

Activité d'intégration  
UE 210  
Projet

Bachelier en Informatique de gestion Mons  
**What do you know about it ?**

*Groupe B05*

*Giorgio Caculli LA196672, Guillaume Lambert LA198116, Tanguy Taminiau LA199566*

2020 - 2021



## Table des matières

<b>Introduction</b>	<b>2</b>
<b>1 Présentation du sujet</b>	<b>2</b>
1.1 Règle du jeu . . . . .	2
1.2 Position du problème . . . . .	2
<b>2 Description général de l'application</b>	<b>2</b>
<b>3 Analyse</b>	<b>4</b>
3.1 Product backlog . . . . .	4
3.2 Les classes . . . . .	5
3.2.1 Le modèle . . . . .	5
3.2.2 La vue . . . . .	6
3.2.3 Les exceptions . . . . .	7
<b>4 Design pattern</b>	<b>8</b>
4.1 Iterator . . . . .	8
4.1.1 Explication générale . . . . .	8
4.1.2 Diagramme de classe . . . . .	8
4.1.3 Extrait de code source . . . . .	8
4.2 Template Method . . . . .	10
4.2.1 Explication générale . . . . .	10
4.2.2 Diagramme de classe . . . . .	10
4.2.3 Extrait de code source . . . . .	10
<b>5 Veille technologique</b>	<b>12</b>
5.1 Launcher . . . . .	12
5.2 Chat en ligne . . . . .	12
5.3 Traduction dynamique des menus . . . . .	12
<b>6 Implémentation</b>	<b>13</b>
6.1 Menu principal . . . . .	13
6.2 Règle et menu jouer . . . . .	14
6.3 Menu jeu . . . . .	16
6.4 Menu multi joueurs . . . . .	17
6.5 Menu multi joueur en ligne . . . . .	18
6.6 Menu paramètre . . . . .	19
6.7 Menu d'administration . . . . .	20
6.8 interface "New Card" . . . . .	21
6.9 Tableau carte . . . . .	22
<b>7 Tests unitaires</b>	<b>23</b>
<b>8 Conduite du projet</b>	<b>26</b>
8.1 Tableau Trello sprint 1 . . . . .	26
8.2 Tableau Trello sprint 2 . . . . .	27
8.3 Tableau Trello sprint 3 . . . . .	28
<b>9 Conclusion</b>	<b>28</b>

## Introduction

Dans le cadre du cours de “Projets” de l’UE 210, nous avons dû créer un jeu similaire à “Tu te mets combien?” (TTMC). L’objectif pédagogique de ce projet est de pousser l’élève à mieux appréhender la librairie JavaFX vue au cours de POO (programmation orienté object). De plus, la grande liberté accordée à ce projet oblige l’élève à apprendre à se documenter, savoir faire des recherches et lui permettre de se rendre compte de ce qu’est un véritable projet de groupe, organisé via la méthode SCRUM. L’objectif du projet est simple, créer un jeu sur le principe de TTMC (Tu Te mets Combien?).

## 1 Présentation du sujet

Il s’agit d’un jeu comportant des cartes composées des questions. Il existe quatre thèmes repérable par leurs couleurs respectives, la couleur :

- mauve est attribué aux cartes ayant pour thème “improbable”
- orange pour “plaisir”
- bleu pour “informatique”
- vert pour “scolaire”

Chaque carte possède un thème, un sujet en rapport avec le thème, et quatre questions. Les question sont numérotées de un à quatre et triées par ordre croissant de difficulté.

### 1.1 Règle du jeu

Le jeu a pour but de démontrer que l’on se connaît mieux que les autres se connaissent eux-même. Pour le montrer, il faut arriver au bout du plateau tout en gagnant des points. Pour avancer, il faut répondre correctement aux questions qui seront posées, une bonne réponse vous fait avancer d’une case. Vous pouvez choisir entre quatres difficultés de questions, en fonction de ce que vous connaissez le mieux. Le niveau 1 est le plus facile et le niveau 4 le plus difficile. A chaque bonne réponse, vous gagnez un nombre de points équivalant au niveau de la question. Par exemple : vous répondez correctement à une question de niveau 1, vous gagnez 1 point. Si vous répondez correctement à une question de niveau 4, vous gagnez 4 points. Lorsqu’un joueur arrive sur la dernière case du plateau, il est désigné gagnant si il est le seul à avoir atteint la fin du plateau ou (si ils sont plusieurs sur la dernière case) si il a le plus de points.

### 1.2 Position du problème

La création d’un jeu vidéo étant une première pour nous, le problème résidait surtout dans la manière dont nous allions structurer notre projet, mais aussi imaginer ce jeu autant dans sa présentation graphique que dans les fonctionnalités à incorporées. Les fonctionnalités du jeu nous ont demandé beaucoup de réflexions surtout que nous avions de grandes ambitions.

## 2 Description général de l’application

L’application inclut beaucoup de fonctionnalités qui ont été vues au cours. Les technologies suivantes ont été introduites lors du développement de notre application :

- le Design Pattern Iterator
- le Design Pattern Template Method
- des exceptions customisés permettant de nous donner un message plus claire à un problème rencontré lors d’une quelconque manipulation
- des tests unitaires afin d’assurer que chaque manipulation se passe correctement
- la librairie GSON qui nous a facilité la sérialisation et désérialisation des données
- la librairie JavaFX qui nous a permis de développer un jeu avec un visuel très modernisé
- l’utilisation de ResourceBundle afin de nous permettre d’instaurer un système d’internationalisation dans notre programme

Ces ne sont qu’une petite partie des technologies que nous avons mis en place pour notre programme. Afin d’assurer une meilleure portabilité de notre programme pour des gens n’utilisant pas d’arguments dans leur VM de Java, ou qui utiliseraient le JDK de base, nous avons décidé de mettre en place un « launcher ». Ce dernier

sert à détecter le système d'exploitation de l'utilisateur, sa version du JDK, et l'endroit où il est en train de lancer le launcher-même. Une fois que toutes ces informations sont chargées, ce launcher va automatiquement lancer le jeu avec les paramètres nécessaire pour faire fonctionner **JavaFX**. Cette solution a énormément simplifié le partage de notre application avec les « Beta Testers ».

### 3 Analyse

#### 3.1 Product backlog

US-01	En tant qu'utilisateur je voudrais savoir mon score.
US-02	En tant qu'utilisateur je voudrais savoir si j'ai bien répondu.
US-03	En tant qu'utilisateur je voudrais savoir si j'ai mal répondu.
US-04	En tant qu'utilisateur je voudrais savoir quelle était la bonne réponse.
US-05	En tant qu'utilisateur je voudrais savoir mettre mon jeu sur pause.
US-06	En tant qu'utilisateur je voudrais savoir reprendre mon jeu où je l'avait laissé.
US-07	En tant qu'utilisateur je voudrais savoir arrêter mon jeu à tout moment.
US-08	En tant qu'utilisateur j'aimerais joué en multi joueur localement.
US-09	En tant qu'administrateur je dois pouvoir ajouter une nouvelle carte au deck.
US-10	En tant qu'administrateur je veux pouvoir supprimer une carte du deck.
US-11	En tant qu'administrateur je veux pouvoir modifier une carte existante.
US-12	En tant qu'utilisateur j'aimerais avoir une musique de fond.
US-13	En tant qu'utilisateur j'aimerais pouvoir gérer le volume de la musique.
US-14	En tant qu'utilisateur j'aimerais pouvoir activer ou désactiver la musique de fond.
US-15	En tant qu'utilisateur je voudrais pouvoir choisir mon propre pseudonyme.
US-16	En tant qu'utilisateur je voudrais voir un plateau de jeu.
US-17	En tant qu'utilisateur je voudrais avoir mon pion.
US-18	En tant qu'utilisateur je voudrais savoir reconnaître mon pion.
US-19	En tant que joueur, je voudrais communiquer avec d'autre joueurs.
US-20	En tant que joueur, j'aimerais jouer avec d'autre joueurs en ligne.
US-21	En tant que joueur, j'aimerais rejoindre une partie en ligne.
US-22	En tant que joueur, j'aimerais héberger une partie en ligne.

## 3.2 Les classes

### 3.2.1 Le modèle

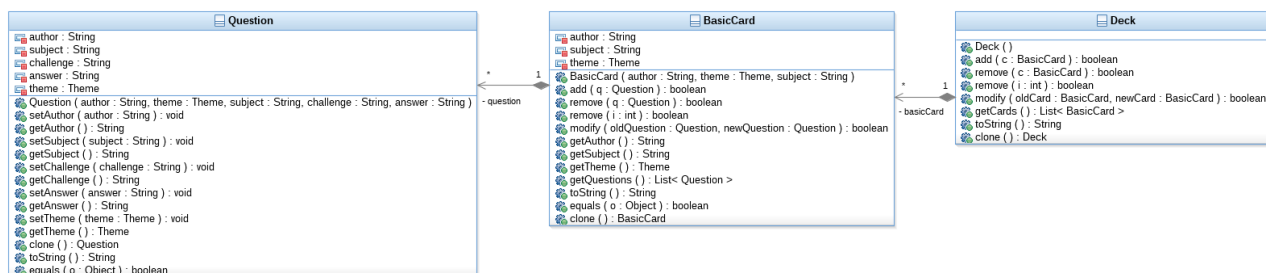


FIGURE 1 – Diagramme du modèle

Le modèle est composé principalement de 3 classes et une énumération. La première classe que l'on va analyser est la classe **Question**. Une question est caractérisée par :

- un auteur
- un sujet
- une question
- une seule réponse possible
- un thème

Étant donné que cette classe ne fait pas de manipulations particulières elle est principalement composée de **Setters** et **Getters** propre à chaque attribut. Afin de vérifier un cas de similitude entre deux questions, une fonction `equals()` a été écrite. Pour permettre une nouvelle instantiation de cette classe, une fonction `clone()` a été mise en place. Enfin, afin d'assurer une sortie facilement lisible à l'être humain, la fonction `toString()` a été surchargée.

La deuxième classe qui caractérise le modèle est la classe **BasicCard**. Une carte dans notre jeu est caractérisée par les attributs suivants :

- un auteur
- un sujet
- un thème

Une **BasicCard** fait des manipulations plus complexes que la classe **Question**. Une carte est **composée** de une ou plusieurs questions, ceci doit être capable d'ajouter des nouvelles questions à sa liste, de supprimer des questions, ou remplacer une question par une autre. Tout comme la classe **Question**, la classe **BasicCard** est aussi composée de **Setters** et **Getters**. La classe **BasicCard** aura aussi sa propre fonction `equals()`, qui permettra la vérification de doublons, lors de l'ajout de ceci dans un deck. Sa propre fonction `toString()` affichera bien les attributs qui la caractérisent ainsi que les questions qui la composent. La méthode `clone()`, qui sert tout de même à renvoyer une nouvelle instance de la carte, servira principalement à assurer la **composition** entre la classe **Deck** la classe **BasicCard**.

