Magic Book

Rapport de projet

DEROUIN Auréline 21806986
MARTIN Justine 21909920
THOMAS Maxime 21810751

Table des matières

1 Présentation du projet

A Présentation de l'application

Magik Book est un éditeur de livre permettant de créer un livre à choix multiples pouvant contenir des conditions pour certains d'entre eux, des choix aléatoires, des combats, etc.
On peut donc créer des paragraphes, appelés des "noeuds", reliés entre eux par des liens. L'application comprend aussi la création d'un pélude, de personnages et d'items.
Une fois le livre créé, nous pouvons alors obtenir une estimation de sa difficulté en choisissant l'option correspondante dans la barre de menu en haut. Cette difficulté est ensuite affichée dans le panel des stats.
Une option est également disponible pour permettre de jouer à l'histoire créé. Enfin, il est également possible d'exporter le livre dans un format texte.
Bien entendu, il est possible d'enregistrer notre livre afin de le réouvrir pour continuer l'édition de celui-ci.

2 Organisation du projet

A Choix des technologies

i Git

Nous avon fais le choix d'utiliser Git comme logiciel de gestion de versions. Quelques-unes des raisons de ce choix sont listées ci-dessous :

- La gestion des branches est efficace
- Logiciel de gestion de versions décentralisé, une interruption de service d'un hébergeur n'empêche pas de continuer le travail et il est facile d'héberger son code sur une autre nouvelle plateforme
- Meilleure gestion des commits et des conflits que SVN

Voici quelques informations supplémentaires concernant notre utilisation de celui-ci.

Branches

Afin d'utiliser au mieux Git, nous avons fait le choix de créer deux branches "principales". Il s'agit de *master* et de *develop*.
La branche *master* correspond à une version stable qui peut être mise en production. Ainsi, on ne travaillera jamais sur cette branche.
La branche *develop*, quant à elle, est donc la branche à partir de laquelle créerons les différentes branches pour le développement de nos fonctionnalités. Ne sont poussées sur celle-ci que les nouvelles fonctionnalités opérationnelles des applications. C'est donc la version en cours de développement.
Les branches créées à partir de *develop* sont donc les branches correspondant aux fonctionnalités développées, elles commencent toutes par *features/* (correspondant à la modification). Par exemple, pour le développement des fourmis, on créera une branche *features/fourmis*.

Nomenclature

Nous avons choisit d'établir et d'utiliser une nomenclature pour les messages de commit. Chaque message est préficé par un mot qui permet d'identifier le type de modification apportée. Nous pouvons par exemple citer l'ajout de fonctionnalités sous le préfixe de *feat*, *fix* pour les corrections de bug, *doc* pour la documentation, etc.

ii Gradle

Gradle est un "build automation system". Il est un équivalent plus récent et plus complet à Maven. Il possède de meilleures performances, un bon support pour de nombreux IDE et permet d'utiliser de nombreux dépots, dont ceux de Maven, pour télécharger les dépendances dont le projet a besoin. Cet outil se révèle pratique car il automatise complètement la réalisation des taches usuels tel que la compilation, l'éxécution et les tests unitaires du code source, etc. Il est également possible de créer ses propres "tasks", afin d'automatiser des actions récurrentes, ou de concevoir et utiliser des plugins pour faciliter la configuration de certains projets (JavaFx11 et plus, Android, ...).

iii JavaFx

Il s'agit d'une technologie plus récente que Swing. De ce fait, beaucoup plus de composants modernes sont disponibles contrairement à Swing. Nous avons fait le choix d'utiliser cette technologie notamment pour élargir nos connaissances sur Java et les librairies usuelles.

B Gestion du projet

Afin de faciliter la communication et le bon déroulement de la conception de notre application, divers moyens ont été mis en oeuvre.

i GitHub et Forge

Bien que nous devions rendre le projet sur la forge, nous avons fait le choix d'utiliser GitHub afin d'héberger et de travailler sur le projet. Ce choix s'est fait au vue de la liste des avantages que cette plateforme apporte :

Webhooks : Ils permettent d'obtenir facilement toutes les informations sur ce qui se passe concernant le dépot. Cela est d'autant plus intéressant que Discord permet d'exploiter ces webhooks.

Pull Requests : Elles permettent de demander une fusion entre deux branches tout en visualisant toutes les mofifications effectués depuis le dernier commit en commun. Cette fonctionnalité nous a notamment été utile pour effectuer les revues de code.

