



UNIVERSITÉ
CAEN
NORMANDIE

Magic Book

Rapport de projet

MARTIN Justine 21909920
DEROUIN Auréline 21806986

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

1 Présentation du projet

A Présentation de l'application

Magik Book est un éditeur de livre. Il permet donc de créer un livre à choix, avec des conditions pour certains choix.

On peut donc créer des paragraphes, appeler des "noeuds", de différents types : choix simple, choix de chances (random), combat, terminaux (victoire ou défaite). Cette application comprend aussi la création d'un prélude ainsi que des personnages et d'items.

Une fois le livre créé, nous pouvons alors regarder sa difficulté en choisissant dans la barre de menu en haut. Cette difficulté est affichée alors dans le panel des stats. Un bouton pour jouer est également disponible afin de pouvoir profiter pleinement de l'histoire créée.

Nous pouvons enregistrer notre livre en format json et le réouvrir afin de pouvoir continuer l'édition de ce dernier.

2 Organisation du projet

A Choix des technologies

i Git

ii Gradle

iii JavaFx

B Gestion du projet

i GitHub et Forge

ii Trello

iii 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