Activité intégrative 2021

Itération 1

Durée : 10 heures

Cette itération a pour but de créer le projet, d'y intégrer le code de l'interface graphique et de commencer le développement des US 1.1, 1.2 et 1.3.

Planification de l'itération 1

1 : découverte et présentation de l'énoncé (45 min)

En grand groupe, le responsable présente le projet, les modalités d'évaluation, etc.

2 : créer et configurer le projet (30 min)

Individuellement, les étudiants essaient de créer et configurer le projet (voir ÉTAPE INITIALE :). Le responsable peut éventuellement faire cette activité en même temps que les étudiants.

3 : concevoir l'application (2 h 30 min)

Partez des modèles fournis en fin d'énoncé et appliquez le *RDD* pour concevoir votre solution. Procédez en trois étapes :

1. Individuellement, élaborez une liste **de responsabilités candidates** (15 min).
2. En petits groupes (3 à 4 étudiants, si compatible avec les règles COVID-19 en vigueur), **complétez le** modèle de cartes CRC et les diagrammes de collaboration. **Créez un diagramme de classe qui met en évidence les méthodes publiques** (~1 h).
3. En grand groupe, vous élaborez une solution commune (~1h 15 min).

Cette étape est également un moment idéal pour réfléchir aux choix de collections (interface et implémentation).

4 : implémenter et tester de l'application (6 heures)

Nous vous recommandons d'implémenter progressivement l'application, une user story à la fois. **Attention,** votre implémentation doit réussir au moins 7 des 8 tests d'acceptation pour que votre itération soit jugée recevable.

Étape initiale : démarrer le projet

En tant que programmeur je souhaite exécuter un squelette du programme afin de valider mon environnement de développement.

Architecture du projet et construction du système

Avant d'implémenter les fonctionnalités du programme, vous devez mettre en place une architecture. L'architecture d'un programme est un ensemble de règles qui structurent son code source. En dotant votre programme d'une architecture et en veillant à sa cohérence, vous facilitez sa prise en main par d'autres programmeurs, notamment vos responsables 😊.

Nous souhaitons mettre en place une architecture qui sépare clairement le code qui dirige une partie du système (le superviseur [`Superviser`]) de sa représentation à l'écran (nous parlerons de sa vue [`View`]). La Figure 1 présente une vue statique.

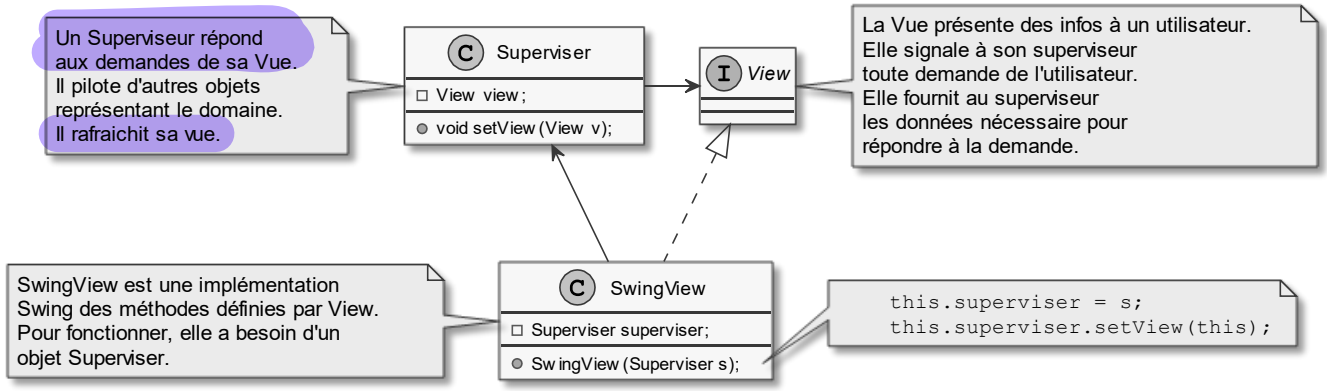

Figure 1 Vue statique présentant l'interface d'une vue, son implémentation et le superviseur qui interagit avec

À ce stade, retenez que chaque itération possède sa propre version du triplet `<superviseur, vue, implémentation en swing de la vue>`. Quand vous construirez le système, les vues des triplets seront intégrées à une fenêtre principale Swing qui les présentera sous forme d'onglets. La fenêtre principale, les superviseurs et les vues concrètes sont définis dans la méthode main de la classe Program.