Actions : Il est possible d'exécuter certaines actions, par exemple, lorsqu'un évènement se déclenche. Nous avons utilisé cette fonctionnalité afin de lancer automatiquement les tests unitaires à chaque push et pull request. On était alors prévenu dès qu'ils échouaient.

De plus, grace à git, il suffit simplement d'ajouter une remote vers la forge afin de push les changements sur celle-ci. Cela est d'autant plus pratique que l'entiereté des commits est conservé. Des pushs sur la Forge sont donc réalisés toutes les semaines afin d'actualiser le dépôt. Bien entendu un push final a été réalisé sur la Forge pour rendre le projet.

ii Trello

Concernant la répartition et le "listing" du travail à effectuer, nous avons fais le choix d'utiliser Trello, une plateforme qui nous permet d'utiliser des tableaux pour planifier un projet.

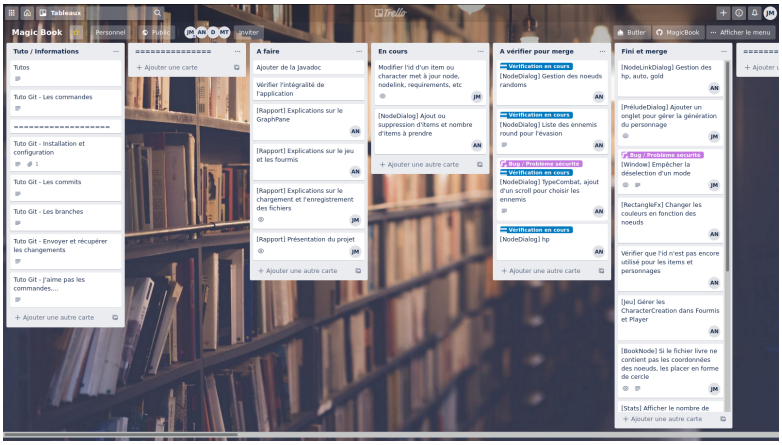

FIGURE 2.1 – Notre tableau Trello

Ainsi, comme nous pouvons le constater, les différentes tâches passent par différents états, "*A faire*", "*En cours*", "*A vérifier*", "*Fini et merge*". Enfin, bien que ce ne soit pas visible sur l'image 2.1, il existe un "*Backlog*" sur la droite qui contient les différentes tâches restantes à accomplir. Celles-ci peuvent ensuite être déplacées dans la colonne "*A faire*" au moment où nous jugeons qu'elles peuvent être réalisées.

Les colonnes "*A verifier*" et "*Fini et merge*" nécessitent quelques précisions,. Pour la première, lorsqu'une tâche est terminée, elle est soumise à évaluation et relecture. Cela permet d'obtenir un avis sur la fonctionnalité et d'éviter d'éventuels bugs par la suite mais aussi de garder une cohérence au travers du code. Raisons pour lesquelles les personnes qui effectuent cette relecture sont souvent les mêmes. Enfin, quand celle-ci est vérifiée et validée, on peut alors merge la branche *feature* dans *develop* la déplacer dans la seconde colonne.

iii Discord

Afin de faciliter la communication au sein du groupe, nous avons utilisé le service de messagerie Discord car tous les membres du groupe l'utilisaient déjà de manière personnelle. Celui-ci permet de parler par le biais de "serveurs" gratuits dans lesquels nous pouvons ajouter des salons textuels ou des salons vocaux à volonté. Ainsi, nous avions trois salons de discussion. L'un nommé "*news-magic-book*" nous permettait d'obtenir toutes les informations sur les push, pull-request, résultats des tests concernant le dépot sur GitHub. "*important-magic-book*" permet de transmettre des messages importants sur ce qui a été fait, sur des changements importants concernant le projet, etc. Enfin, "*dev-magic-book*" était une discution beaucoup plus générale dans laquelle on pouvait demander de l'aide, aider des membres en difficulté, ou même de discuter de certains choix à faire.

