Qui est-ce ?
Projet de programmation

Groupe *Z*
Frédéric, Laurent, Tony et Romain
git:git@gitlab.etu.umontpellier.fr:e20180001091/qui-est-ce.git
L2 informatique
Faculté des Sciences
Université de Montpellier.

April 16, 2022

[redacted]

[redacted]

Figure 1: Squelette type de nos fichier JSON

```
{
    "metadonnees": {
        "images": "../ressources/img/<theme>/",
        "ligne": 0,
        "colonne": 0,
        "largeurImage": 0,
        "hauteurImage": 0,
        "espacementPhoto": 0,
        "tailleMaxHauteurImage": 120
    },
    "possibilites": {
        "0": {
            "fichier": ["NomImagePerso1.jpg"],
            "nom": ["Perso1"],
            "attribut1": ["A", "B"],
            "attribut2": [42]
        },
        "1": {
            "fichier": ["NomImagePerso2.png"],
            "nom": ["Perso2"],
            "attribut1": ["B", "C"],
            "attribut2": [12]
        }
    }
}
```

Abstract

Nous avons réalisé l'IHM et le jeu/générateur en parrallèle. De nombreux problèmes d'organisation et de git nous ont ralentis.

1 Étape 1 : permettre à l'utilisateur de jouer

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

Le jeu (de base)

Après avoir lancé le terminal, nous entrons notre nom, choissisons le plateau, puis mode facile ou difficile.

Une fois en jeu, nous pouvons coché des portrait (clic gauche) et accuser un personnage (clic droit) En bas de la fenêtre, des bouttons nous permettent de créer notre question, de la poser puis voir la réponse. Un haut, un petit menu vous permettra de sauvegarder la partie en cours, voir votre historique de question et abbandoner.

Le générateur

Le joueur doit lancé le générateur par le terminal en choisisant en paramètre le thème.

En ce qui concerne les métadonnées (taille de la grille, des images), il est possible de les modifier si les valeurs par défaut ne conviennent pas.

Ensuite, l'utilisateur doit définir des attributs puis y ajouter des valeurs afin de les utiliser par la suite.

Par ailleurs, les attributs avec leurs valeurs peuvent être donner à des personnages.

