Linux, un seul emplacement possible pour sauvegarder la partie, un seul également pour sauvegarder les personnages créés, dossier des images fixé relativement à l'emplacement d'exécution)

7.3 Problèmes rencontrés et Conclusion

La bibliothèque Swing n'encourage pas particulièrement un modèle MVC (séparation des classes liées au traitement), chacun des composants interactifs qu'elle propose est aisément utilisable comme un pont, une liaison, entre une partie des données et leurs affichages, le code aurait cependant gagné en qualité générale en consacrant à chaque type se composant sa classe java (champ de saisie d'une valeur, pour le générateur, champ de saisie d'un type d'attribut, liste des personnages pour faire une proposition).

Nous avions choisi collégialement de ne pas établir de modèle UML au début du développement, cela nous a posé des problèmes importants pour grossir le code en travaillant ensemble parallèlement.

Hormis un approfondissement du langage Java, l'importance de convenir d'un modèle UML est probablement le plus grand enseignement que ce travail d'équipe nous a apporté.