Rapport Conception Logicielle Avancée

Antonin Montagne, Lou-Anne Gautherie, Oumoul Kiramy Bah,
Pape Maguette Diongue

4 Avril 2022

UNIVERSITÉ
CAEN
NORMANDIE

Table des matières

1 Introduction

1.1 Description du projet

Nous avions pour but de coder en Java, un jeu qui s'intitule "Un livre dont vous êtes le héros". Ce genre de jeu est une progression de lecture en fonction des choix et du résultat des actions du lecteur.
Ce genre de "livre" est très populaire chez les enfants, comme chez les adultes. La collection de livre "Sherlock Holmes" possède sa propre série de "livre dont vous êtes le héros".

Quelques exemples de cette série :

- Meurtre au Club Diogène
- L'Émeraude de la Rivière Noire
- L'Héritier Disparu

1.2 Plan du rapport

Nous évoquerons d'abord quels étaient nos objectifs de départ (??), puis nous détaillerons les différentes étapes de la création de notre projet avec les rôles de chacuns (??). Ensuite nous présenterons les éléments techniques utilisées (??) dans notre code, ainsi que l'achitecture de ce projet (??). Finalement nous présenterons certaines expérimentations (??) et terminerons par une courte conclusion (??).

2 Objectifs du projet

Nous voulions développer un editeur de texte, dans lequel le joueur peut créer sa propre histoire dont le lecteur est le héros. Cet editeur se présenterait sous forme de page vierge avec une fenetre qui demanderait à l'utilisateur de taper l'introduction de son histoire, puis les différents chapitres, qui conduisent à d'autres chapitres en fonction des choix choisis. D'un autre côté, le joueur pourrait choisir de charger son histoire précédemment écrite, ou une déjà existante. Une fois l'histoire chargée, le joueur pourra jouer de manière basique en suivant les choix déjà définis, ou d'en créer, ou supprimer en temps réel. Un fois la fenêtre du jeu fermée, le graphique de l'histoire s'afficherait sous format .png.

3 Fonctionnalités implémentées

3.1 Description des fonctionnalités

Les possibilités du jeu sont multiples. A partir de l'accueil, il est possible de créer une histoire à partir de rien, c'est-à-dire d'écrire l'intro, ainsi que de créer les choix possibles. On pourra choisir d'enregistrer cette nouvelle histoire dans un fichier json, et de la charger. Il est aussi possible de modifier une histoire déjà écrite en ajoutant (avec le bouton "*plus*") ou supprimant (avec le bouton "*moins*") des chapitres et des choix. La dernière fonctionnalité est de charger les histoires .json. L'intro s'affichera ainsi qu'un bouton "*Commencer l'histoire*" qui lancera le premier chapitre avec les choix en dessous. Il suffira juste de cliquer dessus pour accéder au chapitre correspondant. En fermant la fenêtre, le graphe de cette histoire s'affichera.

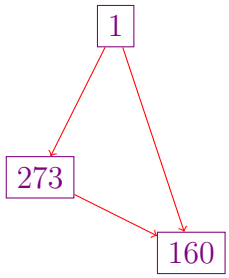


FIGURE 1 – Graphe du premier chapitre et des premiers choix du fichier *fotw.json*

3.2 Organisation du projet

Pour commencer nous avons pris le fichier fotw.json qui était joint sur ecampus avec la description du projet. Ce fichier contient une histoire déjà écrite avec 350 chapitres comportant tous des choix. Cette histoire est donc séparée en "sections" allant de 1 à 350.

Oumoul Kiramy a codé la classe Noeud qui permet de gérer un fichier JSON, c'est-à-dire accéder au numéro du chapitre, à son texte, ou à ses choix.
Cette classe Noeud contient des méthodes qui permettent de récuperer des

sections dans le livre qui représente des chapitres ainsi que les autres informations du livre comme les choix possibles pour lire la suite du livre.

