

Rapport du projet de Java S6 – Team Will\_it\_compile.java

Théo Delettre, Rémy Cherkaoui, Lilian Favre-Garcia, Timothé Chauvet

mme. Melekhova

Java S6 Promo 2022

Table des matières

[Introduction 2](#_Toc42775184)

[Répartition des tâches 2](#_Toc42775185)

[Diagrammes UML 3](#_Toc42775186)

[Diagramme d’activité 3](#_Toc42775187)

[Diagramme de cas d’utilisation 4](#_Toc42775188)

[Diagramme d’états 4](#_Toc42775189)

[Diagramme de classes 5](#_Toc42775190)

[Fonctionnement de L’application 6](#_Toc42775191)

[Création et sauvegarde de Questions 6](#_Toc42775192)

[Chargement et sauvegarde des questions 6](#_Toc42775193)

[Système de timer 6](#_Toc42775194)

[Joueurs 6](#_Toc42775195)

[Interface utilisateur 7](#_Toc42775196)

[Retour d’expérience 8](#_Toc42775197)

# Introduction

Pour ce module de Java du Semestre 6 en électif informatique, notre équipe composée de

* **Théo DELETTRE ;**
* **Rémy CHERKAOUI ;**
* **Lilian FAVRE-GARCIA ;**
* **Timothé CHAUVET**.

Il nous a été demandé de réaliser **un système avec interface graphique pour faire un jeu de questions-réponses**. Quatre à 20 joueurs participeront à ce jeu, trois niveaux de difficulté et types de questions seront implémentés, deux livrables et une interface.

Ce projet a été réalisé à l’aide de la bibliothèque **Javax.Swing**, fournissant une base d’éléments permettant de rendre une application Java sur console classique plus intuitive et interactive grâce à l’utilisation d’interfaces graphiques. Il nous a été aussi nécessaire d’apprendre et d’utiliser d’autres concepts comme **la généralisation, la surcharge de méthodes, l’héritage, la gestion de vecteurs et listes chaînées, les threads, les interfaces** ou encore la **liaison Interface – Code**.

Nous avons utilisé **l’outil Git** pour la gestion et le partage de notre code ; vous pourrez retrouver le dépôt à ce lien : <https://github.com/timothecht/JavaProject2020>

Vous trouverez dans la suite de ce rapport l’étendue du travail que nous avons fourni.

# Répartition des tâches

Nous avons réparti les tâches comme suit :

|  |  |  |
| --- | --- | --- |
| Membre de l’équipe | Rôle | |
| Théo DELETTRE | Phases de jeu, nettoyage du code | |
| Rémy CHERKAOUI | Structure des classes, test et débugge du code |
| Lilian FAVRE-GARCIA | Interface graphique | |
| Timothé CHAUVET | UML, test et débugge du code, rapport |

