Magic Book

Rapport de projet

MARTIN Justine 21909920
DEROUIN Auréline 21806986

Table des matières

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 aplication comprend aussi la création d'un pélude ainsi que des personnages et d'items.
Une fois le livre créer, nous pouvons alors regarder sa difficulté en choisissant dans la bar des menu en haut. Cette difficulté est affiché alors dans le panel des stats. Une bouton pour jouer est également disponible afin de pouvoir profité pleinement de l'hitoire créer.
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

.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

5 Aspects techniques

A Jeu,Player,Foumis

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 beaucoup 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

B Edition du livre

6 Conclusions

A Éléments à améliorer

B Avis personnels

7 Ressources utiles et sources utilisés