Les méthodes de recupération sont les suivantes :

- `getNumber()` récupère le numéro du chapitre courant.
- `getTexte()` récupère le texte du chapitre.
- `getChoiceTexte()` retourne le texte qui propose d'aller aux chapitres suivant.
- `getChoiceNumber()` retourne les numéros des choix suivants proposés sous forme d'`ArrayList`.
- `choice` retourne les choix sous forme d'`Hashmap`, avec comme clé, les choix suivant et comme valeurs, les textes des choix.

En plus de ces methodes de récupération, Oumoul Kiramy a aussi créé des méthodes qui permettent de créer des nœuds :

- `addNoeud()` (comme son nom l'indique) ajoute un chapitre dans un livre.
- `ajoutChoix()` rajoute des numéros pour aller à un autre chapitre.
- `ajoutChoice()` permet de rajouter en même temps le texte qui propose d'aller à un chapitre et le numéro de ce chapitre.

Toutes ces informations sont stockées dans un fichier JSON qui contient principalement une introduction ainsi qu'un `JSONObject sections` qui est l'objet principal du livre vu que c'est lui qui contient tous ses chapitres.
Ainsi pour créer un livre, une fois que toutes les informations, comme le titre et les objets sections sont crées, un fichier JSON est créé à l'aide du paquetage `FileWriter`. Et pour recupérer les informations des fichiers, on utilise les méthodes `JSON` de java qui marche avec la librairie `json-simple-1.1.1.jar` se trouvant dans le dossier lib du projet.

Après avoir pu accéder à tous les noeuds de ce fichier fotw.json, Antonin et Lou-Anne se sont occupés de la partie interface lorsque nous chargeons ce fichier là. Pour cela, ils se sont aidés des sites : **Oracle.com** [?] et **Jmdoudoux.fr** [?].

Lou-Anne a d'abord décidé de créer deux boutons sur la page d'accueil, un pour "*Créer / modifier une histoire*", et un autre pour "*Charger une histoire*". Ces deux boutons se trouve dans la fonction `afficherAccueil()`,

dans la classe Fenetre et affichent la liste des fichiers d'extension .JSON contenus dans le paquetage "files". Pour ce faire, elle a récupérée les fichiers se trouvant dans le dossier, les a stockés dans une liste `DefaultListModel<String>`, puis elle a transformé cette liste en `JList<String>` pour pouvoir la manipuler avec `swing`. Cette liste est devenue par la suite cliquable avec la fonction `setSelectionMode()` et pour récupérer le fichier lorsqu'il est cliqué, elle a utilisé la fonction `getSelectedValue()` pour récupérer son nom.

Pour le bouton "*Charger une histoire*", après avoir choisit le fichier que l'on veut charger, cela affiche l'intro de l'histoire grâce à la classe "ModifierChargerHistoire qui lance la fonction `afficherIntro()` codé par Antonin.
Une fois l'introduction de l'histoire affichée, Lou-Anne a codé un bouton "*Commencer l'histoire*" qui lance un évènement créé par Antonin ; cet évènement crée le premier noeud, retire le bouton du panel des boutons puis affiche le premier paragraphe de l'histoire avec un appel à la fonction `afficherParagraphe()`, ainsi que la liste de tous les chapitres à gauche de l'écran.
Une fois arrivé sur le premier chapitre, nous avons le texte du premier chapitre puis nous pouvons cliquer sur l'un des choix. Le bouton "*Commencer l'histoire*" laisse place au bouton "*accueil*". Antonin a crée un autre évènement qui, lors du clic sur ce bouton, supprime tous les objets que contient le panel principal, le panel liste sections, le panel boutons et le panel texte. Ensuite, il appelle la fonction `afficherAccueil()` pour revenir à la fenêtre d'ouverture du logiciel.