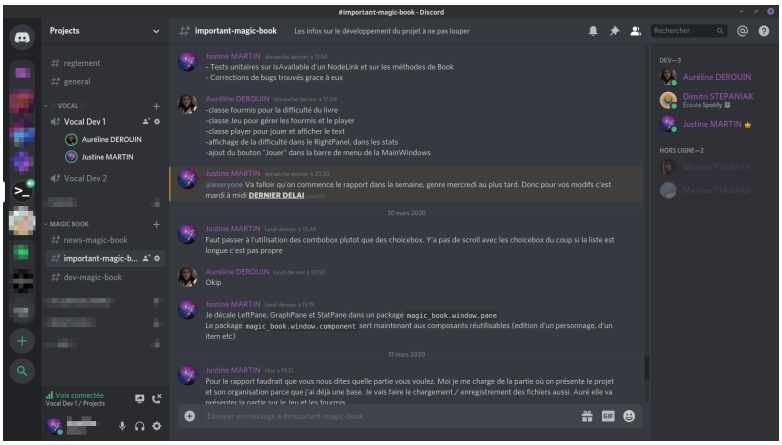

FIGURE 2.2 – Notre serveur Discord

3 Travail de groupe

A Répartition des tâches

Tâches effectués	Auréline	Dimitri	Justine	Maxime
Lecture et enregistrement des fichiers				
Classes pour parser le JSON			X	X
Lecture d'un fichier JSON			X	
Enregistrement d'un fichier JSON			X	
Livre				
Classe Book			X	
Classes pour représenter les noeuds et les liens	X		X	
Classe BookCharacter			X	X
Classes pour les Requirement	X		X	
Classes pour représenter les différents types d'items			X	
Classes pour la "Création du personnage"			X	
Classe pour représenter des skills			X	
Ajout des classes pour le pattern observer (uniquement celles qui concernent le livre)			X	X
Jeu et export au format texte				
Classe Jeu, partie commune au joueur et la fourmis	X			
Classe pour la logique de la fourmis	X			
Classe pour la logique du joueur	X			
Permettre une estimation de la difficulté du livre	X			
Generation du livre en format texte			X	
Classe BookState			X	
Conception d'un Parser pour le texte des liens et des paragraphes			X	
Version primitive de l'estimation de la difficulté d'un livre				X
Fenêtre				
Fenêtre principale		X	X	
Permettre la conception d'un nouveau livre, l'ouverture d'un ancien livre sauvegarder, la sauvegarde et "sauvegarde sous" du livre courant			X	
Lister et permettre l'ajout d'items et de personnages sur le panel de gauche		X		
Permettre d'editer ou supprimer un item ou un personnage du livre			X	
Statistiques concernant les noeuds			X	
Statistique sur le niveau de difficulté du livre	X			
Cacher panel des statistiques si l'on décoche une case dans le menu			X	
Cache le panel de gauche si l'on décoche une case dans le menu				X

Tâches effectués	Auréline	Dimitri	Justine	Maxime
Séparation des différentes parties de la fenêtre en plusieurs classes (LeftPane, GraphPane, RightPane)	X			
Composants réutilisables pour créer des personnages, une phase de la "Création d'un personnage", sélectionner une liste d'items			X	
Boites de dialogues				
Classe mère pour les boites de dialogue	X			
Boite de dialogue pour les noeuds	X			
Boite de dialogue pour les liens entre les noeuds	X			
Boite de dialogue pour les items	X			
Boite de dialogue pour les personnages			X	
Boite de dialogue pour le prélude			X	
Boite de dialogue pour la "Création du personnage"			X	
Boite de dialogue pour le personnage par défaut			X	
Zone d'édition				
Classe pour représenter un noeud graphique	X			
Ajout d'un noeud	X			
Modification d'un noeud	X			
Suppression d'un noeud	X			
Classe pour représenter un lien entre 2 noeuds			X	
Ajout d'un lien entre 2 noeuds (NodeLinkFx)			X	
Un lien suit les noeuds auxquelles il est attaché			X	
Modification d'un lien entre 2 noeuds	X			
Suppression d'un lien entre 2 noeuds	X			
Classe mère commune pour représenter un prélude et un noeud (RectangleFx)			X	
Permettre le déplacement des noeuds	X			
Détecter un clique sur un noeud ou lien (classes observer)	X		X	
Gestion des actions en fonction du mode	X			
Afficher un rectangle qui représentera le prélude			X	
Gestion du texte de prélude			X	
Gestion du personnage par défaut			X	
Gestion de la "Conception du personnage"			X	
Changer le premier noeud du livre			X	
Répartition des différents noeuds lors de l'ouverture d'un fichier			X	
Gestion du niveau de zoom			X	
Rend le GraphPane scrollable			X	
Change la couleur d'un noeud en fonction de son type (normal, aléatoire, combat, victoire, ...)	X			
Mettre en valeur un noeud lorsque l'on passe la souris dessus	X			
Autre				
Rapport	X		X	$\sim$
Restructuration du livre d'exemple (fotw.json)			X	
Création de tests unitaires	X		X	
Javadoc	X			

Tâches effectués	Auréline	Dimitri	Justine	Maxime
Revue de code avant de merge			X	

Nous avons décidé de ne pas inclure le graphique de Forge concernant le nombre de lignes de code commités par personnes. En effet, l'ajout de Gradle et du livre d'exemple font à eux seul 10 000 lignes. De plus, ce livre à été restructuré. De ce fait, après vérification, 17 150 lignes ajoutés, et 10 460 lignes supprimés sont données à la personne qui les ont commit, Justine dans notre cas.