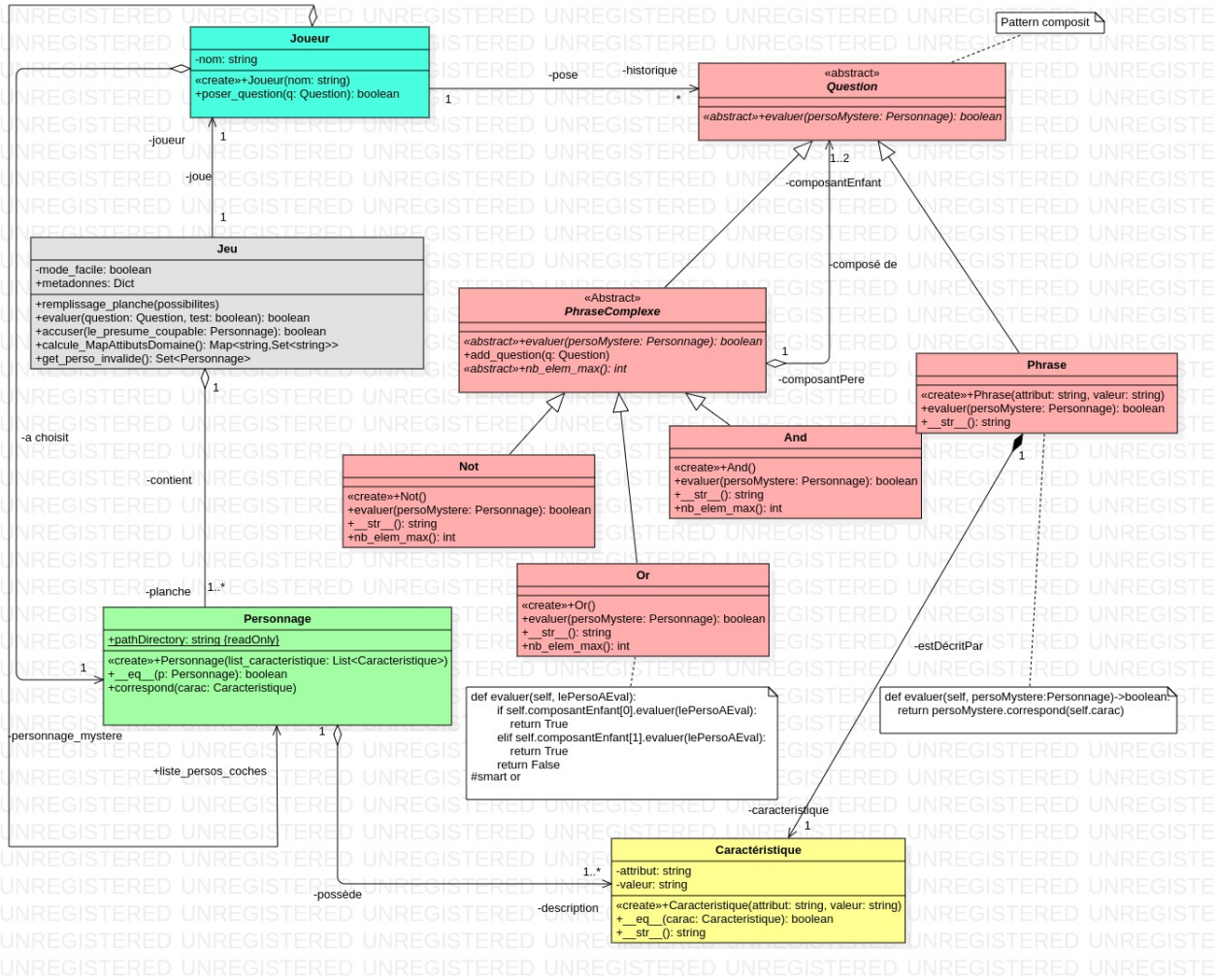
Il n'y a pas d'autré pré-établi si ce n'est que il vous faut au moins un attribut avec au moins 1 valeur pour pouvoir la donner à un personnage.

Nous avons la possibilité d'ajouter des attributs, d'ajouter des valeurs à ces attributs, modifier et supprimer. Ces Fonctionnalités ont des répercussions sur les personnage qui les portent.

Format du fichier JSON, contraintes éventuelles

"metadonnees" Cette clé renvoie vers des données essentiels pour le bon fonctionnement de l'interface graphique.
"images" contient le chemin vers le dossier contenant les images depuis la racine du projet
"ligne" et "colonne" forment la taille de la grille
"largeurImage" et "hauteurImage" correspond à la définition de chaque portrait à l'écran (!= la définition réel des images dans le dossier)
"espacementPhoto" est l'espacement entre les portrait sur l'interface graphique
"possibilites"
Chaque clé correspond à la représentation d'un personnage. Chaque personnage doit avoir les clés "fichier" et "nom". Le reste des clés du personnages correspond aux noms de ces attributs. Les clés des personnages sont associés à des liste pour permettre la multiplicité des valeurs. Les personnages ont le même nombre d'attribut.

Figure 2: Diagramme de classe du jeu



Description des structures de données, classes, variables

La classe Jeu ne contient pas de logique. Elle sert simplement à réunir les autres objets dans un unique objet, facilité la création des objets et contient quelques méthodes "utiles". La méthode calculeMapAttributsDomaine() renvoie les attributs et leurs valeurs sous forme de dictionnaire pour l'interface Nous considérons que poser une question sur le nom d'un personnage et sur ses attributs sont philosophiquement différentes, par conséquent il d'un côté accuser() pour identifier ou pas notre personnage mystère et evaluer() pour poser des questions.