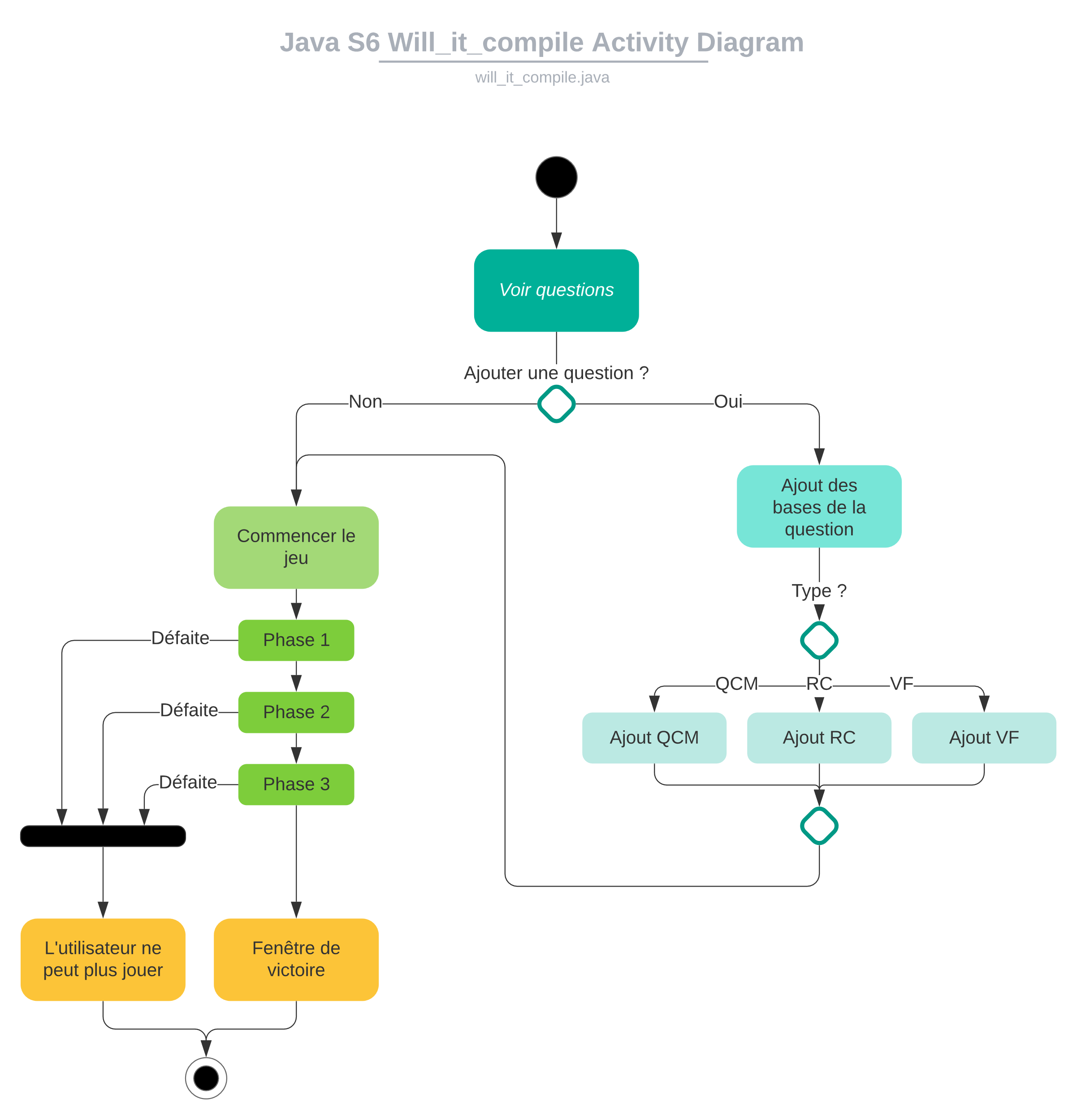
# Diagrammes UML

Nous avons réalisé un **diagramme de classes UML** afin de représenter les différentes classes qui représentent notre programme.

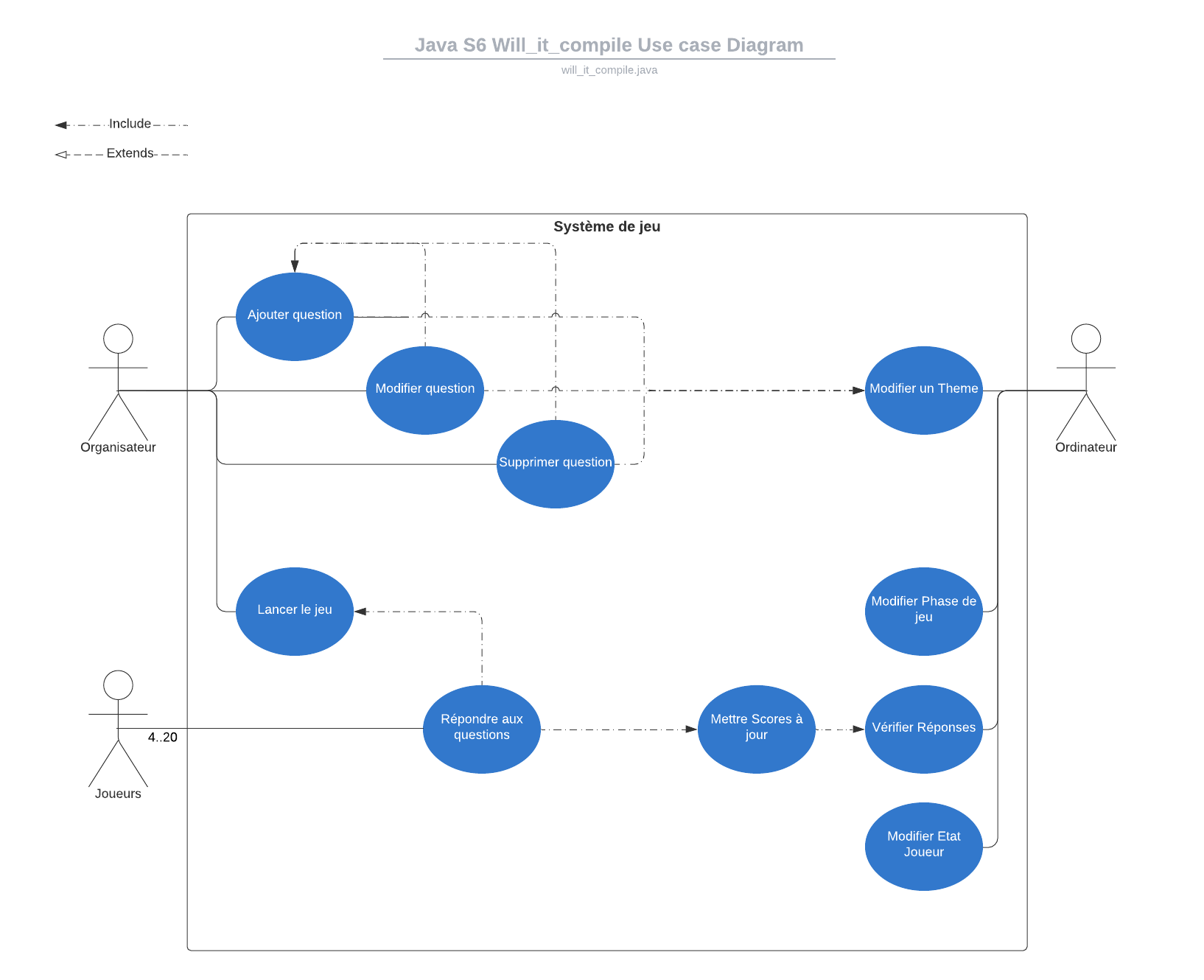
* Diagramme d’activité
* Diagramme de cas d’utilisation
* Diagramme d’états
* Diagramme de classes

