



UNIVERSITÉ
CAEN
NORMANDIE

Magic Book

Rapport de projet

DEROUIN Auréline 21806986
MARTIN Justine 21909920

Table des matières

1	Présentation du projet	1
A	Présentation de l'application	1
2	Organisation du projet	2
A	Choix des technologies	2
i	Git	2
ii	Gradle	3
iii	JavaFx	3
B	Gestion du projet	3
i	GitHub et Forge	3
ii	Trello	4
iii	Discord	4
3	Travail de groupe	6
A	Répartition des fonctionnalités	6
B	Idées d'améliorations	6
C	Bugs connus	7
4	Architecture du projet	8
A	Arborescence du projet	8
B	Présentation des packages	8
5	Aspects techniques	9
A	Jeu,Player,Foumis	9
B	Edition du livre	10
6	Conclusions	11
A	Éléments à améliorer	11
B	Avis personnels	11
7	Ressources utiles et sources utilisés	12

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 avons fait le choix d'utiliser Git comme logiciel de gestion de versions. Les 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
- 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. Elle ne nous servira donc qu'à récupérer l'application dans un état stable afin d'y mettre les différentes applications en production.

La branche *develop*, quant à elle, est donc la branche à partir de laquelle nous travaillons. C'est à partir de cette dernière que nous 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/XXX* (correspondant à la modification). Par exemple, pour le développement des fourmis, on créera une branche *features/fourmis*.

Nomenclature

Nous avons choisi d'établir et d'utiliser une nomenclature pour les messages de commit. Chaque message est préfixé 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.

Ce qui donne des messages comme celui-ci : *feat : Rend le GraphPane scrollable*.

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 dépôts, 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 tâches usuelles tel que la compilation, l'exécution 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 bibliothèques usuelles. Au final, l'avis des membres du groupe qui ont suffisamment codé avec la bibliothèque, est que Swing est beaucoup plus pratique à utiliser que Swing. Les méthodes y sont plus claires et agréables à manier que Swing.

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 vu 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 dépôt. 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 modifications effectuées 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 événement 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, grâce 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'entièreté des commits est conservée. Des pushes 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 fait le choix d'utiliser [Trello](#), une plateforme qui nous permet d'utiliser des tableaux pour planifier un projet.

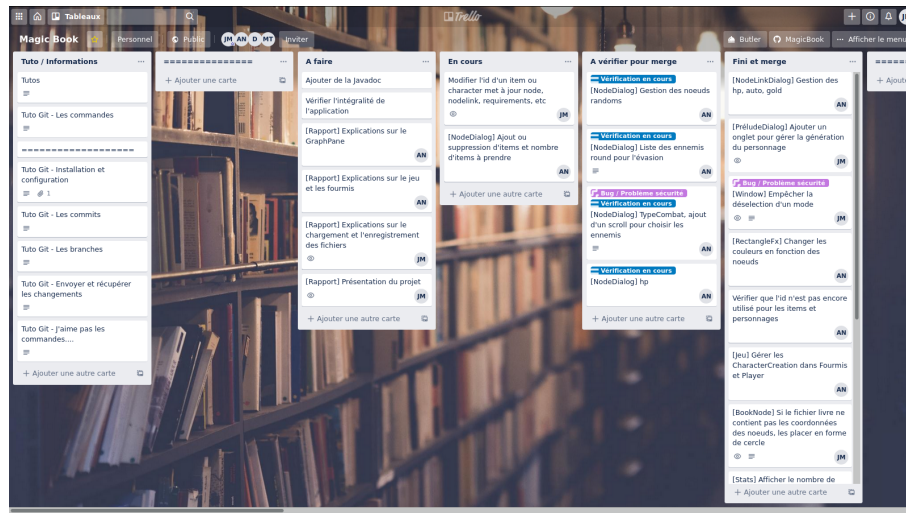


FIGURE 2.1 – Notre tableau Trello

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 vérifier" 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 avons 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 dépôt 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 discussion 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.

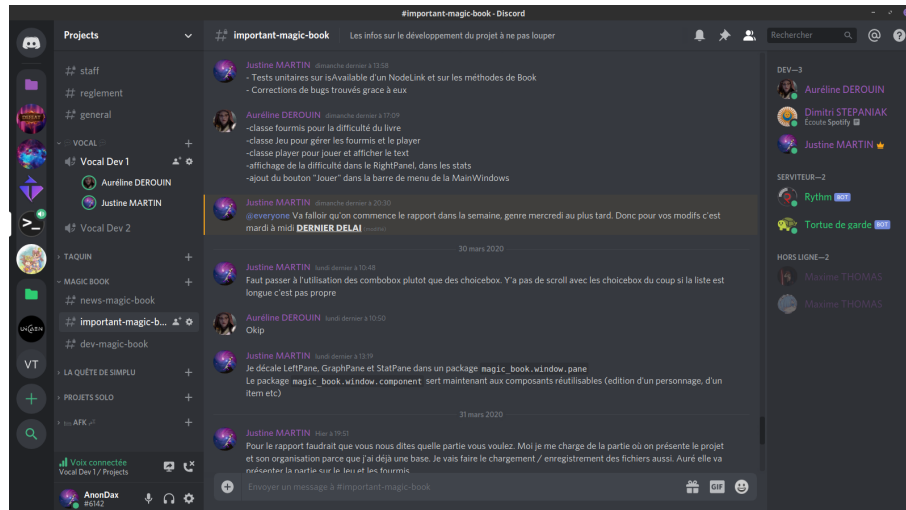


FIGURE 2.2 – Notre serveur Discord

3 Travail de groupe

A Répartition des fonctionnalités

Tâches effectuer		
Justine MARTIN	Prélude	test
	Pannel des stats	test
	BookEditor	test
	Enregistrer et lecture d'un fichier Json	test
	création d'un livre test	test
	mise en place d'un arc de cercle lors de l'affichage des noeuds	test
	zoom sur l'affichage principal	test
	Supression d'un noeud	test
	Création de test unitaire	test
	Correction des codes avant de merges	test
Auréline DEROUIN	Classe Jeu/Fourmis/Player	test
	Mis en place du GraphPane	test
	Mis en place des boites de dialog	test
	Création des classes de BookNode	test
	Supression d'un noeud	test
	test	test
	test	test
	test	test
	test	test
	test	test
Maxime THOMAS	test	test
	test	test
Dimitri STEPANIAK	test	test
	test	test

B Idées d'améliorations

C Bugs connus

4 Architecture du projet

A Arborescence du projet

B Présentation des packages

5 Aspects techniques

A Jeu, Player, Fournis

Jeu Une classe à été créer se nommant **Jeu**, permettant de gérer les méthodes de jeu communes entre le *Player* et les *Fournis*.

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 que le 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 préluide. C'est donc le personnage principal. Si aucun personnages n'est créer, alors un personnage lambda va être 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 fonction 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 beaucoup de possibilité.

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

Interface Player / Fournis Une interface **InterfacePlayerFournis** à été créer permettant une mise en commun des codes *Player* et *Fournis*. Ces classes permettent de faire un choix, prendre les items disponibles, créer un personnage lambda, aller dans l'inventaire, choisir son ennemis ou encore combattre. Elles permettent de d'appeler la même méthode (que cela soit *fournis* ou *player*) au même moment. La méthode sera alors exécuté différemment 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 choisit 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...

Fournis

B Edition du livre

6 Conclusions

A Éléments à améliorer

B Avis personnels

7 Ressources utiles et sources utilisés