Pour le bouton "*Créer / modifier une histoire*", si nous choisissons une histoire existante ; l'introduction s'affiche comme dans le premier cas avec le même bouton. Une fois le premier paragraphe affiché, on peut retrouver la liste des sections sur la gauche, au centre le texte du paragraphe, en dessous la liste des choix puis les boutons "*plus*", "*moins*", "*modification texte*", "*accueil*" et "*sauvegarder*". Lors du clic sur le boutons "*plus*", une fenêtre pop-up apparait pour que l'utilisateur rentre le texte de son choix. Si le choix renvoit vers une section qui n'existe pas encore, une nouvelle fenêtre s'ouvre pour qu'on puisse saisir le texte de la nouvelle section. L'ajout de choix met à jour le dictionnaire qui contient tous les choix de chaque section ainsi que la liste des choix. Lors du clic sur le bouton "*moins*", le choix sélectionné est retiré de la liste des choix ainsi que du dictionnaire. Lors du clic sur le bouton "*modification du texte*", le texte saisit dans le panel du texte est enregistré dans le dictionnaire des textes des sections. Lors du clic sur le bouton "*sau-*

vegarder", une fenêtre pop-up apparait pour que l'utilisateur donne le titre de son histoire. Pour créer un fichier JSON Antonin s'est servit de la classe CreationJson. Il a parcouru chaque section du dictionnaire des sections puis a crée un `JSONObject` contenant le texte de la section puis un `JSONArray` qui contient les choix de la section.

Enfin, pour le bouton "*Créer / modifier une histoire*" si nous choisissons de créer une nouvelle histoire ; une nouvelle `JFrame` s'ouvre pour permettre de saisir le texte de l'introduction avec une bouton pour valider. Lors du clic sur ce bouton, le texte est récupéré et stocké dans une variable afin de pouvoir appeler la fonction `afficherIntro()` plus tard et une nouvelle `JFrame` s'affiche. Cette dernière sert à écrire le texte de notre premier paragraphe. Lors du clic sur le bouton de cette page, un noeud est crée avec le texte que nous venons de saisir puis la fonction `afficherIntro()` est appelée avec la variable qui contient le texte de l'introduction. Pour la suite, les fonctionnalitées sont les mêmes que pour le cas où on veut modifier une histoire déjà existante.

Dans la classe "ModifierChargerHistoire" Antonin a récupéré l'introduction et l'objet section grâce à la méthode `get()` de l'objet `JSONObject`. Il les a passé en argument au constructeur des classes ModeEdition et ModeLecture ainsi qu'une condition pour savoir quel mode l'utilisateur veut-il lancer : le mode lecture ou le mode édition. Ensuite on appel la fonction `afficherIntro()` de l'un des deux modes.
Dans les deux modes, l'introduction est affiché de la même façon. Il a ajouté l'introduction au panel dédié. Pour les sections, il les a affiché de la même manière que dans le mode édition.
Pour cela il a crée une `JList` à partir d'un `DefaultListModel` qui contient chaque section qu'il a rendu cliquable. Lors du clic sur une section de la liste, il a créé un nœud qu'il affiche ensuite grâce à la fonction `afficherParagraphe()` que nous verrons plus tard. Pour le texte de chaque noeud, il a récupéré le texte grâce à la méthode `getTexte()` de la classe Noeud qui a ensuite ajouté au panel consacré au texte. Pour la liste des choix, il a utilisé la même méthode que pour faire la liste des sections.

Pour finir, lorsque la fenêtre se ferme, le graphe de l'histoire s'affiche. Pour ce faire, Lou-Anne a utilisée **Graphviz**. Grâce aux classes File et FileWriter de Java, elle a fait en sorte de créer un fichier d'extension .dot, de le remplir

du codage du graphe (grâce à la fonction `toDot2()` qui appelle `toDot()`). La fonction `toDot2()` écrit le début du fichier avec "disgraph G ..." avec les styles des noeuds et des flèches, ensuite, pour chaque chapitre du fichier et ses choix, la fonction `toDot()` est appelée. Elle trace des flèches entre chaque choix afin de les lier. Tout ça est donc stocké dans le fichier d'extension .dot. Avec l'aide de la classe Process de Java, nous pouvons donc écrire des lignes de commandes dans un fichier .java. Elle donc fait en sorte d'éxécuter le fichier output.dot afin de le compiler en output.jpg avec la commande -dot (qui prend en argument un fichier d'extension .dot et renvoie un graphe d'extension .jpg).