Après avoir lancé un script (git-stats) pour connaitre le nombre de lignes par commit de chacun, voici ce que donnerait le graphique si l'on enlevait toutes ces lignes à Justine :

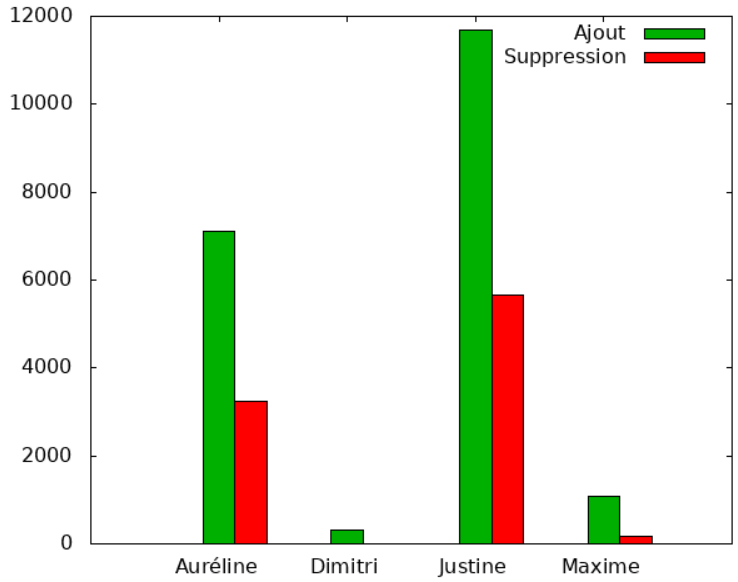

FIGURE 3.1 – Nombre de lignes d'ajoutés et supprimés par personnes

B Idées d'améliorations

Notre application n'ayant pu être terminé faute de temps, voici la liste des améliorations que nous aurions voulus faire et celles qui seraient possibles d'implémenter ensuite :

- Concevoir deux types de fichier, l'un pour l'éditeur et l'autre pour le jeu. Le jeu serait une version épurée de celui de l'éditeur et ne contiendrait pas la position des noeuds par exemple
- Une mise à jour d'un noeud transfert correctement les différents liens (au lieu de les supprimer dans la plupart des cas)
- Vérifier que le livre est valide pour être joué
- Créer une classe mère pour les listes sur le côté gauche de l'application (Item et personnage)
- Une fois la classe mère codé, ajouter une liste à gauche pour gérer les skills dans l'éditeur
- Déclencher plus d'exception si le livre est incorrect
- Gérer les shops(jeu et gui), champs auto (jeu uniquement)
- Afficher les personnages et items inutilisés
- Indiquer si l'estimation de la difficulté est à jour ou non
- Gestion des prérequis sur les boites de dialogue des liens
- Améliorer l'intelligence de la fourmis (pouvoir estimer si un item est plus important qu'un autre, meilleure gestion des combats, ...)
- Ajouter et supprimer des skills au fil du jeu
- Ajout de paramètres aux skills (plutot que d'avoir un simple nom)

- Afficher les chemins gagnants
- Enlever ou ajouter une somme d'argent à un personnage se fait sur une monnaie précise (ex : -5 dollards, +15 euros, etc)
- "Langage" simple permettant de manier des conditions et variables pour des prérequis notamment
- Possibilité d'avoir des pnj qui pourraient nous suivre dans l'aventure pour combattre ou pour dévérouiller certains passages par exemple.

C Bugs et problèmes connus

Certains et problèmes sont connus, en voici une liste non exhaustive une fois de plus :

- Tests incomplets sur le Book et le Jeu
- Le changement d'id d'un personnage ou d'un item ne met pas à jour les différents élements du livre (noeuds, choix, ...)
- Diverses bugs visuels concernant la boite de dialogue sur le Prélude
- Le zoom ne se fait pas selon la position actuel de la souris mais du point supérieur gauche du GraphPane

4 Architecture du projet

A Arborescence du projet

.github : Fichiers spécifiques à GitHub.

workflows Fichiers destiné au module d' "Actions" de GitHub. Nous nous en somme servis pour lancer automatiquement les tests unitaires lors d'un push ou d'une pull-request.

app : Contient tout le code source de notre application.