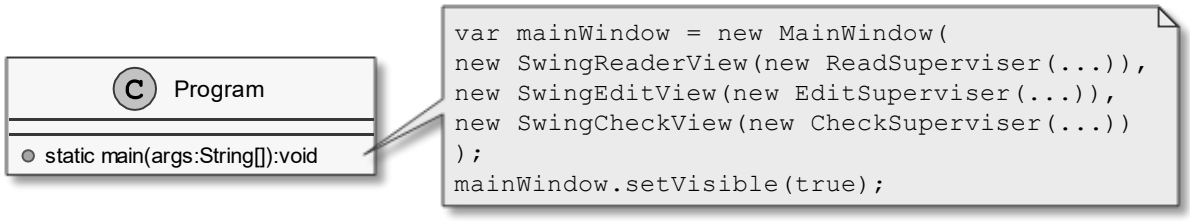

Figure 2 Le main construit le système logiciel et affiche la fenêtre principale.

Les détails de l'architecture seront approfondis par les sections suivantes. À ce stade, notre objectif est d'afficher cette fenêtre principale.

Création et configuration du projet (~15 min)

Créez un projet Eclipse intitulé ai2021.nom.prenom. Ajoutez-y un répertoire de source `tests` et la bibliothèque JUnit5. Enfin, ajoutez le paquetage `gamebook` aux répertoires de source. La structure de départ est définie, mais elle ne suffit pas : nous devons ajouter quelques bibliothèques tierces.

Pour afficher les vues, le code fourni dépend de la bibliothèque miglayout pour Swing. Créez un dossier libs à la racine du projet : nous y placerons nos archives jar. Téléchargez l'archive

ai2021.miglayout.zip depuis HELMo Learn et décompressez-la dans le dossier libs. L'archive contient deux archives jar correspondant respectivement au cœur de MigLayout et à son implémentation Swing.

Reste à indiquer à Eclipse que notre projet dépend de ces archives. Pour ce faire, rendez-vous dans les propriétés de votre projet et affichez l'onglet Librairies de l'item *Java Build Path*. Appuyez sur le bouton *Add JARs…* et sélectionnez vos bibliothèques (les bibliothèques sont internes au projet, pas besoin d'ajouter de jar externes). La Figure 3 présente le résultat attendu.