Figure 3: Diagramme objet représentant une question

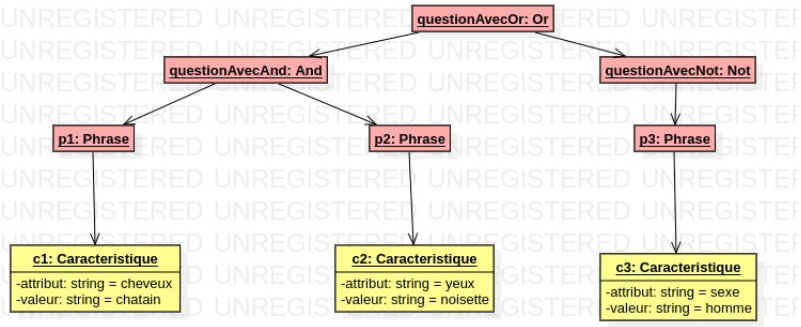
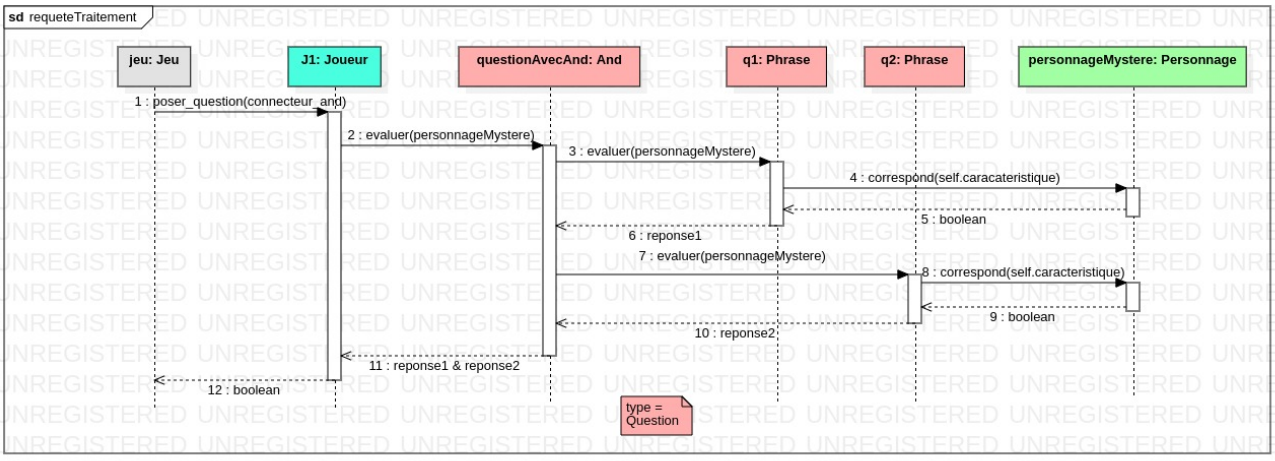


Figure 4: Diagramme de séquence, le joueur pose une question



Description de la forme des requêtes traitées et un traitement.

Nous avons choisit d'utiliser un pattern composite pour représenter les questions. En effet, la situation si prete bien. Puisque dans notre Qui-est ce?, nous voulons qu'un utilisateur puisse intéroger sur plusieurs choses à la fois tout en représetant sufisament précisement la complexité d'une question. Nous voullons qu'une question puisse en contenir d'autre et être relié par un connecteur logique donné. En cela, le composit pattern remplis notre besoin car une question est un arbre dont les différents nœuds sont des connecteurs logiques, et les vrais questions sont les feuilles de cette arbre. Et un sous arbre serait également une question valide.

Description du traitement effectué lors d'une requête.

Lorsque que le joueur pose une question (1), l'objet J1 de type Joueur va passé le personnage mystère pour que la question qu'il veut posé puisse être évaluer. La question questionAvecAnd va s'évaluer (2) récursivement avec ces enfants (3,7) jusqu'à qu'elle tombe sur une feuille. Les phrases au bout de l'arbre vérifie si ils correspond au personnage passé en argument (4,8).