gradle : Wrapper de gradle.

livre : Exemples de livre.

src : Contient les codes sources, ressources et tests unitaires.

main : Code principal de l'application.

java : Code source.

resources : Ressources pour l'application (images, musiques, ...).

test : Code des tests unitaires.

java : Code source.

resources : Ressources utiles pour les tests (images, musiques, config, ...).

build.gradle : Script de configuration du projet (dépendance, classe principale, ...).

gradlew : Script pour les systèmes Unix afin d'exécuter le Wrapper de Gradle.

graldlew.bat : Script pour les systèmes DOS afin d'exécuter le Wrapper de Gradle.

settings.gradle : Configuration sur les modules à inclure, les noms de ceux-ci, etc.

.gitattributes : Permet de fixer la fin de ligne pour les scripts Unix et DOS.

doc : Contient toute la documentations du projet, notamment le rapport.

.gitignore : Fichier ignorant les changements sur certains fichiers ou dossier sur Git.

CONVENTIONS.md : Conventions de nommage concernant le projet et les commits.

LICENSE : Licence du projet.

README.md : README pour présenter notre projet et expliquer la compilation de celui-ci.

Pour mieux comprendre la structure de gradle les liens suivants sont utiles https://guides.gradle.org/creating-new-gradle-builds/ et https://docs.gradle.org/6.3/userguide/gradle_wrapper.html

B Présentation des packages

Notre application contenant beaucoup de classes, celles-ci sont réparties en packages que nous allons détailler :

core Classes principales de l'application

exception Classes d'exceptions

file Classes utiles à la lecture, l'écriture de fichier (json et texte)

deserializer Classes qui héritent de JsonDeserializer (provient de GSON)

json Classes JSON intermédiaire pour la lecture et l'écriture avec GSON

game Classes spécifiques au jeu (Personnage, Skill, BookState, ...)

character_creation Classes qui représentent une étape de la "Création du personnage"

player Classes qui permettent de jouer au jeu (Joueur ou fourmis)

graph Classes qui représentent les noeuds et les liens

node Classes pour les noeuds

node_link Classe pour les liens

item Classes qui représentent les items

parser Classes qui permettent de parser un texte pour afficher le nom de l'item ou du personnage

requirement Classes pour gérer les prérequis sur un noeud

observer Classes pour le pattern observer

book Classes pour le pattern observer du livre

fx Classes pour le pattern observer des éléments JavaFx

window Classes pour l'affichage avec JavaFx

component Composants réutilisables à différents endroits (dans plusieurs boites de dialogues par exemple)

dialog Les différentes boites de dialogue

gui Les différents éléments graphiques pour JavaFx (NodeFx, NodeLinkFx, PreludeFx)

pane Les différentes parties qui composent notre affichage sur la fenêtre (Partie de gauche, centrale, droite)

5 Aspects techniques

A Représentation d'un livre

i Représentation des noeuds et des liens

ii Quelques algorithmes

B Lecture et écriture d'un livre

L'objectif de l'application étant de concevoir un éditeur, il était important de permettre la sauvegarde et la lecture du livre que l'on édite. Le choix du format JSON est rapidement survenue. Premièrement car un fichier d'exemple qui nous a été fournis était sous ce format mais aussi car il s'agit d'une structure simple et très facile à lire. Nous avon alors utilisé GSON, une librairie, conçue par Google, extrêmement simple. Elle permet de retranscrire sous forme d'objet Java un fichier JSON structuré, c'est à dire où l'on distingue très clairement des objets qui se répète.

Afin de lire un fichier JSON, avec cette librairie, il suffit de concevoir des objets Java avec les même attributs que ceux du fichier à lire ou à écrire. Voici un exemple très court de ce à quoi nos fichiers ressembles :

i La structure du JSON

```
{
    "prelude": "Vous êtes l'enseignant qui note notre projet",
    "setup": {
        "skills": [],
        "items": [],
        "characters": [],
        "character_creation": []
    },
    "sections": {
        "1" : {
            "text": "Vous être en train d'étudier notre projet",
            "choices": [
                {
                    "text": "Mettre une bonne note",
                    "section": 3
                },
                {
                    "text": "Mettre une mauvaise note",
                    "section": 2
                }
            ]
        },
        "2": {
            "text": "Les étudiants du projet sont tristes",
            "end_type": "FAILURE"
        },
        "3": {
            "text": "Les étudiants sont satisfait de leur travail",
            "end_type": "VICTORY"
        }
```

```
31      }
32  }
```

Listing 5.1 – Exemple de livre très simple

On retrouve plusieurs éléments différents. On remarque par exemple un attribut "prelude", ainsi que deux grosses parties, "setup" et "sections". Dans la suite, nous détaillerons uniquement les attributs les plus frequemment présents.