La troisième et dernière classe qui fait des manipulations toutes autant complexes est la classe **Deck**. Cette classe n'a pas d'attributs qui la caractérisent car elle sert principalement à stocker les cartes. Tout comme la classe **BasicCard** cette classe aussi doit être capable d'ajouter des nouveaux éléments à sa liste, doit pouvoir supprimer un ou plusieurs éléments et doit être capable de remplacer un élément par un autre. Étant donné que cette classe ne sera pas stockée dans une autre classe, elle ne nécessite pas de `equals()`. Tout comme les autres classes, elle possède aussi son propre `toString()` qui lui permettra d'afficher toutes les cartes qui seront stockées à l'intérieur de sa liste. Afin d'assurer la sécurité de ses données nativement introduites, il est possible de cloner le deck grâce à la fonction `clone()` qui créera une nouvelle instance de la classe **Deck** originale, copiera toutes les cartes présentes dans le deck original et les ajoutera dans la liste de la nouvelle instance.

### 3.2.2 La vue

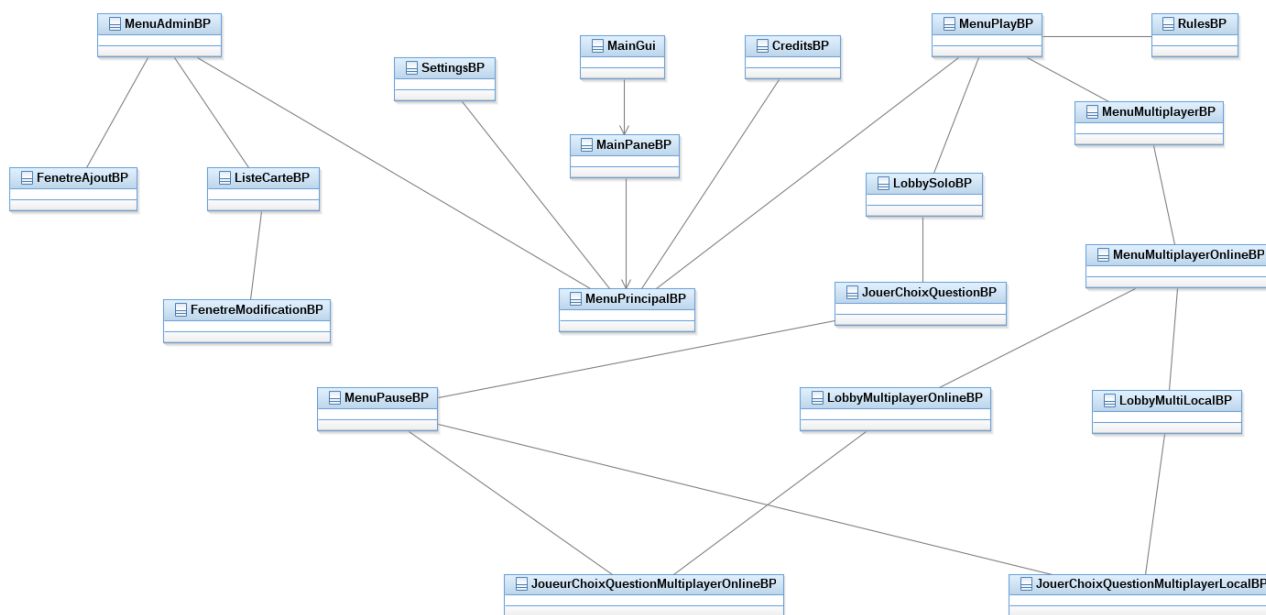


FIGURE 2 – Diagramme de la vue

Lors du lancement du programme, l'instantiation de la vue est faite de manière linéaire jusqu'à ce que la classe **MenuPrincipalBP** ne soit pas instanciée. La classe **MainGui** va charger les différents paramètres qui caractérisent la fenêtre :

- la largeur et la hauteur de la fenêtre
- la musique qui sera jouée en arrière-plan
- la langue qui sera visualisée

Une fois que les paramètres seront chargés, la classe **MainPaneBP** va commencer à instancier les différentes parties qui composent le menu principale :

- le menu principale même
- le menu pour commencer une partie
- le menu pour accéder au panneau d'administration
- le panneau des paramètres de la fenêtre
- le panneau des crédits

Une fois que ces parties seront instanciées, si l'utilisateur décide d'accéder au menu pour jouer, il se trouvera devant un menu qui lui permettra de :

- lancer une partie solo
- lancer une partie en multijoueur

Si son choix est celui de lancer une partie solo, une fois avoir inséré son pseudonyme, il sera introduit à une « lobby » ou « salle d'attente », partie du menu qui lui permettra de faire les choix suivants :

- lancer une toute nouvelle partie avec le deck qui a été chargée dès le lancement du programme
- lancer une partie en chargeant un deck de son choix

Indépendamment de son choix, il sera lancé vers **JouerChoixQuestionBP**, fenêtre dans laquelle il mènera l'entièreté de sa partie. Si son choix était de lancer une partie multijoueur, alors il aurait été introduit à un menu avec les choix suivants :

- lancer une partie multijoueur en local
- lancer une partie multijoueur en ligne

S'il décide de lancer la partie en multijoueur local, il lui sera demandé d'insérer le nombre de joueurs qui participeront à la partie. Une fois que chaque joueur a introduit son pseudonyme, tout comme pour la partie solo, il seront tous envoyés dans une « lobby » ou « salle d'attente ». Cette solution permet aux joueur de relancer une partie sans devoir réinsérer leurs respectifs pseudonymes à nouveau. Lorsque la « lobby » aura chargé, ils seront capables de lancer une nouvelle partie, lors du lancement, tous les joueurs seront renvoyés vers

**JouerChoixQuestionMultiplayerLocalBP**, classe qui laissera dérouler l'entièreté de leur partie. Dans le cas du multijoueur en ligne, l'utilisateur sera invité à choisir entre :

- héberger un serveur
- rejoindre un serveur

Si l'utilisateur décide d'héberger une partie, il lui sera demandé d'introduire le nombre de joueur qui participeront à la partie, ce dernier équivalra au nombre maximal de joueurs qui sauront rejoindre le serveur. Dans le cas où l'utilisateur décide de rejoindre un serveur, il sera invité à introduire son pseudonyme et l'adresse du serveur à rejoindre. Indépendamment s'il héberge ou rejoint un serveur, il sera introduit à une « lobby » ou « salle d'attente » dans laquelle il y aura aussi les autres joueurs lorsqu'ils rejoindront le serveur. Dans ce « lobby », les joueurs qui rejoindront seront introduits à une fenêtre qui leur permettra de quitter le serveur et interagir avec les autres joueurs à travers une « chat box », seulement celui qui héberge la partie peut activer **JouerChoixQuestionMultiplayerOnlineBP**. Indépendamment du type de partie choisi, tout utilisateur sera capable de mettre en pause la partie à travers la touche **ESCAPE**, ceci leur permettra d'accéder au **MenuPauseBP**. Ce dernier, leur permettra de quitter la partie et revenir au « lobby » ou reprendre la partie mise en pause. Ceci est tout pour la partie « jeu » de la vue.

Une autre partie de la vue que l'utilisateur sait accéder depuis **MenuPrincipalBP** est **MenuAdminBP**. Dans ce panneau, l'utilisateur sera capable d'accéder à une fenêtre qui lui permettra d'ajouter une toute nouvelle carte à son deck, ainsi qu'à une fenêtre qui lui montrera la liste de cartes qui composent son deck : **ListeCarteBP**. En cliquant sur une des cartes sur la liste, il sera renvoyé vers une autre fenêtre : **FenetreModificationBP** qui lui permettra de modifier la carte choisie s'il le désire.

Les autre fonctionnalités qui caractérisent la vue sont : **SettingsBP** et **CreditsBP**. **SettingsBP** permet à l'utilisateur de changer les paramètres qui caractérisent le jeu et la fenêtre.

**CreditsBP** lui montrera les informations sur les développeurs du jeu ainsi que les musiques utilisées et toutes les personnes qui ont contribué aux traductions.

### 3.2.3 Les exceptions

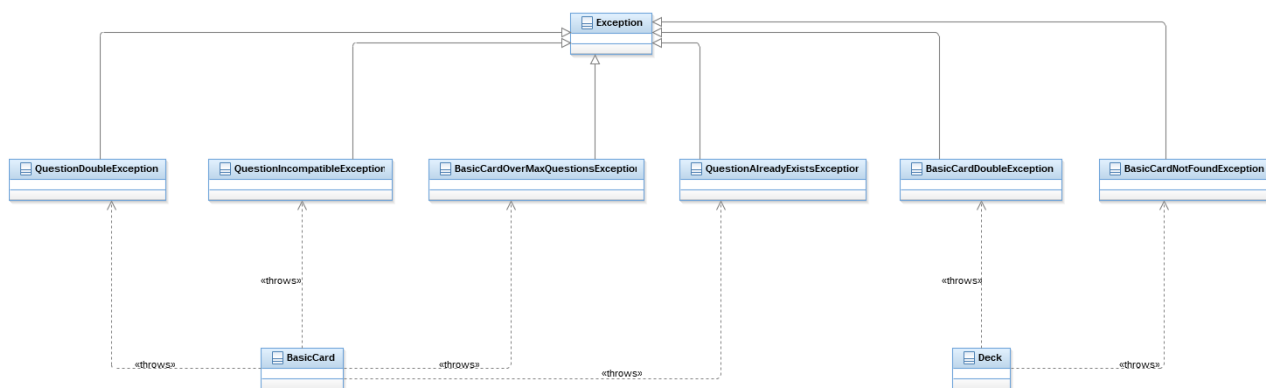


FIGURE 3 – Diagramme des exceptions



## 4 Design pattern

### 4.1 Iterator

#### 4.1.1 Explication générale

Tout d'abord, la "classe" musique présente sur le diagramme n'existe pas vraiment, mais nous avons décidé de la représenter quand même pour faciliter la compréhension. Cette "classe" représente juste le chemin d'accès, en String, à une musique. Au lancement du jeu, un objet MusicGestion va être instancié. Lors de cette instanciation, nous allons créer une liste de String, ces String sont des chemins d'accès relatif pour pouvoir accéder aux différentes musiques du jeu. Nous allons ensuite instancier un objet IteratorMusic qui prendra en paramètre cette liste. Dans la méthode gererMusic, nous allons appeler la méthode item() de la classe IteratorMusic. Cette méthode nous permet de renvoyer l'objet sur lequel nous sommes actuellement en train d'interagir. Ensuite, la méthode gererThread va être appelée, cette méthode va faire appel à la méthode next() de la classe IteratorMusic. Celle-ci va permettre d'accéder à l'objet suivant dans la liste de l'itérateur. La méthode gererMusic sera de nouveau appeler permettant ainsi d'avoir la musique suivante. Dans le cas où la liste de l'itérateur a été entièrement parcourue, une condition, présente dans la méthode item() de la classe IteratorMusic, va appeler la méthode reset() présente dans cette même classe. Cette méthode permet de revenir directement au début de la liste, pour ainsi boucler sur celle-ci.

