

What do you know about it ?

Giorgio Caculli LA196672, Guillaume Lambert LA198116, Tanguy Taminiau LA199566
Groupe B05

2 mai 2021

Table des matières

Introduction	2
1 Présentation du sujet	2
1.1 Règle du jeu	2
1.2 Position du problème	2
2 Description général de l'application	2
3 Analyse	3
3.1 Product backlog	3
3.2 Les classes	4
3.2.1 Le modèle	4
3.2.2 La vue	4
3.2.3 Les exceptions	4
4 Design pattern	5
4.1 Iterator	5
4.1.1 Explication générale	5
4.1.2 Diagramme de classe	5
4.1.3 Extrait de code source	5
4.2 Template Method	7
4.2.1 Explication générale	7
4.2.2 Diagramme de classe	7
4.2.3 Extrait de code source	7
5 Veille technologique	9
5.1 Launcher	9
5.2 Chat en ligne	9
5.3 Traduction dynamique des menus	9
6 Implémentation	10
6.1 Menu principal	10
6.2 Règle et menu jouer	11
6.3 Menu jeu	11
6.4 Menu multi joueurs	11
6.5 Menu multi joueur en ligne	11
6.6 Menu paramètre	11
6.7 Menu d'administration	11
6.8 interface "New Card"	12
6.9 Tableau carte	12
7 Tests unitaires	13
8 Conduite du projet	15
9 Conclusion	15

Introduction

Dans le cadre du cours de “Projets” de l’UE 210, nous allons devoir 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 vu au cours de POO (programmation orienté objet). De plus la grande liberté accordée à ce projet consent et oblige l’élève à apprendre à se documenter et savoir faire des recherches. L’objectif du projet est simple, créer un jeu sur le principe de TTMC (Tu Te mets Combien?).

1 Présentation du sujet

nous allons devoir créer un jeu similaire à TTMC. Il s’agit d’un jeu comportant des cartes comportant des questions. Il existe quatre thèmes repérables 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 questions 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-mêmes. Pour le montrer, il faut arriver au bout du plateau tout en gagnant des points. Pour avancer, il faut répondre correctement aux questions qui se poseront, une bonne réponse nous fait avancer d’une case. Nous pouvons choisir entre quatre difficultés de questions, en fonction de ce que nous connaissons le mieux. Le niveau 1 est le plus facile et le niveau 4 le plus difficile. À chaque bonne réponse, nous gagnons un nombre de points équivalant au niveau de la question. Par exemple : nous répondons correctement à une question de niveau 1, nous gagnons 1 point. Si nous répondons correctement à une question de niveau 4, nous gagnons 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 allons structurer notre projet mais aussi imaginer le jeu autant dans sa présentation graphique que dans les fonctionnalités à incorporer. Les fonctionnalités du jeu nous ont demandé beaucoup de réflexions surtout que nous avions de grande ambition.

2 Description général de l’application

Voici l’histoire d’un nain capable de courir vite et de voyager loin. Dans son épiphanie formidable nous le suivrons une bière à la main.