Setup

Commençons par détailler "setup". Ce passage contient toutes les informations générales à notre livre. On y retrouve la liste des compétences ("skills"), la liste des items ("items") et la liste des personnages ("characters"). "character_creation", lui, détaille toutes les étapes lors de la conception du personnage qui intervient au tout début. Celle-ci permet de sélectionner des skills et items de départ.

Pour le moment les compétences sont uniquement composé d'un id et d'un nom. Dans une future mise à jour il serait intéressant d'ajouter des propriétés pour connaitre la force ajouté dans un combat, la quantité de soins à rendre par noeuds, par exemple.

```
1  {
2      "id": "sixth_sense",
3      "name": "Sixième sens"
4  }
```

Listing 5.2 – Exemple de compétence

Les items peuvent être de différents types : KEY_ITEM, WEAPON, DEFENSE, MONEY, HEALING. On retrouve pour tous les items un id et un nom ("name"). Pour certains types, des attributs supplémentaires sont présent. Par exemple, un attribut "durability" peut être présent. Il permet de déterminer le nombre d'utilisation maximum d'un item. Un item de type HEALING possède un nombre de pv à rendre ("hp") tandis que ceux type WEAPON possède un montant de dégats ("damage") par exemple.

```
1  {
2      "id": "backpack",
3      "name": "Backpack",
4      "item_type": "KEY_ITEM"
5  },
6  {
7      "id": "healing_potion_4",
8      "name": "Potion de soins (4HP)",
9      "hp": 4,
10     "durability": 1,
11     "item_type": "HEALING"
12 }
```

Listing 5.3 – Exemple d'items

Concernant les personnages on y retrouve un id, un nom ("name"), un nombre de pv maximum ("hp"), un boolean pour indiquer s'il a beaucoup de chance que ses coups fassent le double des dégats ("double_damage"), ainsi que "combat_skill" qui représente le montant de ses dégats.

```
1  {
2      "id": "zombie_captain",
3      "name": "Zombie Captain",
4      "hp": 15,
5      "double_damage": true,
6      "combat_skill": 2
7  }
```

Listing 5.4 – Exemple de personnage

Les character character_creation peuvent être de simple texte ou de type "ITEM" ou "SKILL". On y retrouve les différents skills ou items que l'on peut prendre pour débuter notre aventure ainsi que le nombre que l'on doit en choisir ("amount_to_pick").

```
{
    "text": "Kai Disciplines\n\nOver the centuries, the Kai monks have mastered
    the skills of the warrior. These skills are known as the Kai Disciplines,
    [...]",
    "type": "SKILL",
    "skills": [
        "camouflage",
        "hunting",
        "sixth_sense",
        "tracking",
        "healing",
        "weaponskill",
        "mindshield",
        "mindblast",
        "animal_kinship",
        "mind_over_matter"
    ],
    "amount_to_pick": 5
}
```

Listing 5.5 – Exemple de character_creation

Sections

La partie "sections" est une map qui représente le numéro d'un paragraphe ainsi que le paragraphe associé. Il existe différents types de paragraphes, à choix, à choix aléatoire, avec des combats, terminaux. Tous possède un texte. Les noeuds terminaux possèdent un type de fin ("end_type") afin savoir si l'on a gagné ou pas (cf : Listing 5.1). Les noeuds aléatoires eux, possède un attribut "is_random_pick" qui vaut true. Pour tous les autres types de noeuds, on retrouve parmis les attributs les plus importants une liste d'items qu'il est possible de prendre, un montant d'item maximum qui peut être pris ("amount_to_pick"), des items disponibles à l'achat ("shop").

```
{
    "text": "The back door opens [...]",
    "items": [
        {
            "id": "gold",
            "amount": 5
        },
        {
            "id": "dagger"
        },
        {
            "id": "seal_hammerdal"
        }
    ],
    "amount_to_pick": 2
    "choices": [
        {
            "text": "Return to the tavern.",
            "section": "177"
        },
        {
            "text": "Study the tomb.",
            "section": "24"
        }
    ]
```

```
}
```