#### 4.1.2 Diagramme de classe

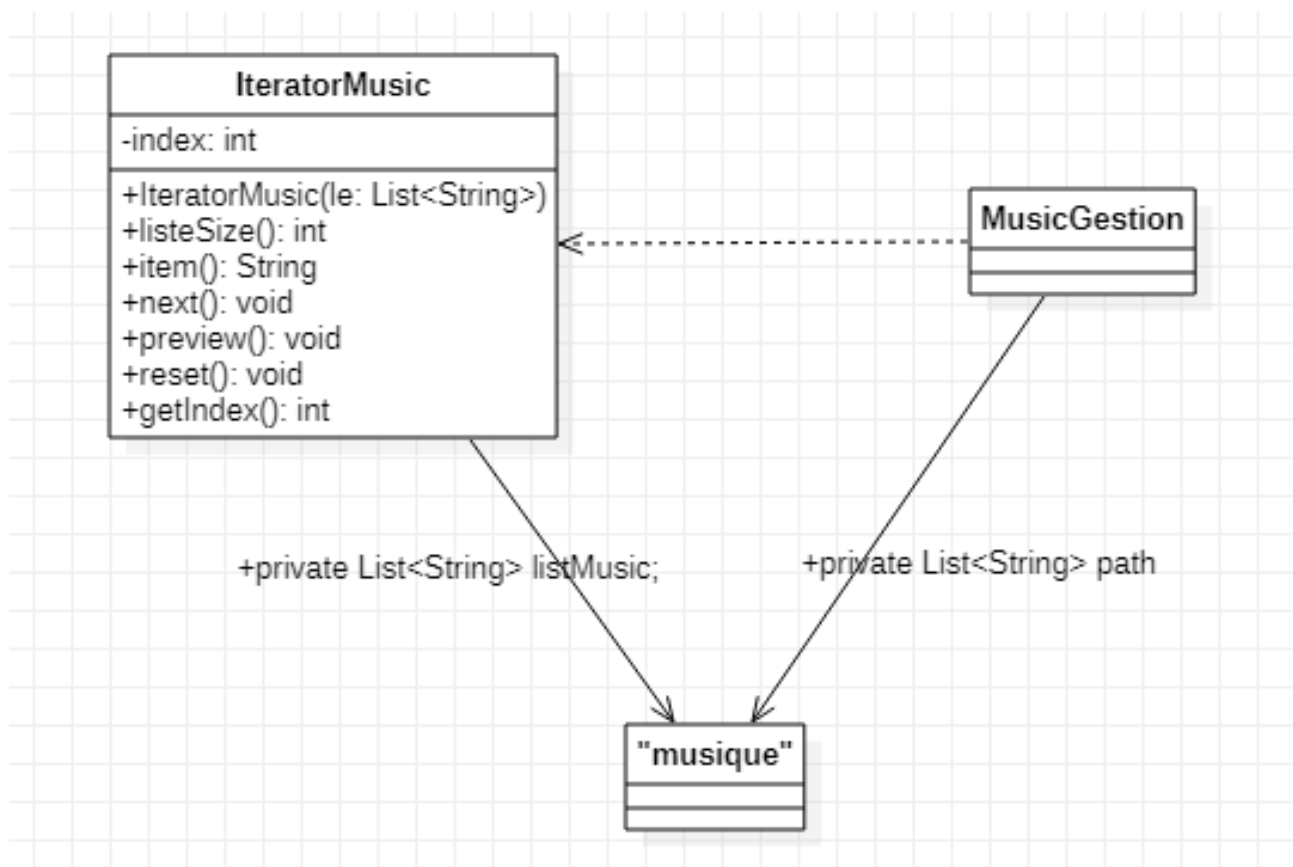


FIGURE 4 – Diagramme de l'iterator

#### 4.1.3 Extrait de code source

```

1 package be.helha.ttmc.ui.gui.util;
2

```

```
3 import java.util.List;
4
5 public class IteratorMusic
6 {
7     private List< String > listMusic;
8     private int index;
9
10    /*
11     * Constructeur de l'itérateur
12     *
13     */
14
15    public IteratorMusic( List< String > le )
16    {
17        super();
18        this.listMusic = le;
19        index = 0;
20    }
21
22    /*
23     * Méthode permettant de récupérer la taille de la liste de l'itérateur
24     */
25    public int listeSize()
26    {
27        return listMusic.size();
28    }
29
30    /*
31     * méthode permettant de renvoyer l'objet avec lequel on est entrain d'interagir
32     */
33
34    public String item()
35    {
36        // Instruction permettant de boucler et d'éviter de dépasser la taille de la liste
37        if ( index == listMusic.size() )
38        {
39            reset();
40        }
41
42        return listMusic.get( index );
43    }
44
45    /*
46     * Méthode permettant de passer à l'objet suivant
47     */
48
49    public void next()
50    {
51        index++;
52    }
53
54
55    /*
56     * Méthode permettant de revenir à l'objet précédant
57     */
58
59    public void preview()
60    {
61        index--;
62    }
63
64    /*
65     * Méthode permettant de revenir au début de la liste
66     */
67
68    public void reset()
69    {
70        index = 0;
71    }
72
73    /*
74     * Méthode permettant de récupérer la position, dans la liste, sur laquelle on est
75     */
76
77    public int getIndex()
78    {
79        return index;
80    }
81 }
```

## 4.2 Template Method

### 4.2.1 Explication générale

Notre objectif était de rendre notre programme plus accessible aux gens que ne parlent pas l'anglais. Appliquer un concept d'internationalisation était alors notre objectif. Étant donné que l'on savait que le résultat obtenu ne se baserait que sur la langue choisie par l'utilisateur, le simple fait de savoir que les manipulations seraient exactement les mêmes, on a donc décidé de mettre en place le Design Pattern : Template Method. Voici le mécanisme :

1. Le programme détecte la langue choisie par l'utilisateur, si aucune langue a été choisie, alors le programme créera une nouvelle instance de English, soit, il mettra le jeu en anglais par défaut.
2. Lors de l'instanciation des boutons ou différents messages textuels, la fonction getString qui renverra la chaîne de caractère propre à ce dernier traduite dans la langue choisie.

Si l'utilisateur choisi une langue différente, alors le programme fera une appel à la classe qui correspond à la langue. Les différents cas possibles sont :

- English() pour l'anglais
- French() pour le français
- Italian() pour l'italien
- Japanese() pour le japonais

### 4.2.2 Diagramme de classe

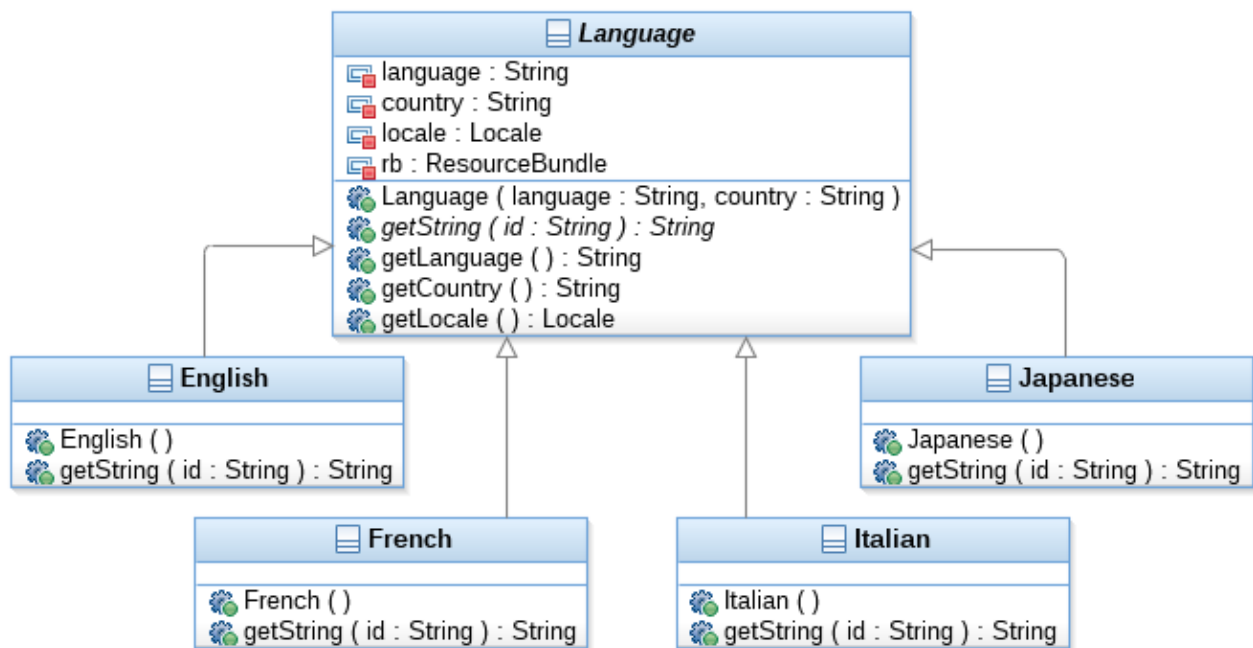


FIGURE 5 – Diagramme du template method

### 4.2.3 Extrait de code source

```

1 package be.helha.ttmc.ui.internationalization;
2
3 import java.util.Locale;
4 import java.util.ResourceBundle;
5
6 /*
7  * Classe qui sert à traduire les chaînes de caractères présentes sur l'interface graphique.
8  * Dépendamment de la langue, la sous-classe propre à la langue va représenter les
9  * chaînes de caractères adéquates sur l'interface graphique.

```

```
10  */
11  public abstract class Language
12  {
13      private String language; // La langue choisie
14      private String country; // Le pays qui parle la langue choisie
15      private Locale locale; // Le locale qui sera utilisé lors du lancement du programme
16      private ResourceBundle rb; // Le bundle à partir duquel les chaînes seront récupérées
17
18      /*
19       * Le constructeur prend en paramètre:
20       * Une chaîne de caractère propre à la langue choisie
21       * Une chaîne de caractère propre au pays qui parle la langue choisie
22       * Un "locale" sera généré à partir de la langue et du pays
23       * Le ResourceBundle propre à au locale sera initialisé
24       */
25      public Language( String language, String country )
26      {
27          this.language = language;
28          this.country = country;
29          this.locale = new Locale( language, country );
30          rb = ResourceBundle.getBundle( "be.helha.ttmc.res.strings", locale );
31      }
32
33      /*
34       * Fonction qui permettra aux sous-classes de récupérer la chaîne de caractère
35       * propre à la langue
36       */
37      public abstract String getString( String id );
38
39      /*
40       * On souhaite récupérer dans les sous-classes,
41       * le ResourceBundle instancié avec le locale correct.
42       */
43      public ResourceBundle getResourceBundle()
44      {
45          return rb;
46      }
47
48      /*
49       * Ce getter va nous permettre de récupérer la langue choisie
50       */
51      public String getLanguage()
52      {
53          return language;
54      }
55
56      /*
57       * Ce getter va nous permettre de récupérer le pays choisi
58       */
59      public String getCountry()
60      {
61          return country;
62      }
63
64      /*
65       * Ce getter va nous permettre de récupérer le locale instancié
66       */
67      public Locale getLocale()
68      {
69          return locale;
70      }
71  }
```

## 5 Veille technologique

### 5.1 Launcher

Le launcher est un logiciel à part entière qui permet à la fois de télécharger JavaFX 11 dépendamment de l'OS utilisé sur le système qui le fait tourner ainsi que de lancer l'application. Le launcher permet donc à n'importe qui de savoir lancer notre application tant qu'une version Java 11 au minimum est installé.