Pour mieux visualiser le diagramme, vous pouvez zoomer sur les diagrammes suivants avec CTRL et + ou CTRL et en roulant la molette vers le haut.

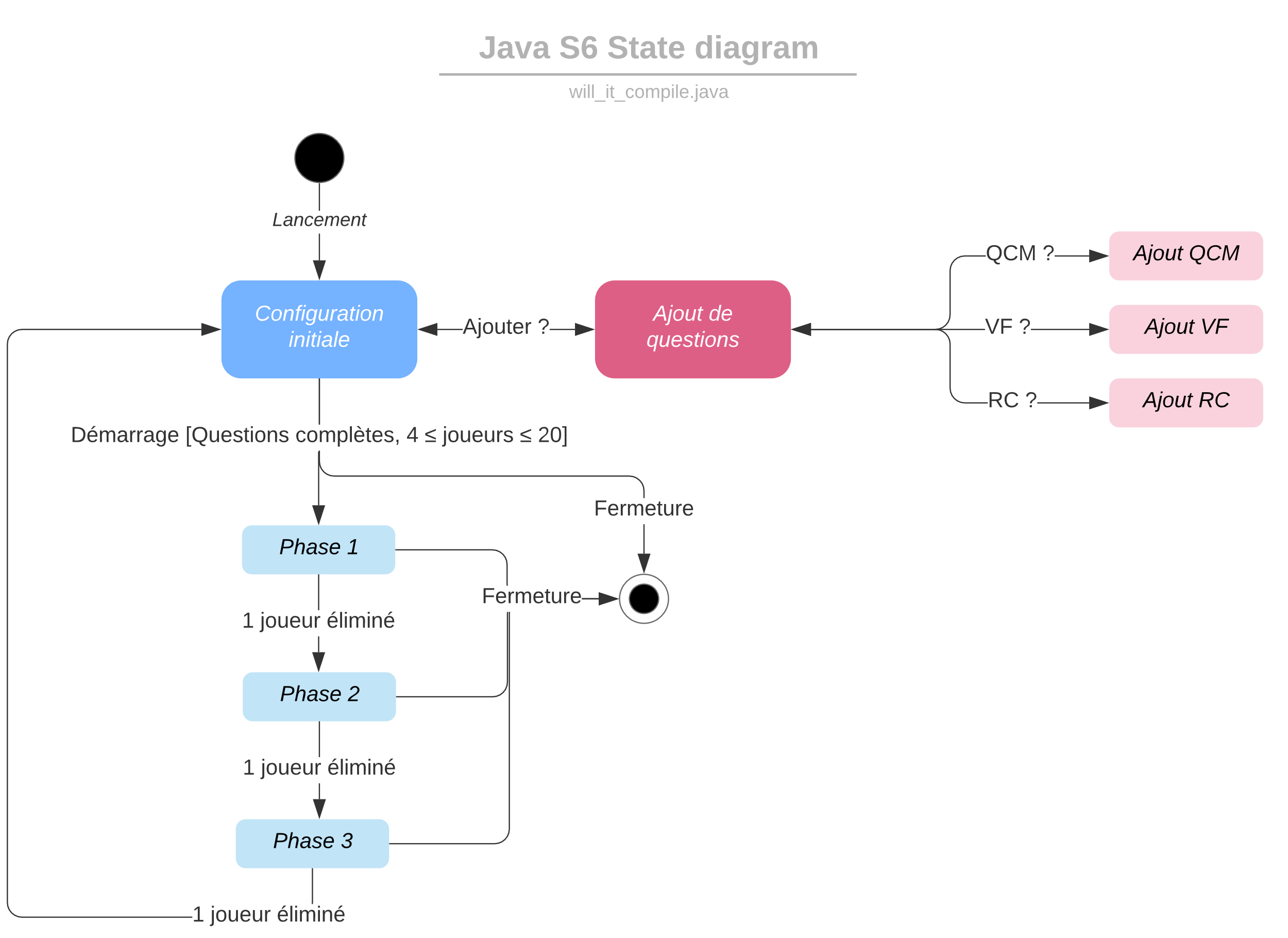
## Diagramme d’activité



## Diagramme de cas d’utilisation



## Diagramme d’états



## Diagramme de classes

# Fonctionnement de L’application

## Création et sauvegarde de Questions

Notre jeu se base sur trois types des questions auxquelles des joueurs doivent répondre pour gagner des points e progresser : Des Question à Choix Multiples (QCM) comportant quatre réponses possibles, des questions à la Réponse Courte (RC) dont la bonne réponse doit être entrée par le Joueur et les question Vrai ou Faux (VF) auxquelles il faut simplement répondre par “vrai” ou “faux” via des variables booléennes.

Ces questions doivent être créées et sauvegardées avant le jeu afin de garantir une partie palpitante ! Pour ce faire, l’organisateur dispose d’une interface accessible dès que le programme est lancé. Cette interface permet l’ajout de questions de différents types (QCM, RC et VF). Ces questions fraichement crées seront ajouter au thème adéquat choisis par l’organisateur. Thème qui serra sauvegarder lui-même et dont le processus et sauvegarde et de chargement est détaillé plus bas.

## Chargement et sauvegarde des questions

Notre jeu comporte dix thèmes différents contenus dans une instance singleton de notre classe “Themes”. Ceci permet de gérer plus facilement les thèmes car ils sont réunis dans une seule et même classe. Ces dits thèmes contienne eux-mêmes différentes questions en relation avec ledit thème. Ces questions sont contenues dans un attribut de la classe “Theme” de type “ListeQuestions” ; une liste contenant des questions. Ceci permet d’organiser le code et faciliter l’accès à un certain type de question en fonction de leur thème et de leur difficulté.

C'est l'attribut listeQuestions de ces instances de la classe “Theme” qui sont sauvegardés en tant que fichier texte.txt dans un sous-dossier ‘Questions’ au sein du dossier du projet grâce à l’utilisation d’un ObjectOutputStream qui écrit l’état actuel d’un objet (ici la liste de Question) en tant que données brutes dans un fichier. A l’inverse, on peut se servir d’un ObjectInputStream afin de charger les données brutes d’une liste de Questions depuis un fichier texte.txt dans la mémoire du programme.