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

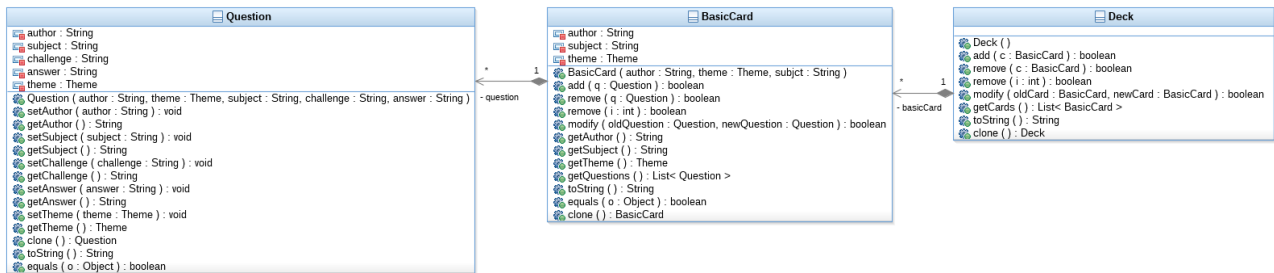


FIGURE 1 – Diagramme du modèle

3.2.2 La vue

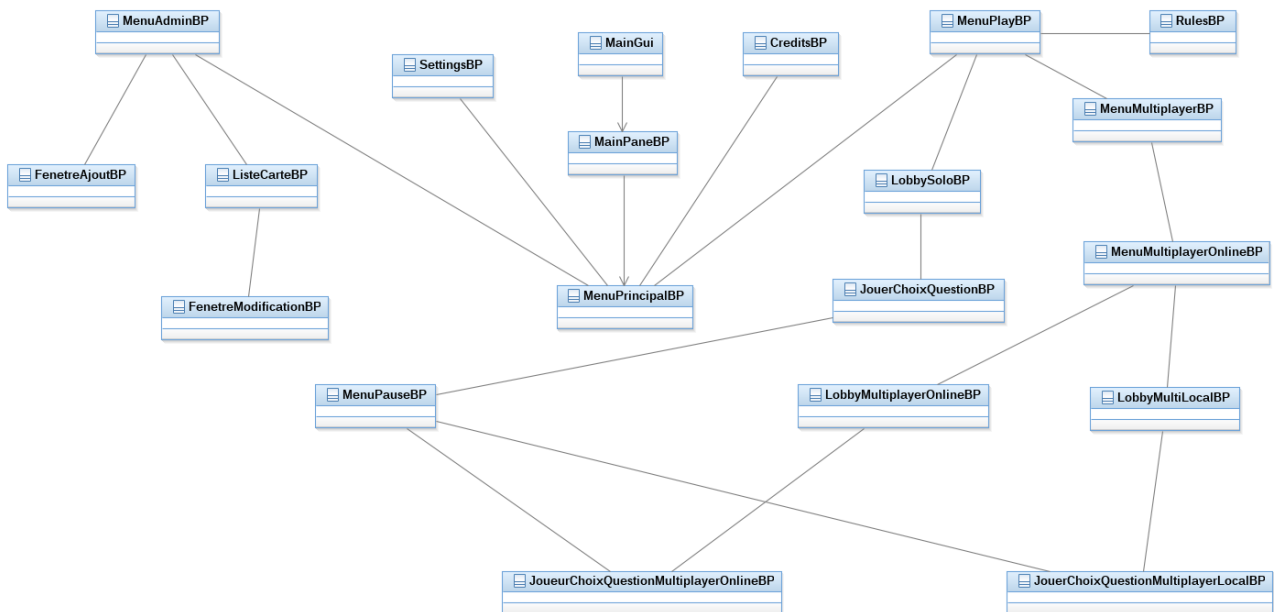


FIGURE 2 – Diagramme de la vue

3.2.3 Les exceptions

4 Design pattern

4.1 Iterator

4.1.1 Explication générale

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

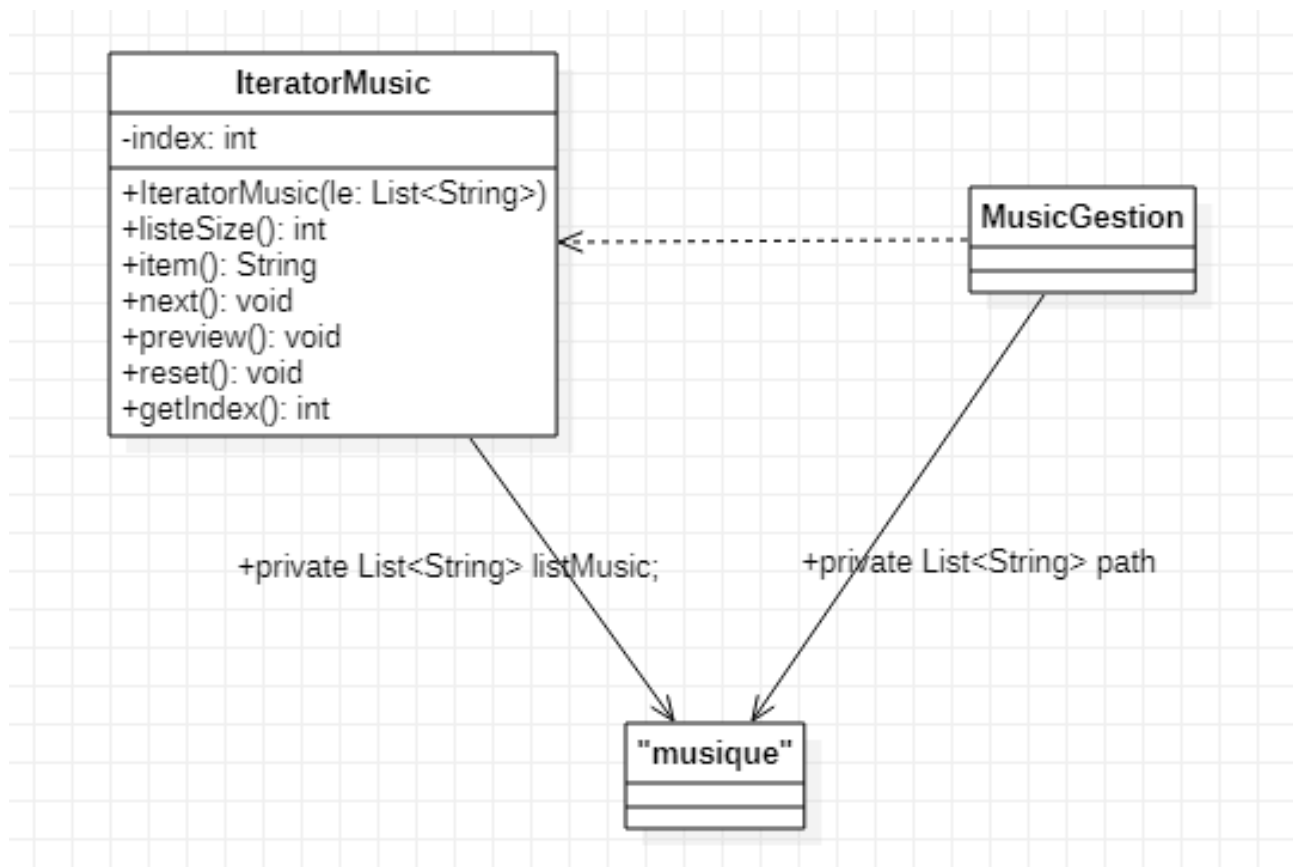


FIGURE 3 – Diagramme de l'iterator

4.1.3 Extrait de code source

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.

```

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 choisit une langue différente, alors le programme fera un 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