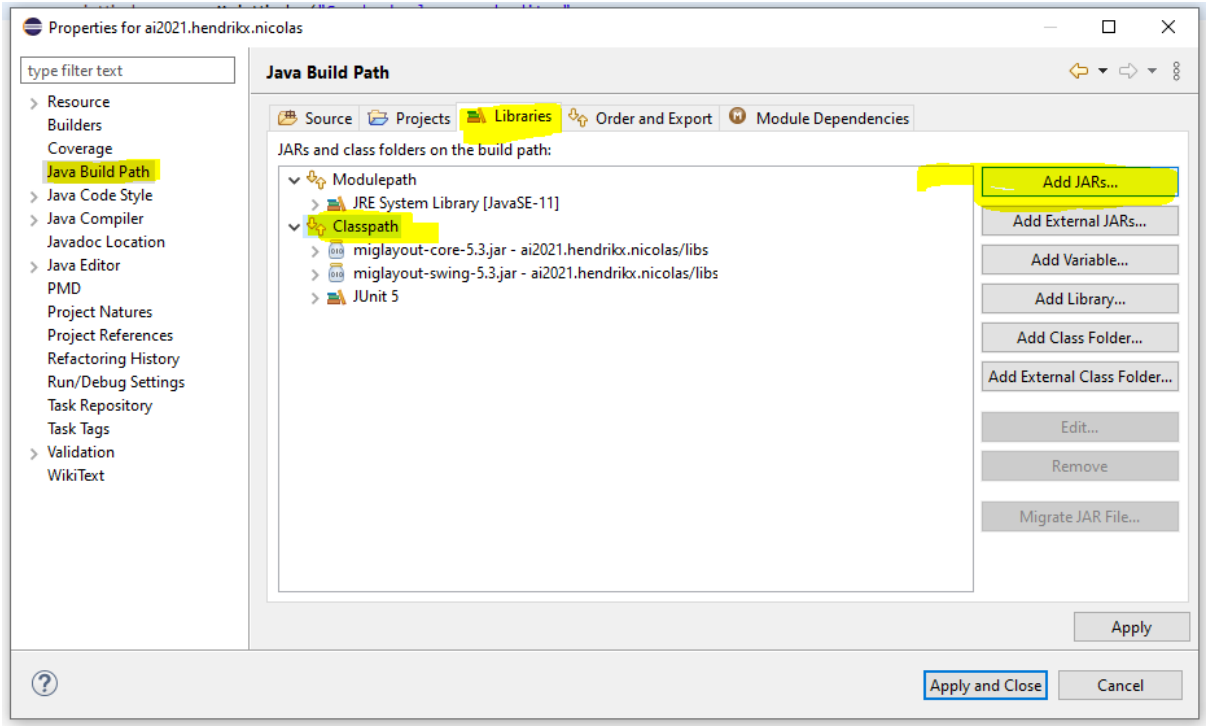

Figure 3 Ajout des bibliothèques miglayout-core et miglayout-swing

Nous pouvons enfin ajouter le code fourni. Téléchargez et décompressez l'archive ai2021.src.zip dans votre répertoire src. Téléchargez et décompressez l'archive ai2021.tests.zip dans votre répertoire tests. Téléchargez et ajoutez le fichier ai-ruleset.xml à la racine de votre projet et configurez PMD pour qu'il travaille avec ce fichier.

Vous devriez obtenir une structure finale proche de celle présentée par la figure suivante. Tentez d'exécuter la classe `gamebook.Program`. Si tout va bien elle affichera un écran analogue à la Figure 4.

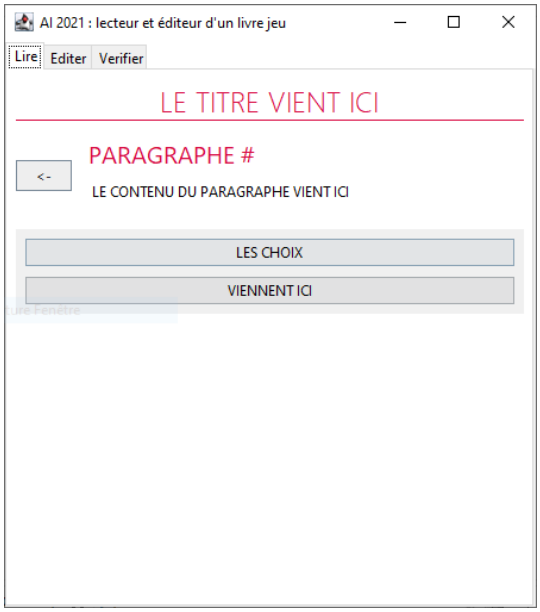

Figure 4 Données présentées par le squelette

À ce stade, vous pouvez attaquer la conception.

AI-1.1 : lire un livre jeu

En tant que lecteur, je souhaite choisir une action afin d'afficher le paragraphe suivant.