## Système de timer

## Joueurs

Les joueurs sont stockés dans une classe comparable. Ils possèdent un numéro, un score, un état allant de 0 à 4 et un temps. Les 5 états du joueur sont implémentés dans des variables statiques :

public static final int SELECTED        = 0;

public static final int WINNER          = 1;

public static final int SUPER\_WINNER    = 2;

public static final int ELIMINATED      = 3;

public static final int WAITING         = 4;

Chaque joueur peut saisir une question, mettre à jour son score et changer son état. Un joueur peut aussi être comparé à un autre, et il est aussi possible d’afficher les caractéristiques du joueur.

Les joueurs sont implémentés dans une ArrayList de type joueur, dont la classe EnsJoueurs s’occupe. Ainsi, l’ArrayList a une taille de 20, et la classe permet de changer de tour, de selectionner un joueur, de créer un joueur, d’afficher les joueurs et d’en sélectionner un.

## Interface utilisateur

# Retour d’expérience

Ce projet de Java nous a enseigné les fondations d’une bonne application avec interface graphique. Il est nécessaire que l’interface implémente tous les éléments nécessaires afin de créer les objets locaux et invisibles à l’utilisateur pour le fonctionnement du jeu. Donner à l’utilisateur un moyen d’interagir facilement avec un système complexe invisible est une tâche qui nécessite l’empathie du processus de *design thinking*.

Notre équipe de quatre membres a su coordonner son travail grâce à l’outil Git et la plateforme Github. La distribution du code a ainsi été facilitée et nous a permis d’être plus productifs quant à la création d’une application aussi complexe.

L’une des difficultés a été de lier l’interface graphique au code. En effet, l’héritage avec Swing ne permet pas l’utilisation de variables globales ou le transfert aisé de variables. En conséquence, nous avons décidé d’utiliser des variables statiques afin de pouvoir y accéder depuis n’importe-quelle interface utilisateur.

Globalement, nous estimons avoir bien réussi un projet complexe de la sorte. Nous avons réussi à coordonner une équipe décentralisée techniquement et physiquement, en sachant rassembler des fragments de programmes et d’en créer un ensemble cohérent.