### 5.2 Chat en ligne

Fonctionnalité disponible depuis le jeu en multi joueur en ligne. Chat en ligne permet à plusieurs joueur de communiquer ensemble via un hôte (personne ayant décidé d'héberger la partie). Les autres joueurs devront rejoindre le jeu de l'hôte. Pour le reste cela est un chat dans ce qui a de plus commun, tous les joueurs peuvent envoyer des messages lisibles par tous les autres.

### 5.3 Traduction dynamique des menus

Depuis le menu "setting" (paramètre) il est possible de changer la langue du jeux. En la changeant ainsi qu'en redémarrant le jeu. Le jeu aura changé de langue, tous les boutons des menus seront dès lors traduit dans la langue sélectionné. Les différentes langues disponible sont

- Anglais (par défaut)
- Français
- Italien
- Japonais

## 6 Implémentation

### 6.1 Menu principal

Sur ce menu principal, nous pouvons voir cinq boutons.

- un bouton "play!" permettant d'accéder au menu jouer
- un bouton "settings" permettant d'accéder au menu paramètre
- un bouton "leave the game" permettant de quitter le jeu une fois que le joueur a confirmé son choix de fermer le jeu.
- un bouton "admin panel" permettant d'accéder au menu d'administration une fois le login réussi.
- et enfin un bouton "credits" permettant de voir les crédits du jeu.



FIGURE 6 – Menu principale du jeu

## 6.2 Règle et menu jouer

Après avoir appuyer sur "Play!", vous arrivez à une fenêtre expliquant les règles du jeu ( celle-ci n'est visible que la première fois qu'on appuie sur le bouton "Play!" par lancement de l'application). On accède au menu jouer, cette interface possède trois boutons.

- le bouton "Single Player" permettant d'atteindre après avoir mis un pseudo au menu jeu.
- le bouton "Multiplayer" permet de parvenir au menu multi joueurs.
- le bouton "Return" permet bien évidemment de retourner au menu précédant.

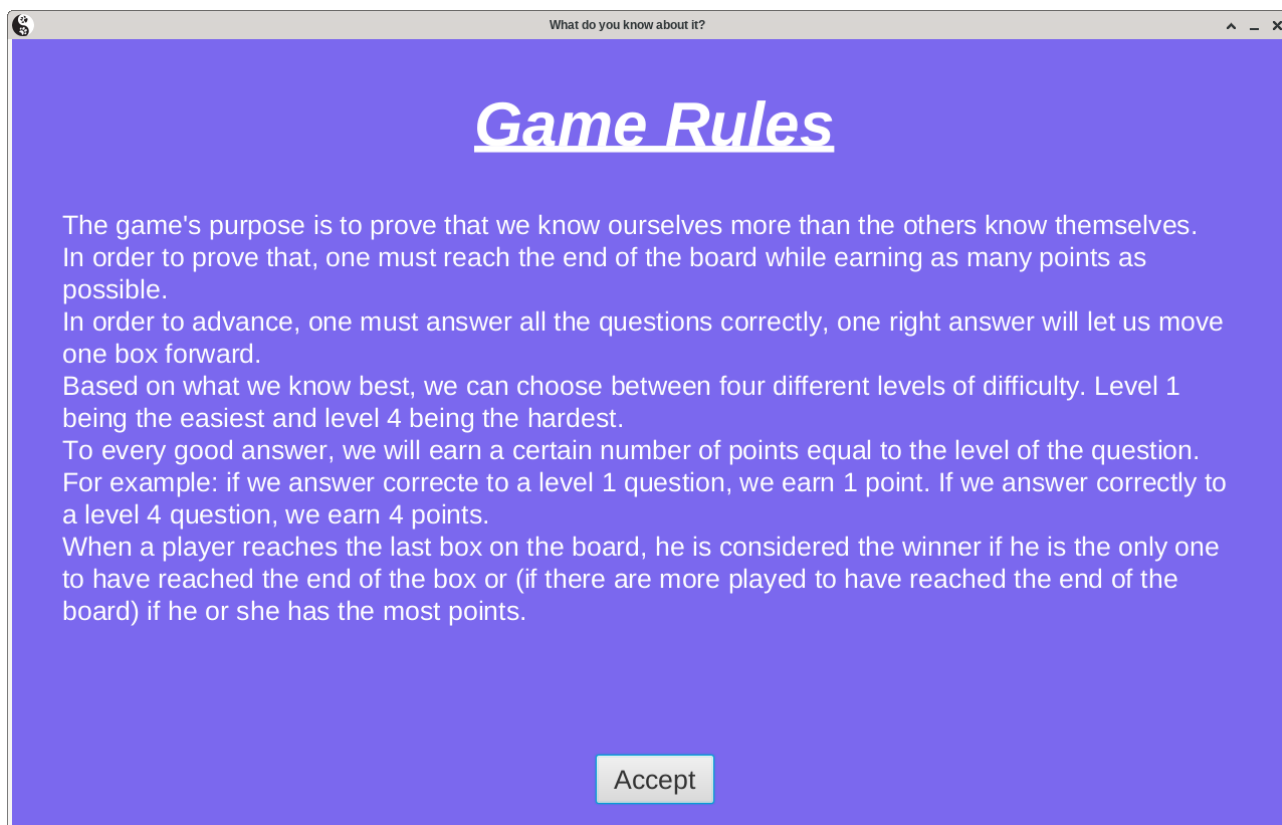


FIGURE 7 – Règle du jeu

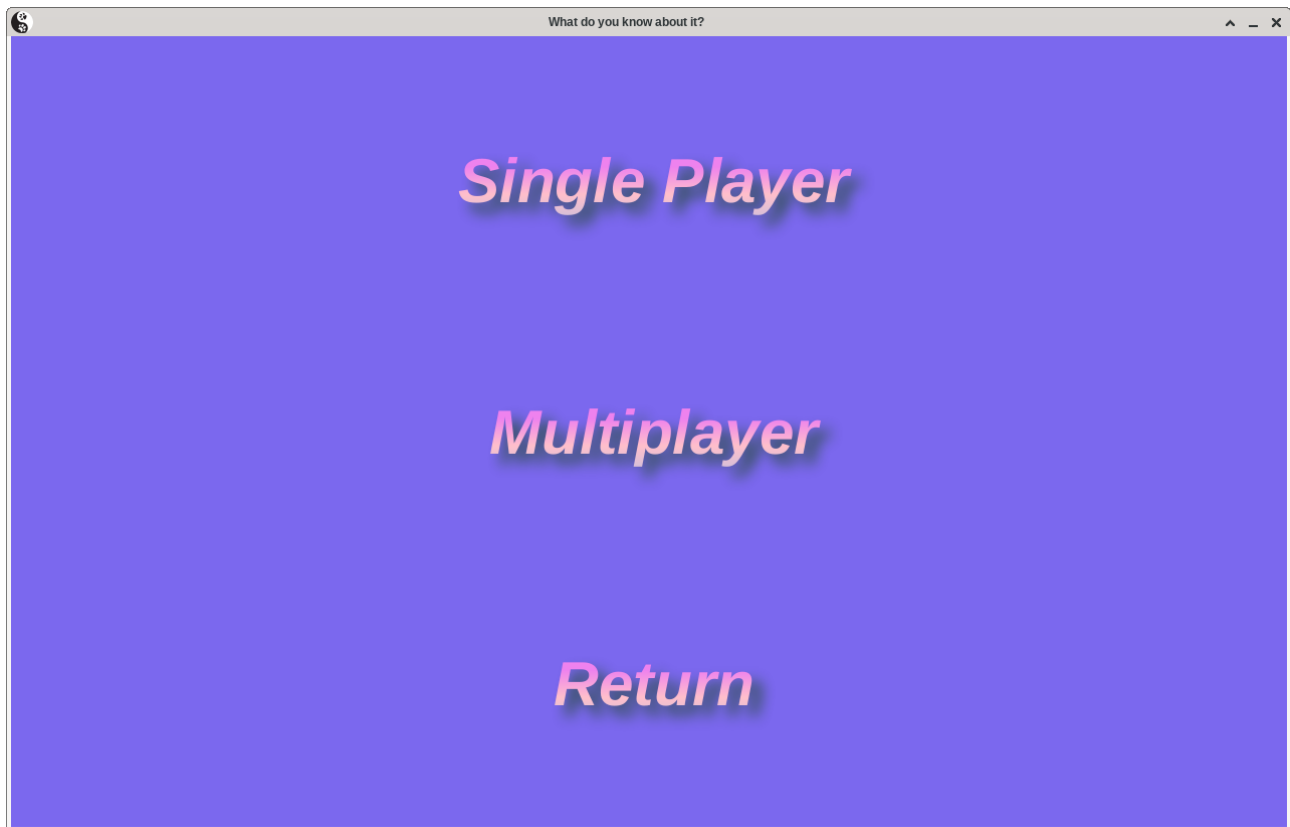


FIGURE 8 – menu jouer



### 6.3 Menu jeu

Menu accessible uniquement en partie solo, ce menu est composé de trois boutons

- Le bouton "New Game" permet de lancer une nouvelle partie.
- Le bouton "Load Game" permet de récupérer un deck (fichier json) pour l'utiliser durant la partie.
- Le bouton "Return" permet de retourner au menu précédant.

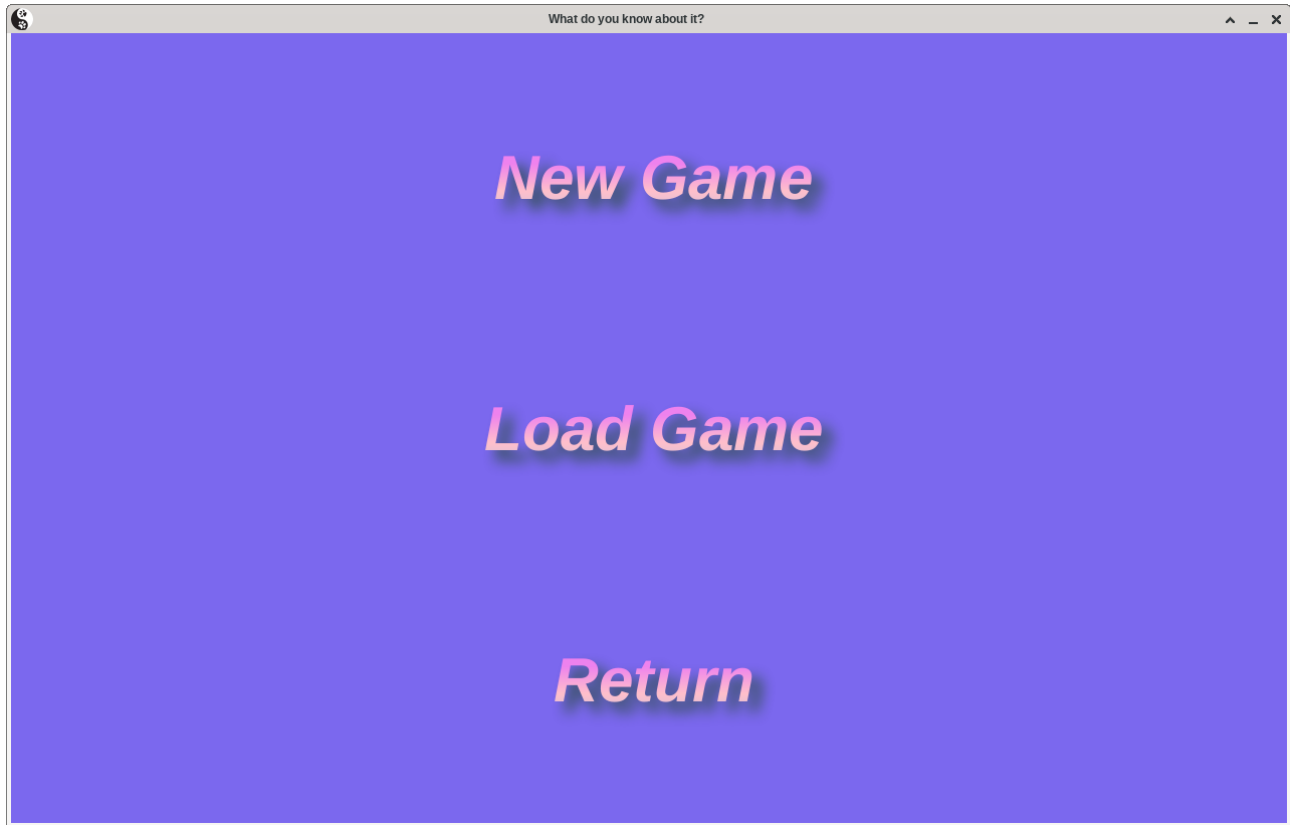


FIGURE 9 – menu jeu

## 6.4 Menu multi joueurs

Dans ce menu, se trouve toute les options disponibles pour les parties multijoueurs. Ce menu est également composé de trois boutons.