Détails

- On part du principe que le premier paragraphe d'un livre-jeu est son paragraphe de départ
- Les textes des choix possibles depuis un paragraphe sont uniques au sein de ce paragraphe
- Les choix au sein d'un paragraphe n'ont pas d'ordre particuliers
- Après avoir démarré, le programme affiche le titre du livre-jeu, le contenu et les choix du premier paragraphe
- Quand l'utilisateur appuie sur le bouton correspondant, le programme présente à l'utilisateur le contenu du paragraphe ciblé par le choix

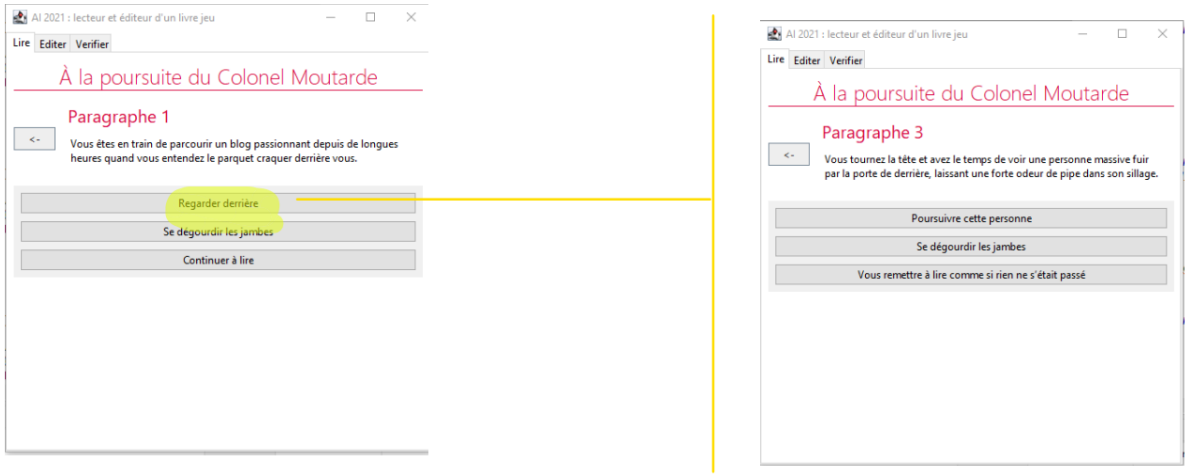

Figure 5 Début de lecture et transition provoquée par le choix "Regarder derrière"

Tests d'acceptation

Remarque : Comme leur nom l'indique, les tests d'acceptation déterminent si votre programme fonctionne comme attendu. La classe `gamebook.supervisers.ReadSuperviserTests` implémente ces tests. **Nous l'utiliserons pour valider la partie fonctionnelle de l'itération.**

- ✓ Quand le programme a démarré, je peux voir le titre du livre, le contenu et les choix du paragraphe 1.
- ✓ Étant donné que le programme affiche le paragraphe 1, quand j'appuie sur le bouton « Se dégourdir les jambes », alors le programme affiche le contenu et les choix du paragraphe 4.
- ✓ Étant donné que le programme affiche le paragraphe 4, quand j'appuie sur le bouton « Poursuivre cette personne », », alors le programme affiche le contenu et les choix du paragraphe 5 (dans ce dernier cas, il n'y a pas de choix).

AI-1.2 : finir la lecture

En tant que lecteur, je souhaite savoir quand j'ai achevé la lecture du livre, afin d'entamer une nouvelle aventure.

Détails

- Un paragraphe sans choix est un paragraphe de fin de lecture.
- Le programme signale la fin de l'histoire et propose de retourner au menu principal (voir figure ci-dessous).