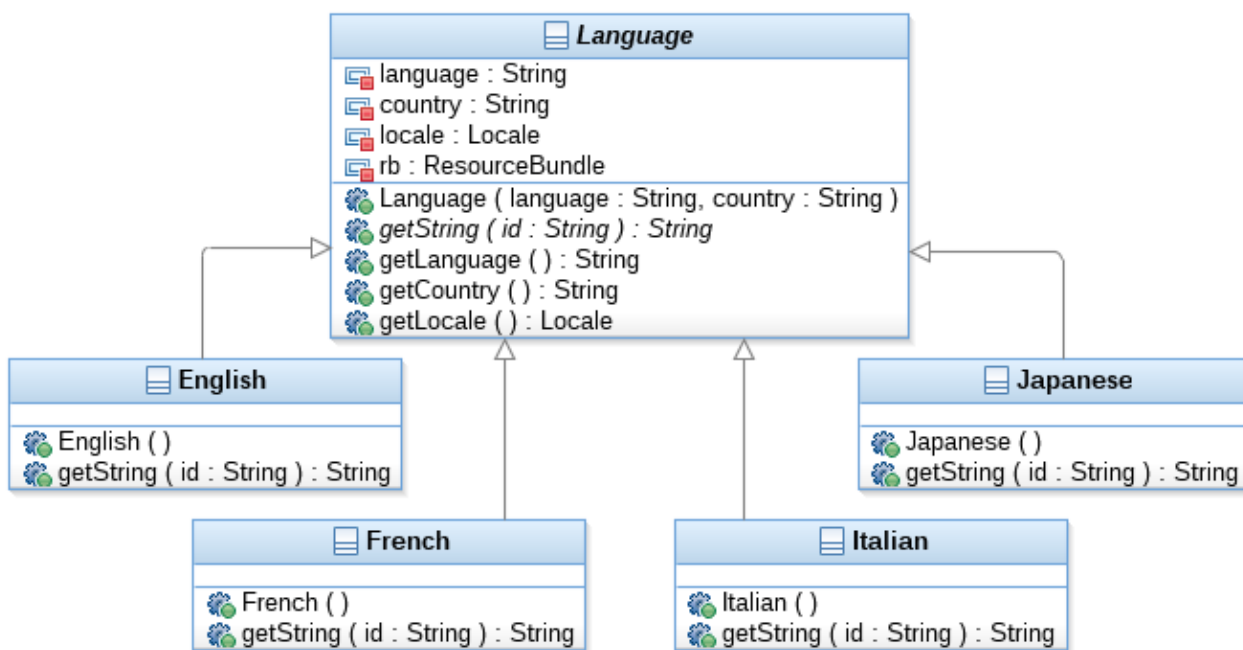


FIGURE 4 – Diagramme du template method

4.2.3 Extrait de code source

Voici l'histoire d'un nain capable de courir vite et de voyager loin. Dans son épopée formidable nous le suivrons une bière à la main.

```

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 dépendemment 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 décider d'héberger la partie). Les autres joueur 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 lisible 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 changer de langue, tous les boutons des menu 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.