- Un bouton "Local" permet d'accéder à un multi local, il faudra ensuite renseigner le nombre de joueurs ainsi que le pseudo de chaque joueur et ensuite appuyer sur "New Game" pour lancer la partie.
- Un bouton "Online" permettant d'arriver au menu multijoueur en ligne avec des options propre à se mode de jeu.
- le bouton "Return" permet de retourner au menu précédant.

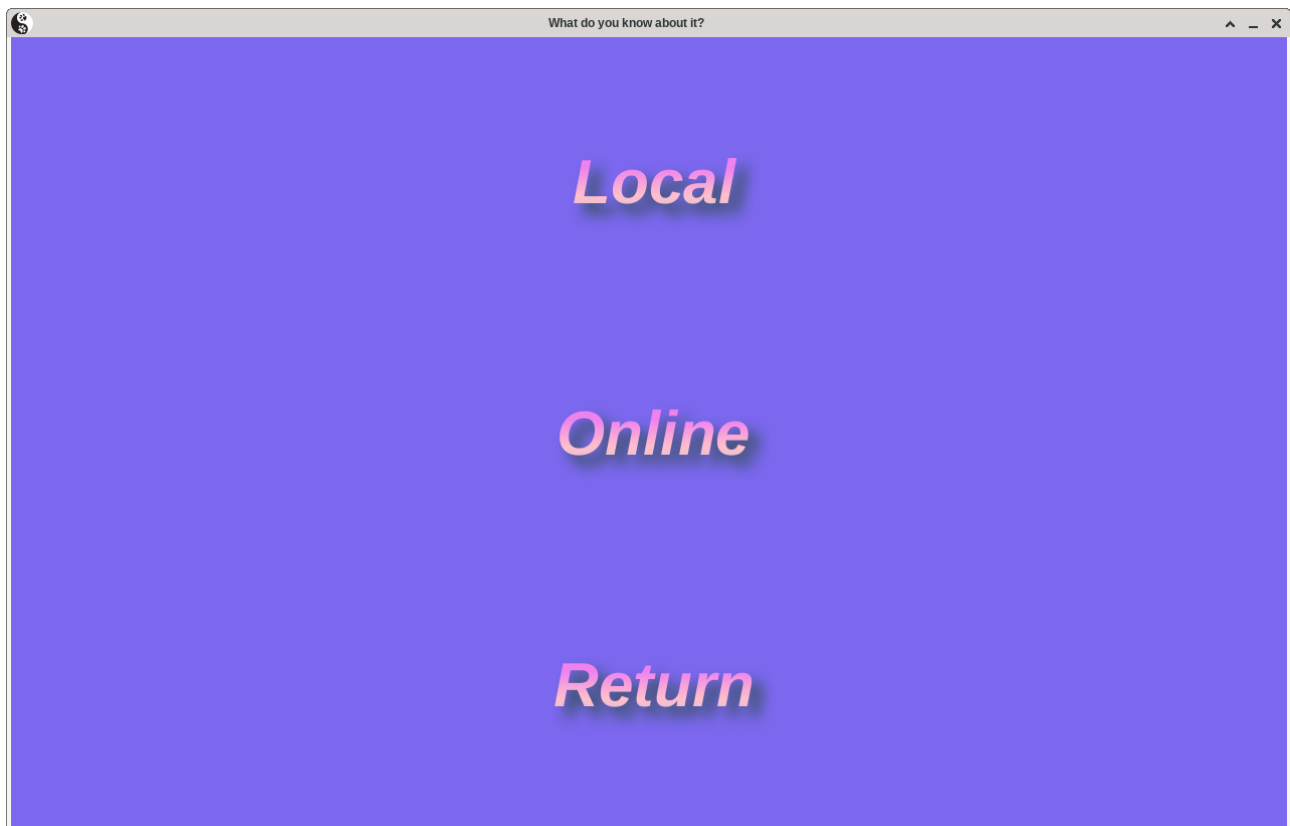


FIGURE 10 – menu multi joueurs

## 6.5 Menu multi joueur en ligne

Dans ce menu nous retrouverons les options pour le multi en ligne. Ce menu est composé de trois boutons.

- Le bouton "Host" permettant d'héberger le jeux.
- Le bouton "Join" nous offre la possibilité de rejoindre un joueur hôte. Pour ce faire, après avoir sélectionné cette option, il sera nécessaire de renseigner dans une boite de dialogue un nom d'utilisateur ainsi que l'adresse IP du joueur hôte.
- le bouton "Return" permet de retourner au menu précédent.

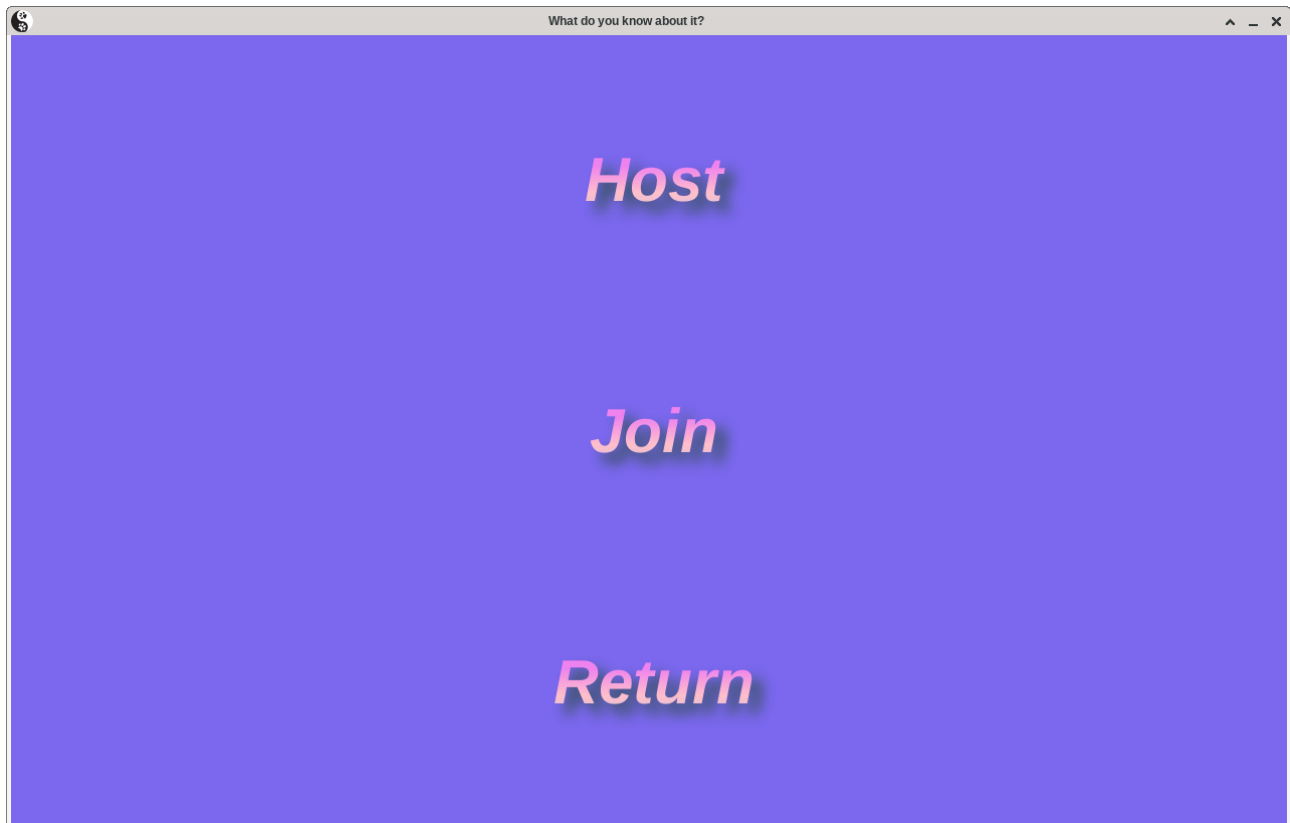


FIGURE 11 – menu multi joueur en ligne

## 6.6 Menu paramètre

Ici vous retrouverez tous les options de notre application. Commençons par la barre de volume permettant de gérer le volume du son. Cette barre est accompagné d'un bouton "Mute" permettant de complètement mettre le son en sourdine. Ensuite ce trouve une textfield permettant de modifier le chrono du jeu. Pour sélectionner une langue, cela se fait depuis une liste déroulante. Les langues disponible sont l'anglais, le français, l'italien et le japonais. Vient ensuite une deuxième liste déroulante qui permet cette fois-ci de modifier la taille de la fenêtre. Les taille disponible sont 1280x800 et 1440x900. Il est également possible de mettre le jeu en mode plein écran, pour ce faire, il faut cocher l'option "Maximize window".