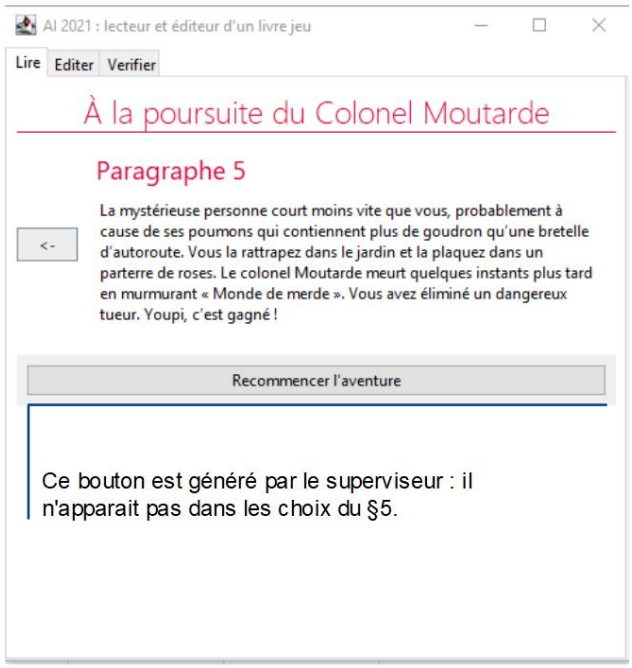

Figure 6 Fin de lecture

Tests d'acceptation

- ✓ Quand le programme atteint un paragraphe de fin, il propose de recommencer la lecture sous forme de choix fictif.
- ✓ Quand l'utilisateur appuie sur le bouton « Recommencer la lecture ». Alors le programme redémarre l'aventure depuis le premier paragraphe.

AI-1.3 : revenir au paragraphe précédent

En tant que lecteur, je souhaite revenir sur mon dernier choix, afin d'explorer d'autres possibilités.

Détails

- Les choix de l'utilisateur seront mémorisés par une classe dédiée.
- En début de lecture, le programme ignore toute demande de retour en arrière.
- Le retour en arrière et le nouveau choix seront mémorisés de sorte qu'il devient possible de rejouer tout le parcours de l'utilisateur (en tenant compte de ses choix précédents).
- Commencer une nouvelle lecture efface le parcours précédent.