Listing 5.6 – Exemple de paragraphe

Certains paragraphes peuvent contenir un attribut "combat". Dès lors on peut connaitre le choix en cas de victoire ("win"), de défaite ("loose") ou d'évasion ("evasion). Si l'évasion est possible seulement à partir d'un certains nombre de tour on retrouve alors un attribut nommé "evasion_round". Pour finir, un attribut "enemies" permet de connaitre les personnages que l'on combat.

```
{
    "text": "The dead zombies lie [...]",
    "combat": {
        "win": {
            "text": "If you win the combat.",
            "section": "309"
        },
        "enemies": [
            "zombie_captain"
        ]
    }
}
```

Listing 5.7 – Exemple de paragraphe avec des combats

Pour représenter un lien vers un autre paragraphe on retrouve une liste de choix ("choices"). Ils possèdent également un texte qui correspond à l'intitulé du choix, le numéro du paragraphe suivant ("section"), un nombre d'hp à retirer, un nombre d'argent à ajouter ainsi qu'un liste de prérequis ("requirements"). Comme pour les BookNodeLink, il s'agit d'un tableau à deux dimensions. Le premier représente une liste de condition en OU et le second une liste de condition en ET. Enfin, pour les noeuds aléatoire, un poid est également présent ("weight").

```
{
    "text": "If you have the Kai Discipline of Tracking.",
    "section": "182",
    "hp": -5,
    "requirements":  [
        [
            {
                "id": "tracking",
                "type": "SKILL"
            }
        ]
    ]
}
```

Listing 5.8 – Exemple de choix

ii La lecture et l'écriture

Du fait que la structure en Json n'est pas identique à celle détaillé dans Représentation d'un livre (page 11), nous avons fait des classes intermédiaires pour permettre cette lecture. Celles-ci sont disponibles dans le package *magic_book/core/file/json* et ne contiennent rien de plus que des getter et setter. Aussi, afin de permettre une convertion entre les classes faites pour représenter un fichier json et celles faites pour être utilisées par l'application, une interface *JsonExportable* existe. Celle ci permet de redéfinir 2 méthodes. L'une renvoyant la classe JSON associé à notre classe actuelle, l'autre permettant à partir d'une classe JSON d'obtenir la classe Java correspondante.

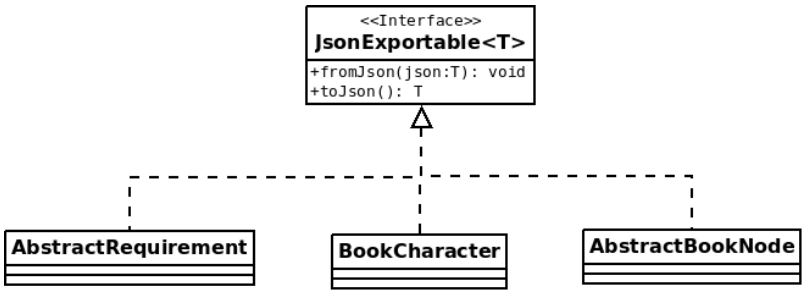

FIGURE 5.1 – L'interface JsonExportable et quelques classes qui l'implémentent

Enfin, les classes BookReader et BookWritter permettent de récupérer toutes les classes JSON intermédiaires pour les regrouper dans le BookJson qui correspond à la structure complète de notre livre. Ces classes sont également une couche d'abstraction à GSON car c'est elles qui se chargent d'écrire le JSON correspondant dans un flux.

Pour résumer, on peut schématiser ces échanges de telle sorte :