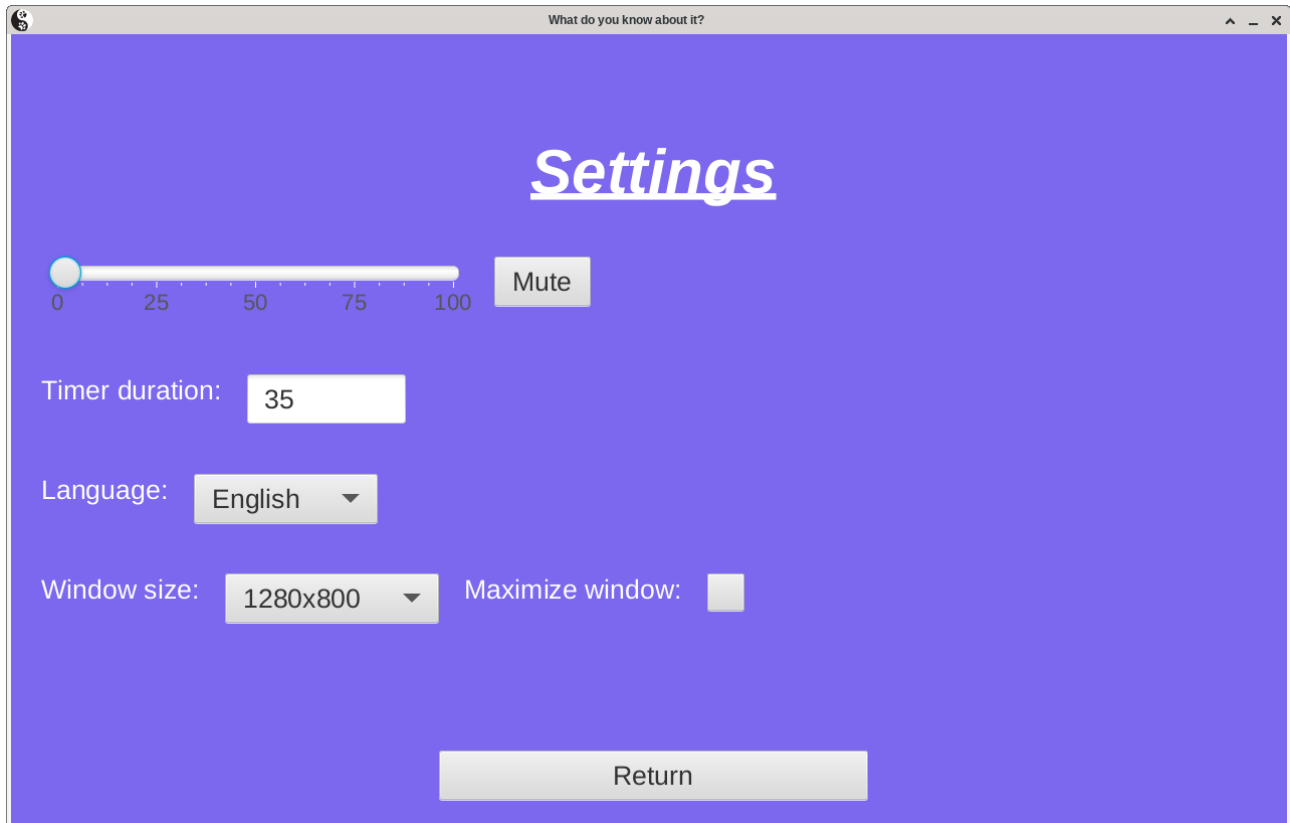


FIGURE 12 – menu paramètre

## 6.7 Menu d'administration

Après avoir cliqué sur le bouton "Admin Panel" du menu principal et vous êtes correctement connecté en tant qu'administrateur, vous accéderez au menu administrateur. ce menu est composé de cinq boutons.

- Le bouton "Add a new card" permet d'arriver à une nouvelle interface permettant de créer une nouvelle carte.
- Le bouton "List of cards" permet d'accéder au tableau de toutes les cartes du deck. Il est également possible de modifier une carte depuis ce tableau
- Le bouton "Import a new deck" permet d'importer un nouveau deck pour le jeu. Ce bouton ouvre un explorateur de fichier afin de pouvoir trouver la cible à importer.
- Le bouton "Export the current deck" permet d'exporter le deck actuellement utiliser par le jeu. Ce bouton ouvre également un explorateur de fichier afin de pouvoir nommer et placer où on le désire le deck.
- le bouton "Return" permet de retourner au menu précédant.

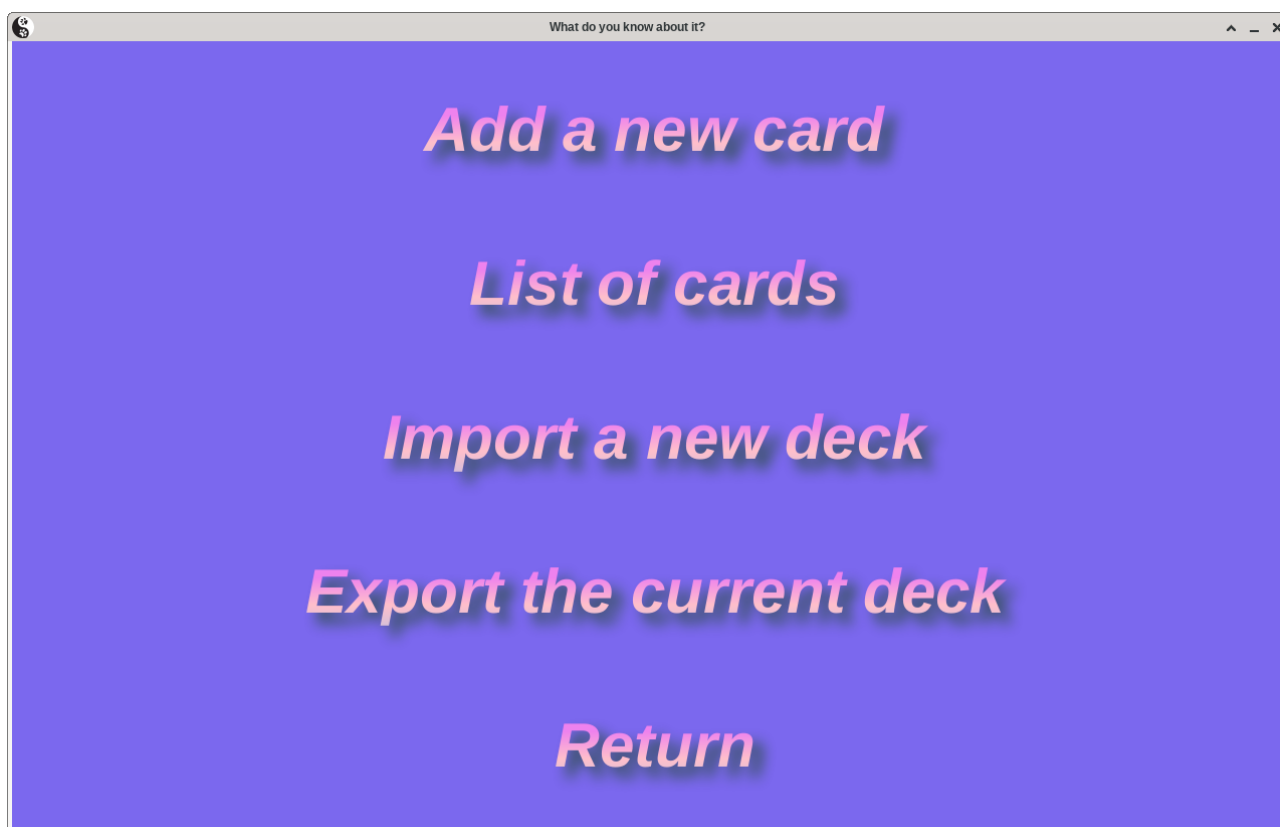


FIGURE 13 – Menu d' administration

## 6.8 interface "New Card"

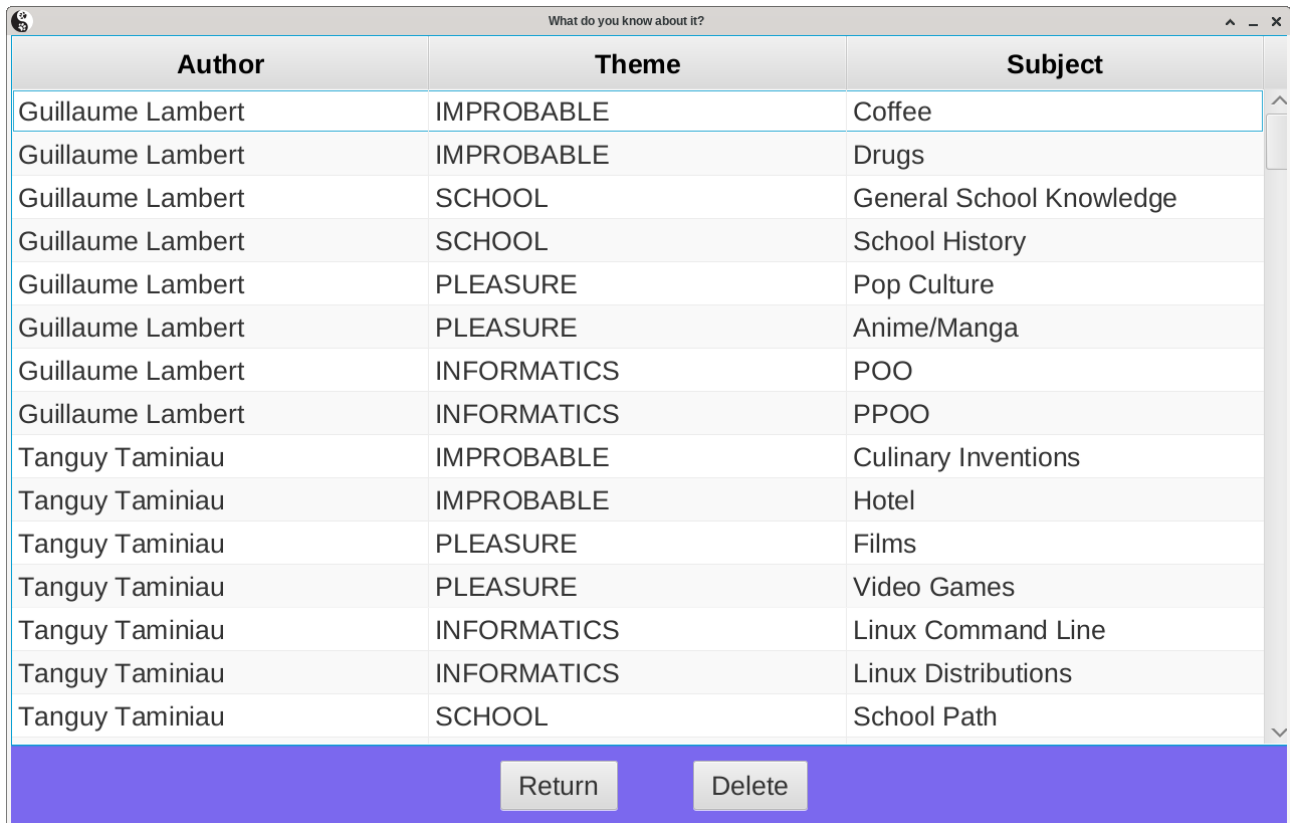
Cette interface permet d'ajouter une carte au deck. Pour ce faire il faudra d'abord choisir un thème via la liste déroulante. Il faudra ensuite renseigner l'auteur, le sujet, les quatre challenges ainsi que les quatre réponses. Sachant que le challenge n°1 est le plus facile et donc le challenge 4 est le plus dur. Le bouton "Add" ajoutera la carte ainsi créée si toutes les informations sont bien renseignées. Le bouton "Clean" permet de vider tous les textfields.

The screenshot shows a web browser window with the title "What do you know about it?". The main content area has a purple background. At the top left, there is a "Theme :" label followed by a dropdown menu currently showing "IMPROBABLE". To the right of the dropdown is an "Author :" label followed by a text input field. Below these is a "Subject :" label followed by a text input field. In the center, there is a "Challenge :" label. To its right, there is an "Answers :" label. Below the "Challenge :" label, there are four rows, each starting with a number (1, 2, 3, 4) followed by a colon and a text input field. To the right of each challenge input field is another text input field for the corresponding answer. At the bottom of the form, there are three buttons: "Return", "Clean", and "Add".