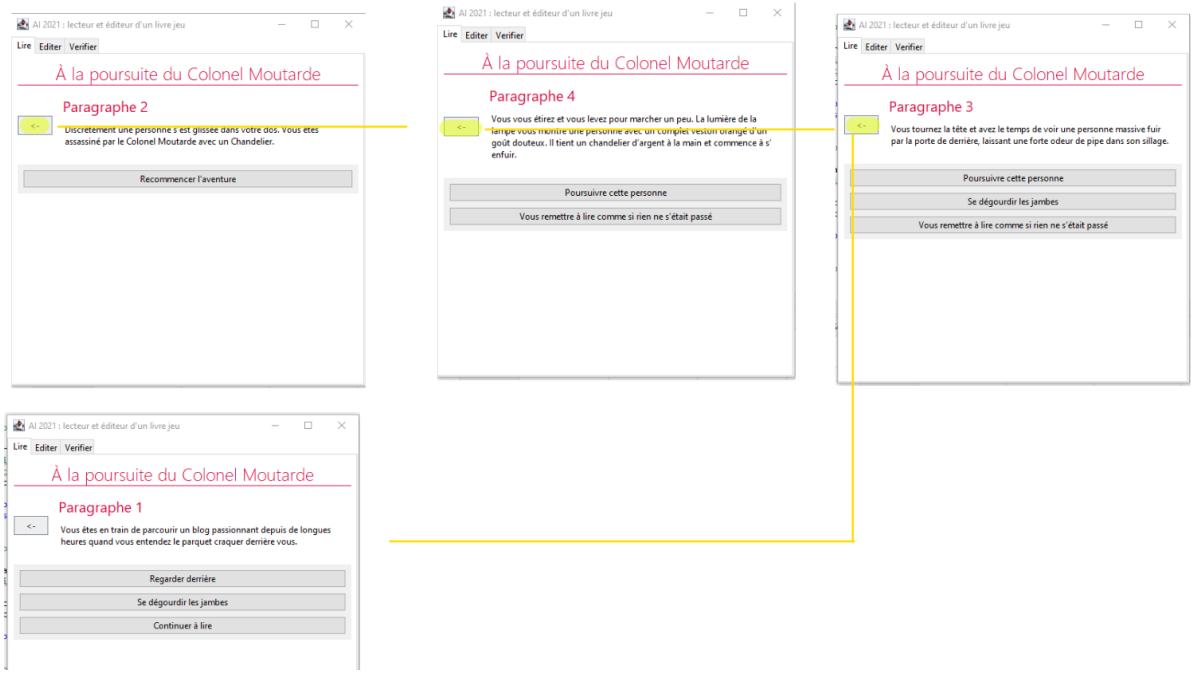

Figure 7 Lecture à rebours possible à partir de §2

Tests d'acceptation

- ✓ Quand le programme démarre la lecture, si l'utilisateur demande un retour en arrière, alors le programme l'ignore.
- ✓ Quand le programme est en cours de lecture, si l'utilisateur demande un retour en arrière, alors le programme affiche le paragraphe précédent.
- ✓ Soit un parcours de N paragraphes, quand le programme atteint un paragraphe de fin, si l'utilisateur revient N fois en arrière, alors le programme rejoue la séance de lecture à rebours.
- ✓ Quand le programme atteint la fin du livre, si l'utilisateur revient en arrière, l'application affiche le dernier paragraphe et permet d'opter pour un autre choix.

Consignes et contraintes

Algorithmique

Répondez aux questions ci-dessous en commentaires dans votre code source. Pour chaque question, nous indiquons l'endroit où la réponse doit être écrite.

Invariant[1] sur l'état d'un livre-jeu

OK

Nous souhaitons que notre application présente le contenu du §1 de votre livre-jeu. Quel invariant tout livre-jeu doit respecter ?

Répondez à cette question dans la Javadoc en-tête de la classe GameBook.

Choix de collection pour les paragraphes

OK

Un livre jeu possède des paragraphes.

Quelle interface de collection (`List, Queue, Deque, (Sorted)Set, (Sorted)Map, etc.`) allez-vous choisir pour mémoriser les paragraphes d'un livre jeu ? Justifiez votre choix en identifiant les principales opérations dont vous aurez besoins au cours de cette itération ? Avez-vous notamment besoin d'accéder à un élément précis ? Si oui, sur base de quelle sorte de clé ?

Quelle implémentation de collection (`ArrayList, LinkedList, ArrayDeque, HashSet, TreeMap, etc.`) allez-vous choisir pour mémoriser les paragraphes d'un livre-jeu ? Justifiez votre choix en déterminant les CTT des principales opérations que vous utiliserez pour l'itération 1.

Répondez à cette question dans la Javadoc en-tête de la classe GameBook.

Choix de collection pour mémoriser les paragraphes lus ?

OK

Une session de lecture mémorise les paragraphes lus.

Quelle interface de collection (`List, Queue, Deque, (Sorted)Set, (Sorted)Map, etc.`) allez-vous choisir pour mémoriser les paragraphes lus ? Justifiez votre choix en identifiant les principales opérations dont vous aurez besoins au cours de cette itération ? Avez-vous notamment besoin d'accéder à un élément précis ? Si oui, sur base de quelle sorte de clé ?

Quelle implémentation de collection (`ArrayList, LinkedList, ArrayDeque, HashSet, TreeMap, etc.`) allez-vous choisir pour mémoriser les paragraphes d'un livre-jeu ? Justifiez votre choix en déterminant les CTT des principales opérations que vous utiliserez pour l'itération 1.

Répondez à cette question dans la Javadoc en-tête de la classe Session.

Postcondition pour le retour en arrière qui retire le dernier paragraphe lu de la session de lecture ?

OK

Quand vous ajoutez les paragraphes p_1, p_2, …, p_m à votre session, quel est le dernier paragraphe lu ? Si vous retirez le dernier paragraphe lu 1 fois, le dernier paragraphe lu devient… ? Si vous retirez le dernier paragraphe lu m fois, le dernier paragraphe lu devient….

Répondez à cette question dans la Javadoc en-tête de la méthode concernée de la classe Session.

CTT pour trouver le paragraphe ciblé par un choix ?