Pour les tests unitaires, c'est Pape qui s'en est occupé afin de vérifier le fonctionnement des méthodes de chaque classe.
D'après ses recherches, il a trouvé qu'il existait plusieurs façons de faire les testes unitaires, cependant, il a utilisé la méthode `AsserEquals` pour effectuer les tests.
Pour commencer, il a téléchargé les libraries nécessaires pour les imports et aussi la classe dont il a besoin (la classe Noeud) pour le bon fonctionnement des tests. La méthode `AsserEquals` permet de prendre en argument un élément et le comparer avec une partie du livre que ça soit un numéro de section, une section, un texte, un choix,... etc
Le test passe si les éléments correspondent, et génèrent une erreur sinon.

La méthode `assertEquals()` fonctionne de la manière suivante :
`assertEquals("élément à tester", Élément comparé dans le livre());`

```
@Test
  public void testGetNumber(){
   assertEquals("1", noeud.getNumber());
  }
```

FIGURE 2 – Exemple de la méthode `assertEquals()` avec la fonction `testGetNumber()`.

Les `@Test` marquent les différents tests à effectués.

La fonction `testGetNumber()` permet de prendre un numéro de section et regarde s'il existe parmi les numéros de section dans le livre créé.

Ensuite la fonction `testGetSection()` pour voir si la section existe et correspond au numéro donné en `testGetNumber`.
La fonction `testGetTexte()` permet de vérifier si cet section est créée et le texte à la section donnée, la fonction `testGetChoice()` permet de vérifier une liste de choix faites avec la section , le numéro de section et le texte.
La fonction `testGetChoiceTexte()` permet de voir le test sur le choix du texte donné. En enfin la fonction `testGetChoiceNumber()` vérifie les tests sur le choix du numéro donné argument.

Pour ce rapport, Lou-Anne s'est occupé d'importer tout ce que nous avions fait en TP d'*Outils et Communication*. Elle a donc fait le diagramme des classes grâce à **Dia**, l'introduction du rapport, les expérimentations et a importé les paquetages utilisés et les sources.
Antonin a créé une fichier .bib et utilisé la libairie Bibtex pour importé les sources.
Oumoul Kiramy a codé l'algorithme utilisé pour parcourir le graphe.

4 Eléments techiniques

4.1 Paquetages utilisées

4.1.1 Paquetages Java

Le paquetage `ArrayList` est utilisé dans les classes CreerHistoire, ModeEdition, ModeLecture et Noeud.

Le paquetage `DefaultListModel`est utilisé dans les classes CreerModifierHistoire, Moins, Plus, ModeEdition, ModeLecture et Fenetre.

Le paquetage `File` est utilisé :
Dans Fenetre pour parcourir la liste des fichiers dans le dossier files.
Dans ModifierChargerHistoire pour écrire ce qu'on veut dans le fichier output.dot.
Dans CreationJson pour donner un nom au fichier JSON créé.
Dans LaunchGraph et NoeudTest pour lire le fichier JSON.

Le paquetage `HashMap` est utilisé dans les classes Moins, Plus, Sauvegarder, ModeEdition, ModeLecture, CreationJson, Noeud et Fenetre.

Le paquetage `JSONObject` est utilisé dans les packages mode et noeud et dans les classes Commencer, CreerModifierHistoire, Plus, Sauvegarder et Fenetre.

Le paquetage `Process` est utilisé dans LaunchGraph et ModifierChargerHistoire pour écrire des lignes de commandes dans un fichier .java.

Beaucoup de classes de `swing` ont été utilisées pour l'interface graphique. Ces classes sont les suivantes :

- `JButton` dans le package boutons et dans la classe CreerHistoire
- `JFrame` dans les classes Plus, Sauvegarder, CreerHistoire et Fenetre
- `JList` dans les classes ChargerHistoire, CreerModifierHistoire, Moins, ModeEdition, ModeLecture et Fenetre
- `JPanel` dans les packages boutons, et mode et dans la classe Fenetre
- `JTextArea` dans les classes ChargerHistoire, CreerModifierHistoire, ModificationTexte, Plus, Creer Histoire, ModeEdition, ModeLecture et Fenetre

Le paquetage `Toolkit` est utilisé dans les classes Plus, CreerHistoire, ModeEdition et Fenetre pour définir la taille de l'écran.