FIGURE 14 – interface "New card"

## 6.9 Tableau carte

Cette interface est composée d'un grand tableau de trois colonnes ( auteur, thème et sujet) ainsi que de trois boutons. Un bouton "return", un bouton "Reload" permettant de recharger les informations modifier et un bouton delete permet à partir d'une sélection de carte de supprimer la carte sélectionné. En double cliquant sur une ligne , nous accédons à une interface similaire à "Menu d'administration" si ce n'est que toutes les textfields sont déjà rempli. Il ne reste plus qu'à faire les modifications puis appuyer sur " Modify" puis sur " reload" et toutes les informations ont été mises à jour.



Author	Theme	Subject
Guillaume Lambert	IMPROBABLE	Coffee
Guillaume Lambert	IMPROBABLE	Drugs
Guillaume Lambert	SCHOOL	General School Knowledge
Guillaume Lambert	SCHOOL	School History
Guillaume Lambert	PLEASURE	Pop Culture
Guillaume Lambert	PLEASURE	Anime/Manga
Guillaume Lambert	INFORMATICS	POO
Guillaume Lambert	INFORMATICS	PPOO
Tanguy Taminiau	IMPROBABLE	Culinary Inventions
Tanguy Taminiau	IMPROBABLE	Hotel
Tanguy Taminiau	PLEASURE	Films
Tanguy Taminiau	PLEASURE	Video Games
Tanguy Taminiau	INFORMATICS	Linux Command Line
Tanguy Taminiau	INFORMATICS	Linux Distributions
Tanguy Taminiau	SCHOOL	School Path

Return Delete

FIGURE 15 – Tableau carte

## 7 Tests unitaires

Le code présent ci-dessous est celui de la fonction qui permet de rajouter une nouvelle question au deck. Différent cas de vérification ont lieu avant de permettre à une question quelconque d'être rajoutée. Le premier cas sert à vérifier que l'objet en entrée ne soit pas null, si c'est le cas, alors la fonction renverra false. Le deuxième consiste à vérifier que l'ajout ne cause pas de doublon, si c'est le cas, l'exception `QuestionDoubleException` sera appelée, et false sera renvoyé comme résultat. Le troisième cas consiste à vérifier que le thème de la question correspond bien à celui de la carte, si c'est pas le cas, `QuestionIncompatibleException` sera appelée et false sera renvoyé comme résultat. Le dernier cas de vérification consiste à vérifier qu'il n'y ait pas plus que le nombre maximal de question sur une carte, dans le cas de notre jeu ce dernier équivaut à 4. Si une 5<sup>ème</sup> est rajoutée, l'exception `BasicCardOverMaxQuestionsException` sera appelée et le résultat sera false; Si tous les cas de vérifications sont réussis, alors la fonction va ajouter la nouvelle question à la carte, et le résultat sera true.

```

1  /**
2   * Function used to add a new question to the card
3   *
4   * @param q The question that the user wishes to add to his card
5   *
6   * @return true if added, false if not
7   */
8  public boolean add( Question q )
9  {
10     try
11     {
12         if ( q == null )
13         {
14             throw new NullPointerException();
15         }
16         if ( questions.contains( q ) )
17         {
18             throw new QuestionDoubleException();
19         }
20         if ( q.getTheme() != theme || !q.getAuthor().equalsIgnoreCase( author )
21             || !q.getSubject().equalsIgnoreCase( subject ) )
22         {
23             throw new QuestionIncompatibleException();
24         }
25         if ( questions.size() == 4 )
26         {
27             throw new BasicCardOverMaxQuestionsException();
28         }
29         return questions.add( q.clone() );
30     }
31     catch ( NullPointerException npe )
32     {
33         npe.printStackTrace();
34         return false;
35     }
36     catch ( QuestionDoubleException qde )
37     {
38         qde.printStackTrace();
39         return false;
40     }
41     catch ( QuestionIncompatibleException qie )
42     {
43         qie.printStackTrace();
44         return false;
45     }
46     catch ( BasicCardOverMaxQuestionsException bcomqe )
47     {
48         bcomqe.printStackTrace();
49         return false;
50     }
51 }

```

Dans les tests unitaires, cette fonction est vérifiée en lui injectant les différents cas possibles. Voici une partie du code des tests unitaires qui ont lieu sur `BasicCard`, ici on ne vérifie que les différentes interactions avec la fonction "add" et on s'assure que les différents cas de vérifications soient bien respectés.

```

1  public class BasicCardTests
2  {
3      private static BasicCard c1, c2;
4      private static Question q1, q2, q3, q4, q5, q6, q7, q8, q9, q10, q11, q12;
5
6      @BeforeAll
7      static void initAll()
8      {
9          c1 = new BasicCard( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms" );

```



```

10     c2 = c1.clone();
11     q1 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does RAM stand for?",
12         "Random Access Memory" );
13     q2 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does JAR stand for?",
14         "Java ARchive" );
15     q3 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does WWW stand for?",
16         "World Wide Web" );
17     q4 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does CPU stand for?",
18         "Central Processing Unit" );
19     q5 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does GPU stand for?",
20         "Graphics Processing Unit" );
21     q6 = new Question( "Giorgio Lambert", Theme.INFORMATICS, "Acronyms", "What does IT stand for?",
22         "Information Technology" );
23     q7 = new Question( "Giorgio Caculli", Theme.IMPROBABLE, "Acronyms", "What does IT stand for?",
24         "Information Technology" );
25     q8 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronym", "What does IT stand for?",
26         "Information Technology" );
27     q9 = q5.clone();
28     q10 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does JDK stand for?",
29         "Java Development Kit" );
30     q11 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does OOP stand for?",
31         "Object Oriented Programming" );
32     q12 = new Question( "Giorgio Caculli", Theme.INFORMATICS, "Acronyms", "What does OS stand for?",
33         "Operating System" );
34 }
35
36 @BeforeEach
37 void init()
38 {
39 }
40
41 @Test
42 public void testAddQuestions()
43 {
44     assertTrue( () -> c1.add( q1 ), "failure - the question was not added" );
45     assertTrue( () -> c1.add( q2 ), "failure - the question was not added" );
46     assertTrue( () -> c1.add( q3 ), "failure - the question was not added" );
47     assertTrue( () -> c1.add( q4 ), "failure - the question was not added" );
48 }
49
50 @Test
51 public void testAddDouble()
52 {
53     assertFalse( () -> c1.add( q1 ), "failure - the question was added" );
54 }
55
56 @Test
57 public void testAddMoreThanFourQuestions()
58 {
59     assertTrue( () -> c1.add( q11 ), "failure - the question was not added" );
60     assertFalse( () -> c1.add( q12 ), "failure - the question was added" );
61 }
62
63 @Test
64 public void testAddQuestionWithDifferentAuthor()
65 {
66     assertFalse( () -> c1.add( q6 ), "failure - the question was added" );
67 }
68
69 @Test
70 public void testAddQuestionWithDifferentTheme()
71 {
72     assertFalse( () -> c1.add( q7 ), "failure - the question was added" );
73 }
74
75 @Test
76 public void testAddQuestionWithDifferentSubject()
77 {
78     assertFalse( () -> c1.add( q8 ), "failure - the question was added" );
79 }
80
81 @Test
82 public void testAddNullQuestion()
83 {
84     assertFalse( () -> c1.add( null ), "failure - the question was added" );
85 }
86 }

```

Voici le rapport de couverture des différents tests menés sur les différentes fonctions de BasicCard












Element	Coverage	Covered Instructions	Missed Instructions
▼ BasicCard.java	 88.9 %	265	33
▼ BasicCard	 88.9 %	265	33
BasicCard(String, Theme, String)	 100.0 %	17	0
add(Question)	 100.0 %	71	0
clone()	 69.0 %	20	9
equals(Object)	 82.6 %	19	4
getAuthor()	100.0 %	3	0
getQuestions()	 100.0 %	23	0
getSubject()	100.0 %	3	0
getTheme()	100.0 %	3	0
modify(Question, Question)	 100.0 %	37	0
remove(int)	 80.0 %	20	5
remove(Question)	 77.8 %	7	2
toString()	 76.4 %	42	13

FIGURE 16 – Rapport de couverture du code

## 8 Conduite du projet

### 8.1 Tableau Trello sprint 1

Lors du premier sprint (le 8 mars 2021), la visualisation de carte ainsi que la visualisation de carte avec contrainte était en cours alors que le bouton "play!", la sélection du niveau de questions, le bouton "exit", une alerte de confirmation de la réponse ainsi que le menu pour ajouter une carte était déjà prête. Il restait encore énormément de travail.

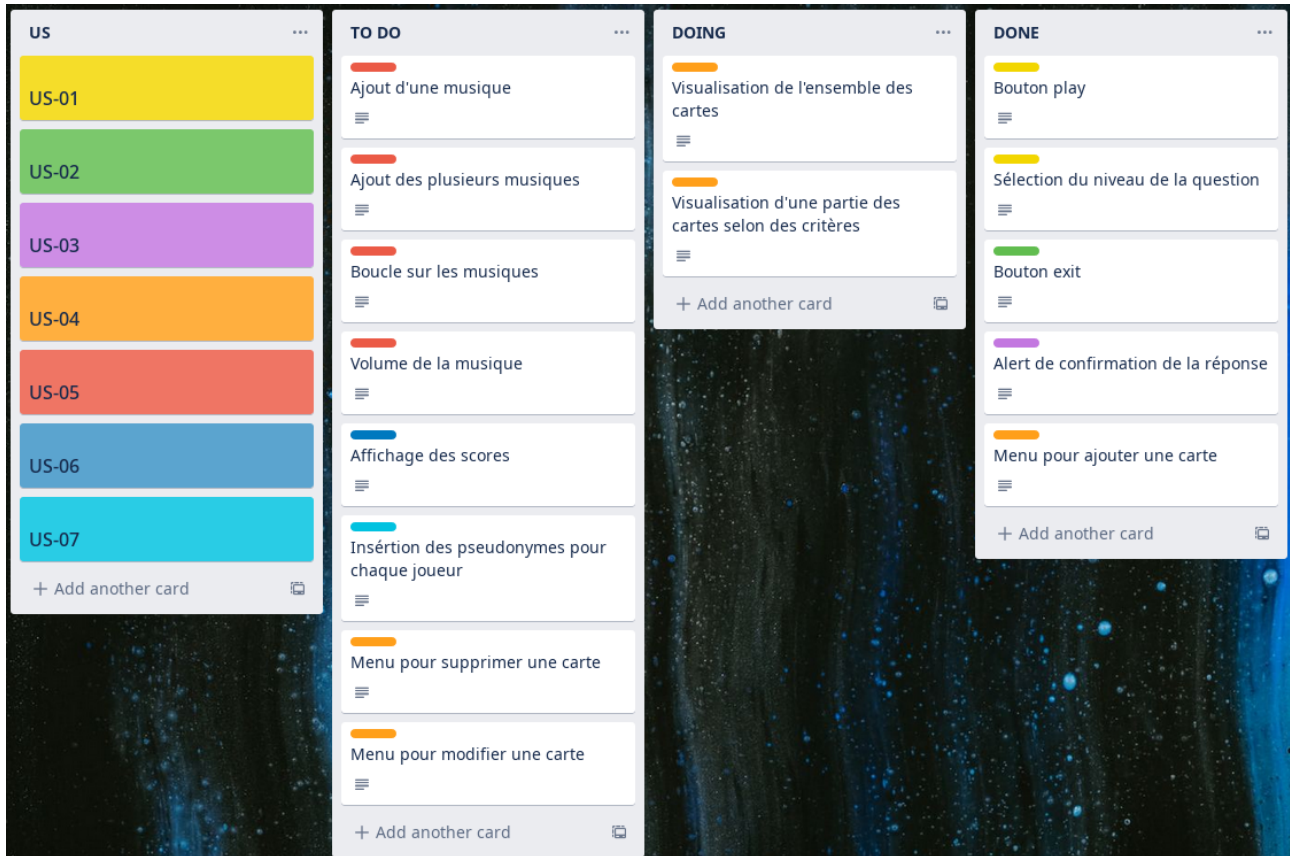


FIGURE 17 – Tableau Trello sprint 1

## 8.2 Tableau Trello sprint 2

Lors du deuxième sprint (le 29 mars 2021), la fenêtre des paramètres ainsi que ses premiers composants (volume musique, le bouton "mute") était en cours de codage tandis que l'affichage des scores, les menus de modification et de suppression de carte, l'ajout de la musique, la playlist musicale ainsi que l'utilisation de pseudo encoder par l'utilisateur avait été terminé. Le projet avançait à bon pas et la base du projet était quasiment finie.