OK

Évaluez la CTT de la méthode `onChoiceSelected(String choiceKey)` pour retrouver le paragraphe ciblé par choiceKey. Pour répondre à cette question, identifiez les boucles, les

[1] Un invariant d'objet est une condition qui est toujours respectée par un objet pendant tout sa vie.

imbrications, mais aussi les collections utilisées et leurs opérations. Quand vous répondez, n'oubliez pas d'indiquer à quoi correspondent vos libellés N,M, etc.

Répondez à cette question dans la Javadoc de méthode ReadSuperviser.`onChoiceSelected(String choiceKey)`.

Programmation Orientée Objet

Vous devrez créer de nouvelles classes modélisant le domaine. Placez ces classes dans le paquetage `gamebook.domains`. Il y a au moins deux classes à créer : la classe `GameBook` et la classe `Session`. D'autres classes sont sûrement à prévoir…

Soignez l'encapsulation et rendez vos classes timides (réduisez les alertes d'infraction à Déméter signalées par PMD).

Créez un objet `GameBook` qui représente le livre jeu donné en exemple de l'énoncé général. La classe `GameBookFactory` propose les données correspondantes sous forme de constante.

Créez une classe de test par classe du domaine. Testez vos classes de sorte que chaque classe de test couvre 90% des instructions de sa classe testée.

Vous devrez également implémenter les méthodes du `ReadSuperviser`. En revanche, vous ne pouvez pas changer son interface à l'exception du constructeur : modifiez le constructeur pour qu'il reçoive le `GameBook` à lire et une `Session`.

Documents de départ pour la conception

Les documents suivants doivent vous aider à concevoir votre solution pour l'itération 1.

Conseil : concentrez-vous sur les récits utilisateurs présentés dans ce document et à ne cherchez pas à anticiper des fonctionnalités qui n'ont pas été encore présentées.