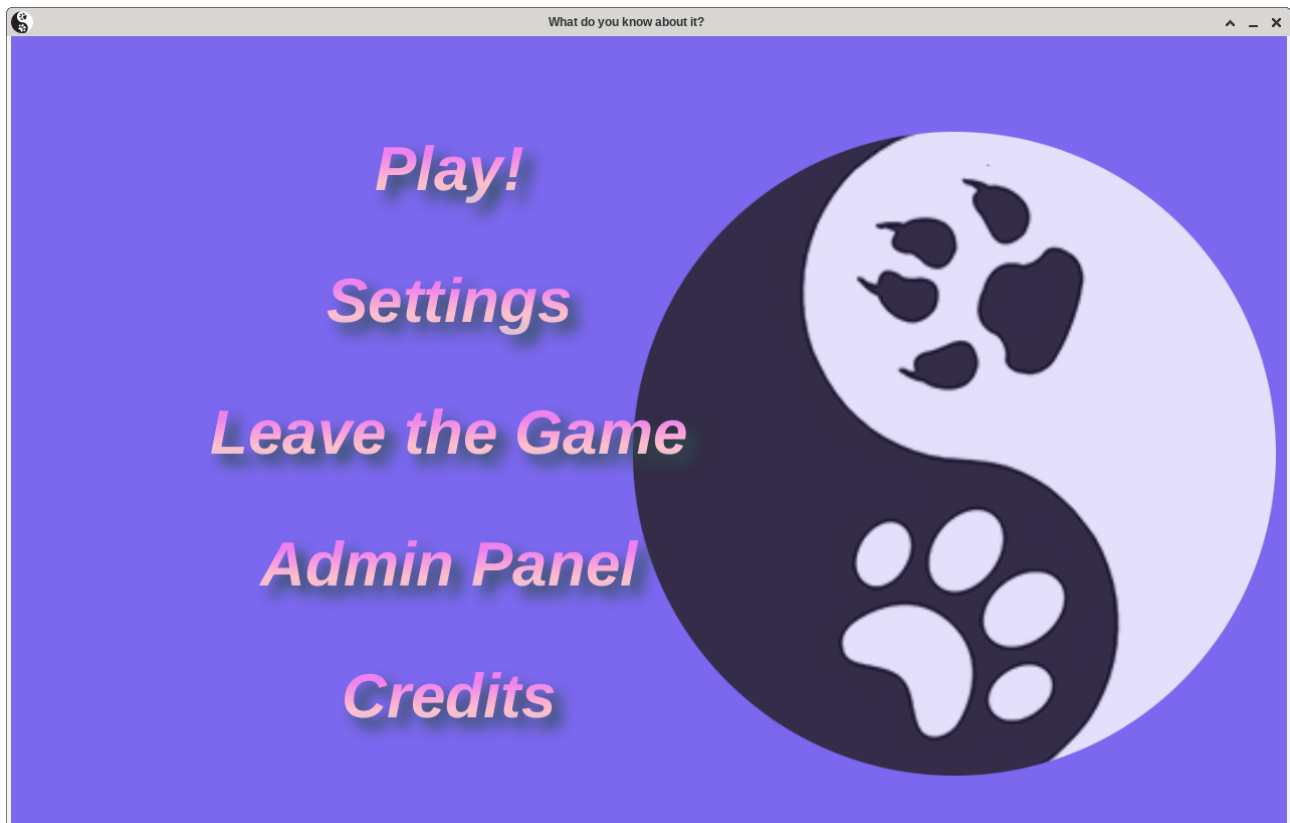


FIGURE 5 – Menu principale du jeu

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.

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.

6.4 Menu multi joueurs

Dans ce menu se trouve toute les options disponible pour les parties multi joueurs. 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 multi joueur en ligne avec des options propre à se mode de jeu.
- le bouton "Return" permet de retourner au menu précédant.

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 sélectionner 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 précédant.

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 selectionner une langue, cela se fait depuis une liste déroulante. Les langues disponible sont l'anglais, le français, l'italien et le japonais. Viens 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".

6.7 Menu d'administration

Après avoir cliquer 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 créé une nouvelle carte.
- Le bouton "List of cards" permet d'accéder au tableau de toute 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.

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 quatres challenge ainsi que les quatres 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 créé si toute les informations sont bien renseigner. Le bouton "Clean" permet de vider tous les textfields.

6.9 Tableau carte

Cette interface est composé 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 selection de carte de supprimer la carte selectionné. 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 mis à jour.

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^{ème} 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 6 – Rapport de couverture du code

8 Conduite du projet

Voici l'histoire d'un nain capable de courir vite et de voyager loin.
Dans son épopée formidable nous le suivrons une bière à la main.

9 Conclusion

Voici l'histoire d'un nain capable de courir vite et de voyager loin.
Dans son épopée formidable nous le suivrons une bière à la main.

Glossaire

TTMC Tu te mets combien?. 2