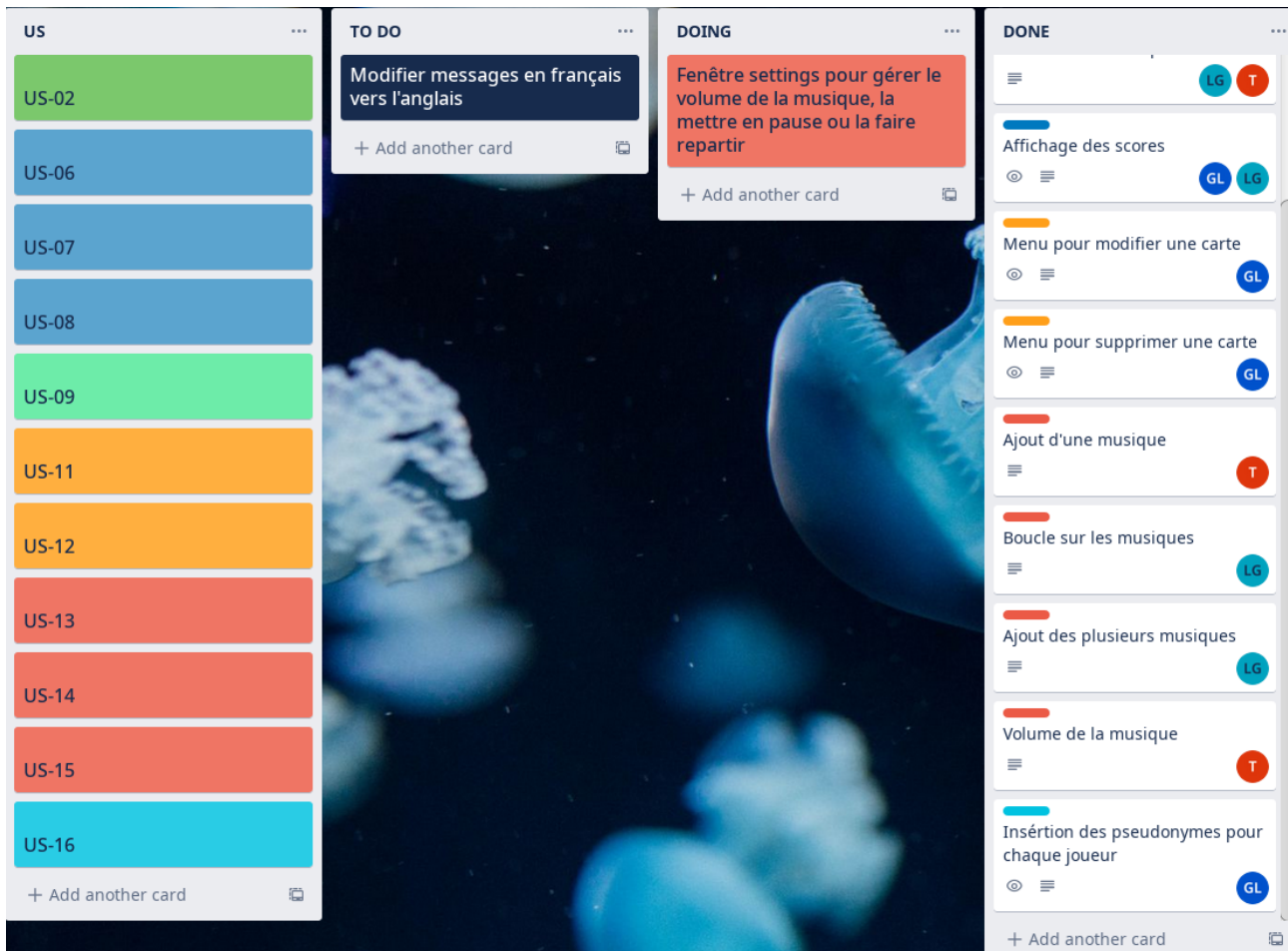


FIGURE 18 – Tableau Trello sprint 2

### 8.3 Tableau Trello sprint 3

Concernant le troisième sprint (le 26 avril 2021), il ne reste plus que la mise en place de la partie multijoueur en ligne. La fenêtre des paramètres ainsi que la création d'un menu pause ont été terminés. Le plateau de jeu ainsi que les pions ont également été fait durant ce sprint. Bien que le mode multijoueur en ligne ne soit pas terminé, les possibilités d'héberger ou de rejoindre une partie ainsi qu'un chat ont été terminés.

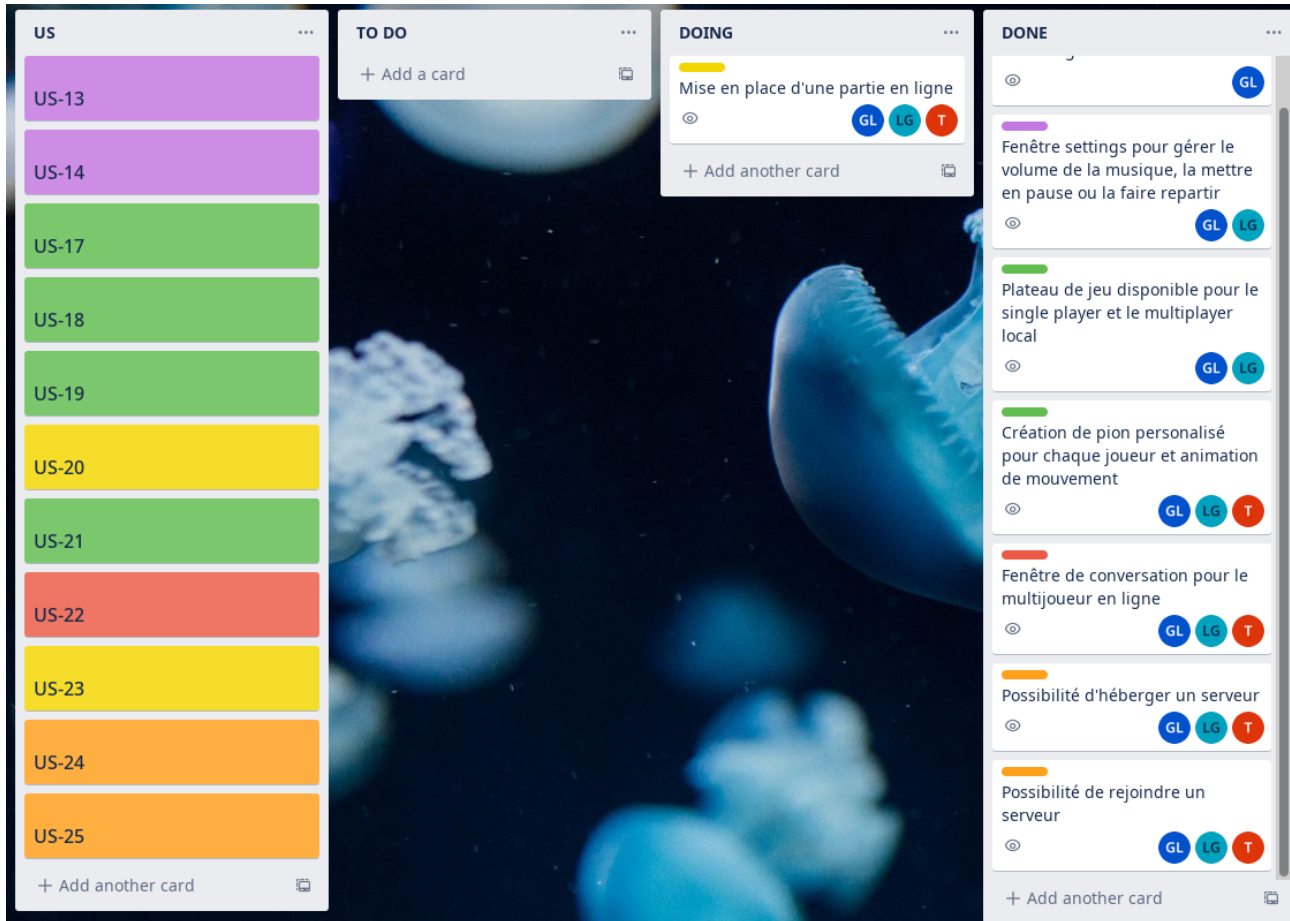


FIGURE 19 – Tableau Trello sprint 3

## 9 Conclusion

Ce projet nous permet de nous rendre compte de ce qu'est un projet de plus grande envergure que ce dont on a l'habitude. Cela nous a aussi permis de nous familiariser avec une méthode de gestion de projet, qu'est la méthode SCRUM. De plus, nous avons appris à bien communiquer et s'organiser entre collègues/coéquipiers. Ce projet nous a permis aussi d'apprendre à combiner ce que nous apprenons en cours avec de la recherche de documentation sur internet. Bien que parfois nous avons eu des problèmes de gestion de temps, notamment au niveau de la documentation, nous avons réussi quand-même à tenir nos timings. En finalité, nous sommes contents du résultat obtenu actuellement, cependant, nous pensons le continuer et le pauffer pour obtenir le résultat que nous avons imaginé. Nous tenons à remercier ceux qui nous ont aidés pour la traduction, et ceux qui nous ont donné leurs avis par rapport au design du jeu. Et nous tenons à remercier particulièrement Mr. Altares pour ses bons conseils, Mme. Crespin pour l'aide sur la traduction et ses conseils sur les présentations, Mr. Godefroid pour nous avoir facilité la compréhension et l'apprentissage des Design Pattern.

## Glossaire

**JDK** Java Development Kit. 2, 3

**POO** Programmation Orienté Objet. 2

**TTMC** Tu te mets combien?. 2

**VM** Virtual Machine. 2