Modèle de cartes CRC

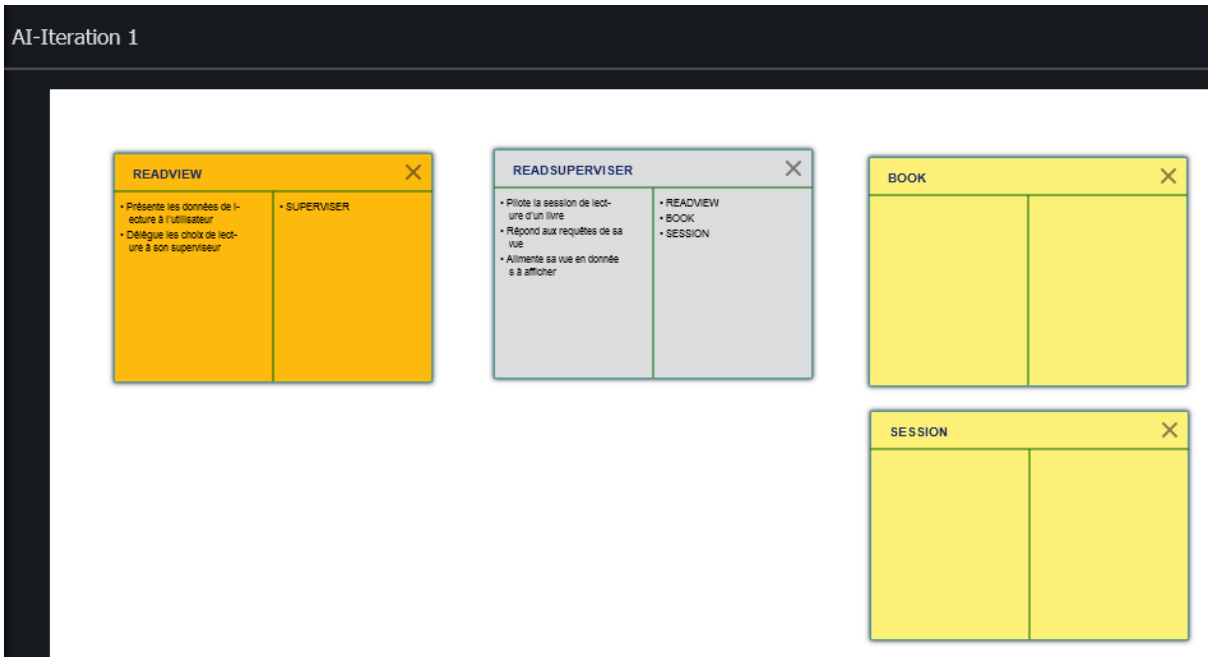

Figure 8 squelette de cartes CRC à compléter

Diagrammes de collaboration

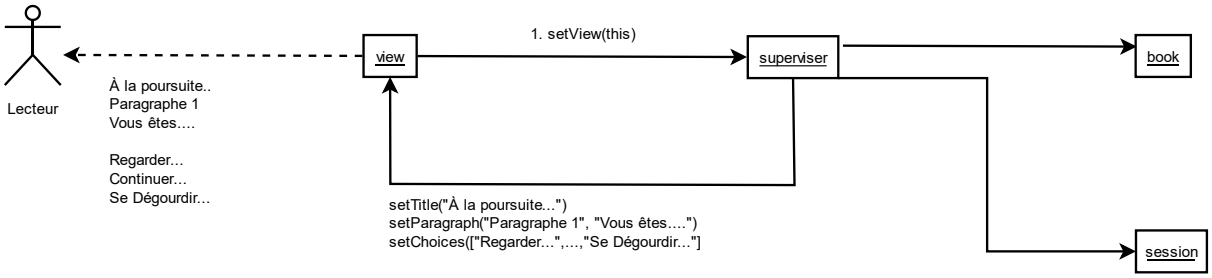

Figure 9 squelette à compléter pour afficher le §1 au démarrage

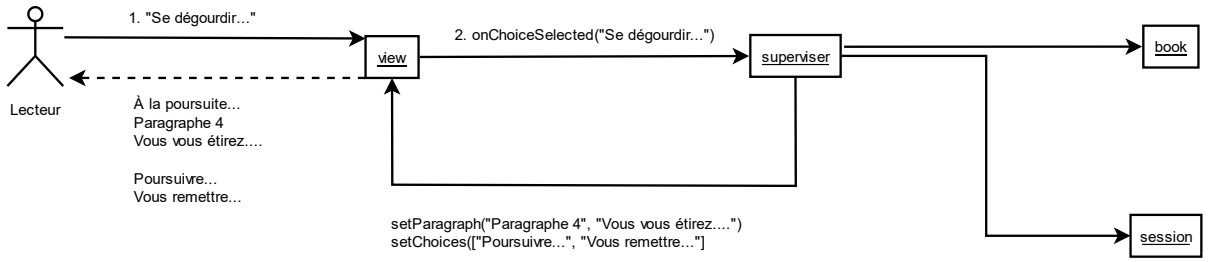

Figure 10 squelette pour passer du §1 au §4

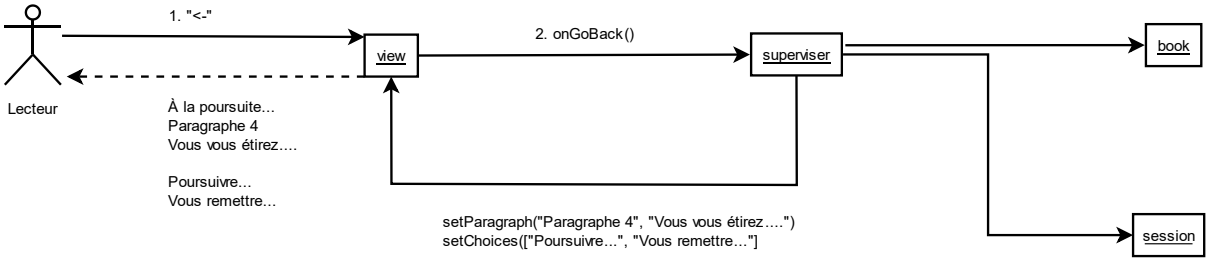

Figure 11 squelette pour revenir de §5 à §4