4.1.2 Paquetages JAR

Nous avons utilisés plusieurs archives .jar :

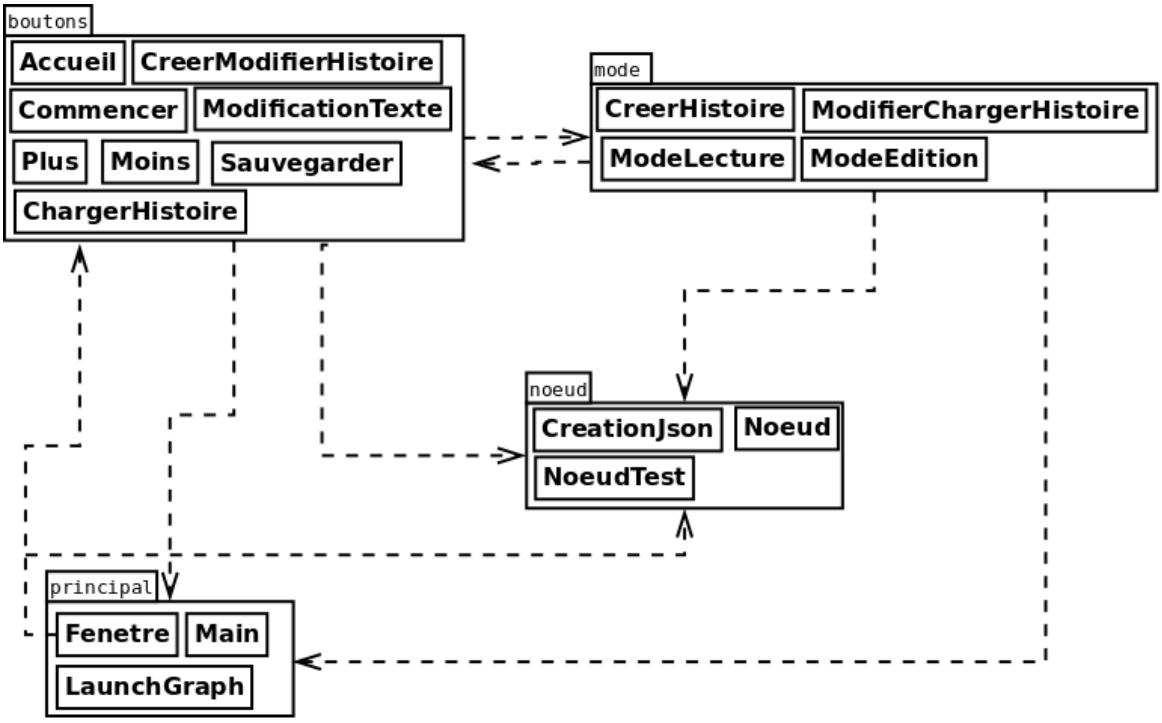
nom librairie	**utilité**
hamcrest-core-1.3.jar	tests
json-simple-1.1.1.jar	gestion d'un JSON
junit-4.13.2.jar	tests

TABLE 1 – Liste des archives .jar utilisées, importées via [?]