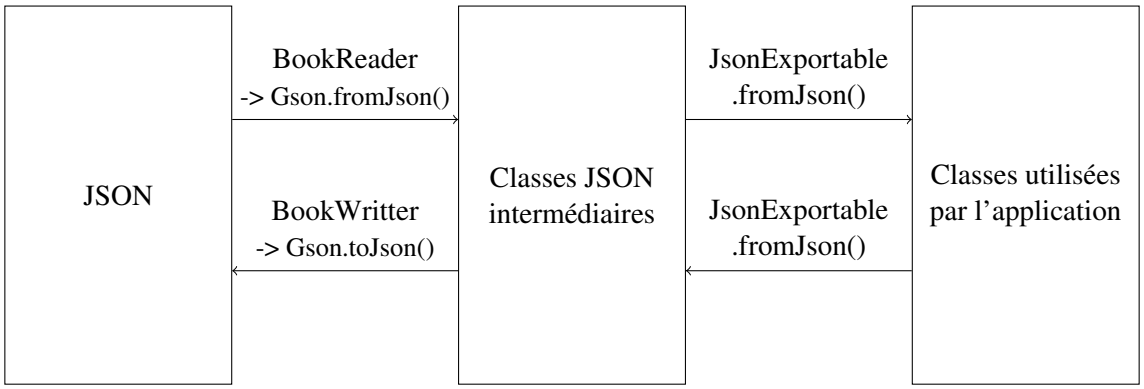

FIGURE 5.2 – Échanges pour la lecture / écriture

C Edition d'un livre

D Rendre le livre jouable et estimer sa difficulté

Jeu Une classe à été créer se nommant **Jeu**, permettant de gérer les méthodes de jeu communes entre le *Player* et les *Fourmis*.
Un construteur est d'abord appelé afin d'avoir le livre commun à toute les classes. Puis, celui le mode sélectionner ("Générer la difficulté"" ou "jouer"), on fait appels à la méthode correspondante au player. Une fois quele mode à été cliqué, le livre est alors copié afin de ne pas le modifier dans la classe au cas où. Un BookState, correspondant à la sauvegarde de la partie, est alors créer à partir du BookCharacter généré par le prélude. C'est donc le personnage principal. Si aucun personnages n'est créer, alors un personnage lambda va étre créer afin de pouvoir jouer au jeu.
Une fois le BookState créer et la copie du livre enregistrer, on prend le premier paragraphe et on regarde à quel "noeud" il appartient. Une méthode sera ainsi appeler en fontion du type de noeuds qui prend en charge.
La méthode correspondante au type de noeud s'exécute et renvoie le noeud de "destination", en fonction du choix du player, ou de la mort du player. En effet, ces "noeuds" peuvent faire venir la mort du player en enlevant de la vie par exemple, ou que ce player tombe dans une embuscade... Ces noeuds offre beau-

coup de possibilité.

Durant l'exécution de la méthode, et en fonction du player, d'autre méthode externe sont appeler, nottament dans la classe Fourmis ou Player.

Interface Player / Foumis Une interface **InterfacePlayerFoumis** à été créer permettant une mise en commun des codes Player et Fourmis. Ces classes permettent de faire un choix, prendre les items disponibles, créer un personage lambda, aller dans l'inventaire, choisir son ennemis ou encore combatre.
Elles permettrent de d'appeler la même méthode (que cela soit fourmis ou player) au même moment. La méthode sera alors exécuté différément en fonction du player. Cela permet donc une harmonie du code

Player La classe **Player** permet de jouer au jeu en tant que joueur. Elle permet de faire des choix grâce aux Scanner.
Cette classe a des méthodes de l'interface, notamment celle de combatChoice qui prend en paramètre le noeud de Combat, le nombre de tour avant l'évasion ainsi que le BookState. Cette méthode permet de choisir nos choix lors de notre tour dans le combat. On peut alors choisir d'attaquer, d'aller dans notre inventaire ou alors de s'évader.
Si on choisi l'inventaire, on va alors dans une autre méthode appelé useInventaire() qui prendre le BookState en parametre. On peut alors utiliser une potion, prendre un objet de défense ou alors une arme. Si l'on choisis un autre choix, cette objet n'est pas utilisable lors d'un combat (comme par exemple de l'argent). Une fois l'objet pris, on retourne dans les choix du combat. On peut alors, soit retourner dans l'inventaire pour prendre un autre objet, soit attaquer ou s'évader.
Si le choix évasion est choisi, un message apparait si le nombre de tour avant l'évasion n'est pas à zero. Si il n'est pas à zéro, un message apparait et il doit refaire un autre choix. Sinon, il va alors dans le noeud de destination qui a été prévu pour l'évasion.
Si le choix attaque est choisi...

Fourmis

6 Conclusion

Ce projet a permis aux différents membres de s'améliorer, que ce soit dans l'apprentissage d'une librairie, la mise en application des concepts vus en cours ou bien encore, dans l'art de la procrastination. Bien que nous ayons été quatre sur le projet, deux de nos camarades n'ont pas fourni suffisamment d'aide dans celui-ci. Il a fallu leur demander à de nombreuses reprises de travailler, des modifications de quelques lignes pouvaient prendre jusqu'à trois semaines pour être rendues, tout en étant parfois incomplètes. Ainsi, pour nous, le groupe était un groupe constitué de deux personnes uniquement. Le projet a été très intéressant mais nous regrettons beaucoup de ne pas avoir pu le compléter et de ne pas avoir rendu le code aussi propre que ce que nous aurions voulu. Nous sommes, par exemple, déçue de ne pas avoir pu fournir certaines fonctionnalités qui sont prises en compte dans le jeu mais pas dans l'éditeur (gestions des skills, des prérequis, ...). L'application reste cependant fonctionnelle et plutôt complète.

7 Ressources utiles et sources utilisés