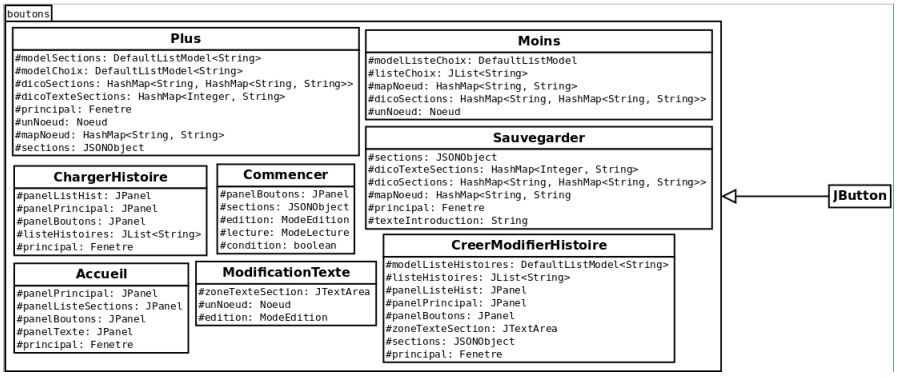
4.2 Algorithmes utilisés

5 Architecture du projet

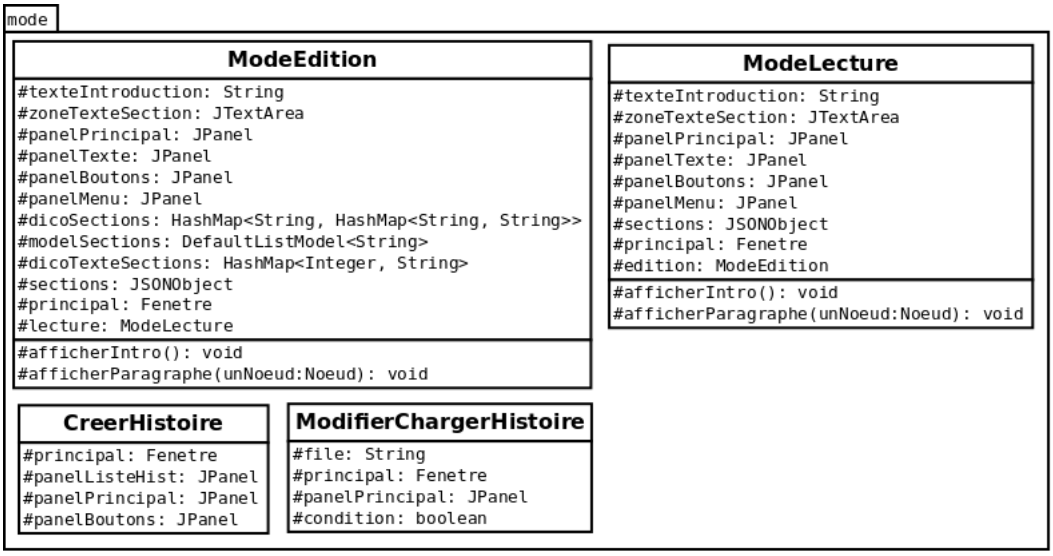
Diagramme de tous les packages (et leurs dépendances) :



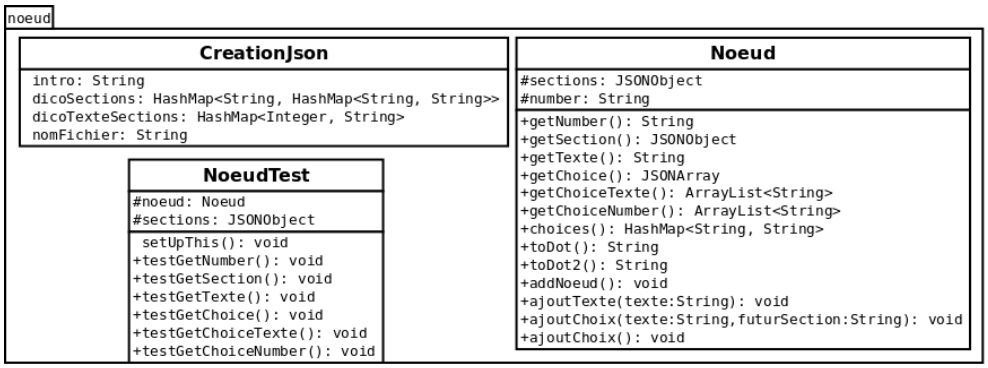
Package *boutons* avec ses classes (attributs) :



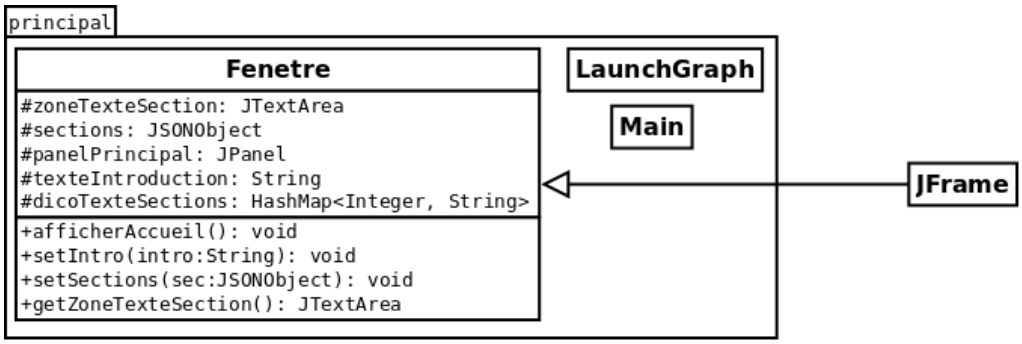
Package *mode* avec ses classes (attributs et méthodes) :



Package *noeud* avec ses classes (attributs et méthodes) :

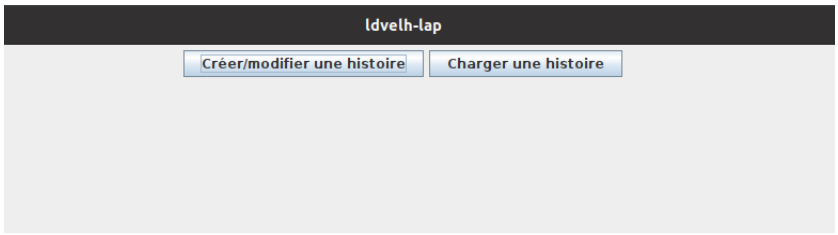


Package *principal* avec ses classes (attributs et méthodes) :



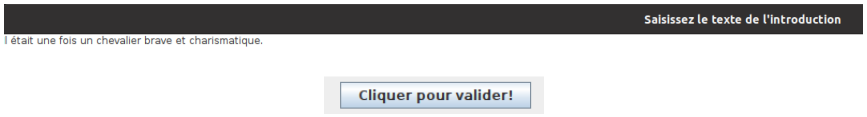
6 Expérimentations

Lorsque vous lancez le jeu, vous arrivez sur une fenêtre d'accueil avec deux boutons :

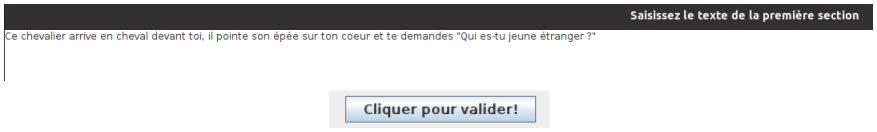


Si vous cliquez sur le bouton "*Créer/modifier une histoire*", vous aurez le choix de créer une histoire à partir de rien, ou de modifier une histoire déjà existante :

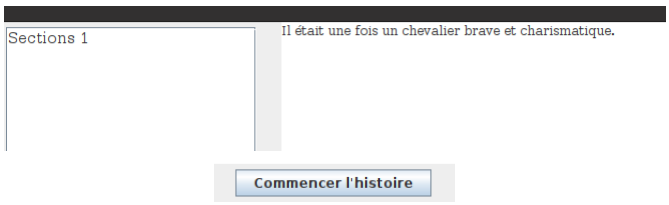
Dans le cas où vous voulez en créer une à partir de rien, une fenêtre va s'ouvrir en vous demandant d'écrire l'intro de votre histoire. Il suffira de cliquer sur le bouton "*Cliquer pour valider*" pour passer à la suite :



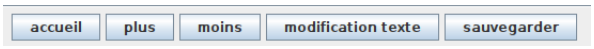
Une autre fenêtre du même type s'ouvrira et vous demandera de saisir le contenu du premier chapitre, avec le même bouton pour valider :



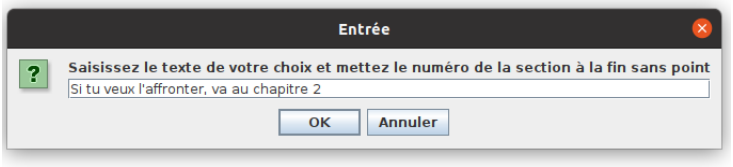
Vous arrivez sur une fenêtre contenant la liste des chapitres à gauche, et le contenu du chapitre en cours à gauche (énoncé + choix). Un bouton "*Commencer l'histoire*" se trouvera en bas de la page pour commencer l'édition :



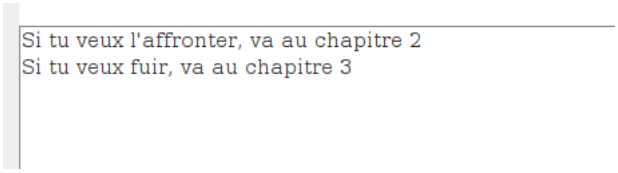
Le bouton "*Commencer l'histoire*" laissera place à 4 boutons : "*accueil*" pour retourner à l'accueil, "*plus*" pour ajouter un choix au chapitre en cours d'édition, "*moins*" pour supprimer un choix après l'avoir sélectionner, "*modification texte*" pour modifier l'énoncer d'un chapitre (il faut d'abord modifier l'énoncé, puis appuyer sur le bouton pour sauvegarder la modification), et enfin, le bouton "*sauvegarder*" qui servira à la fin de la création pour enregistrer l'histoire.



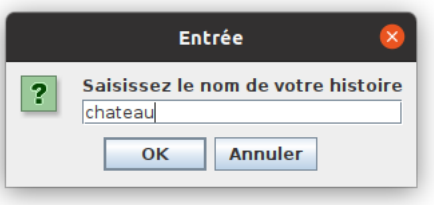
Quand vous appuyez sur le bouton "*plus*", une petite fenêtre s'ouvre et vous demande de saisir votre choix. Il faut que votre choix finisse par le numéro du chapitre voulu :



Voila les choix du chapitre 1 :

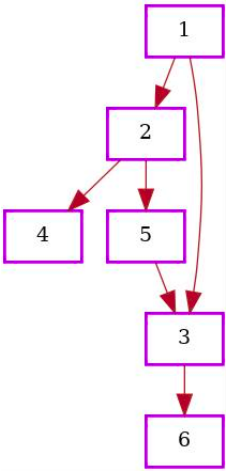


Après avoir écrit votre histoire, vous devez appuyer sur "*sauvegarder*" et une petite fenêtre s'ouvrira en vous demandant de saisir le nom de votre histoire :



Vous pourrez alors retourner à l'accueil, avec le bouton "*accueil*" et la liste des fichiers à charger s'afficheront, dont l'histoire que vous venez d'écrire :

Après avoir joué cette histoire, lorsque vous fermerez la fenêtre, le graphe de votre histoire s'affichera :



7 Conclusion

Nous avons donc un jeu complet avec editeur de texte et algorithmes, c'était bien l'objectif que nous nous étions fixés. Nous n'avons malheureusement pas eu le temps de faire des sytèmes de rencontre mais nous sommes quand même fier de ce que nous avons produit. Nous avons essayé d'implémenter des algorithmes mais la structure de notre classe Noeud ne correspondait pas

avec l'algorithme que nous voulions implémenter. Nous n'avions pas codé d'attribut `voisin`, et `sommet` pour un `noeud`.

Algorithme 1 : Explorer un graphe

Entrées : Un Graphe $\mathsf{G} = \{n_1, n_2, \ldots, n_n\}$ et un noeud n

```
1 marquer n
2 Afficher n
3 ensSommets ← n.getVoisin
4 pour n ← ensSommets faire
5 |   si noeud.etat = false alors
6 |   |   Explorer(G, unVoisin)
7 fin
```

Algorithme 2 : Parcours d'un graphe en profondeur

Entrées : Un Graphe $\mathsf{G} = \{n_1, n_2, \ldots, n_n\}$

```
1 ensSommets ← sommetGraphe
2 pour sommet ← sommetGraphe faire
3 |   si sommet.etat = false alors
4 |   |   Explorer(G, sommet)
5 fin
```