Lina Borg
Delphine Chichery
Margaux Khatchadourian
Marie-Sara Mazen

SUJET 4

CAHIER DES CHARGES

1. Choix du projet

Parmi les sujets proposés, le sujet 4 nous a particulièrement intéressé. En effet, nous avons trouvé qu'il laissait part à beaucoup d'imagination par rapport aux autres sujets proposés.

Nous étions également inspirées, car il nous rappelait des jeux auxquels nous avions déjà joué pour certaines, comme Magic the Gathering, donc beaucoup d'idées nous venaient rapidement en tête.

La programmation du jeu nous paraissait également intéressante, car elle pouvait contenir une multitude de méthodes déjà vues en cours.

2. Organisation en équipe

Nous avons décidé dès la première séance d'utiliser l'outil GitHub pour s'organiser, car Lina et Marie-Sara le connaissait déjà. Elles nous ont donc expliqué son utilisation et nous nous sommes ensuite divisées en binômes pour coder.
Pour chaque modification ou groupe de modification, une branche était créée afin que celle-ci soit validée par l'autre binôme pour ensuite pouvoir être mergée dans la branche master, qui était d'ailleurs protégée (on ne pouvait pas merger sur master tant qu'une review du code ait été faire par un autre membre du groupe).
Nous avons donc choisi de coder au fur à mesure le code, partie par partie, et non de répartir le travail en fonction des méthodes, car nous ne savions pas encore combien de méthode et quelles méthodes nous allions faire exactement.
Il nous paraissait donc judicieux de compléter le code petit à petit sans avoir de réelle grande partie de code attribuée à chaque binôme.

3. Explication du jeu

Basé sur l'univers de Magic The Gathering, on a changé les règles (un peu, beaucoup, beaucoup trop) si bien que ça ne ressemble plus vraiment au jeu de base.

La partie commence avec 20 PV (points de vie) et le but du jeu c'est de tuer ton adversaire avant qu'il ne te tue (PV<=0), en attaquant avec des créatures.

Déroulement de la partie

Le joueur dispose donc de créatures, dotées d'une force, d'un prix et d'un numéro de probabilité de réussite (qui servira pour la phase d'attaque).

A chaque tour, on gagne du mana (c'est comme de l'argent) qui va servir à payer les coûts des créatures, qui sont pour l'instant dans la main (pioche). On gagne un nombre de mana égal au numéro du tour du joueur (et donc 1, puis 2, puis 3...). Le mana qui ne sera pas dépensé est conservé pour le tour d'après.

Dès lors que l'on paye le coût en mana d'une créature, celle-ci se rend sur le champ de bataille. On peut payer autant de créatures que l'on veut dans la mesure où l'on dispose d'assez de mana. Si tu on ne veut pas payer de créature pour économiser le mana, on peut bien évidemment le faire.

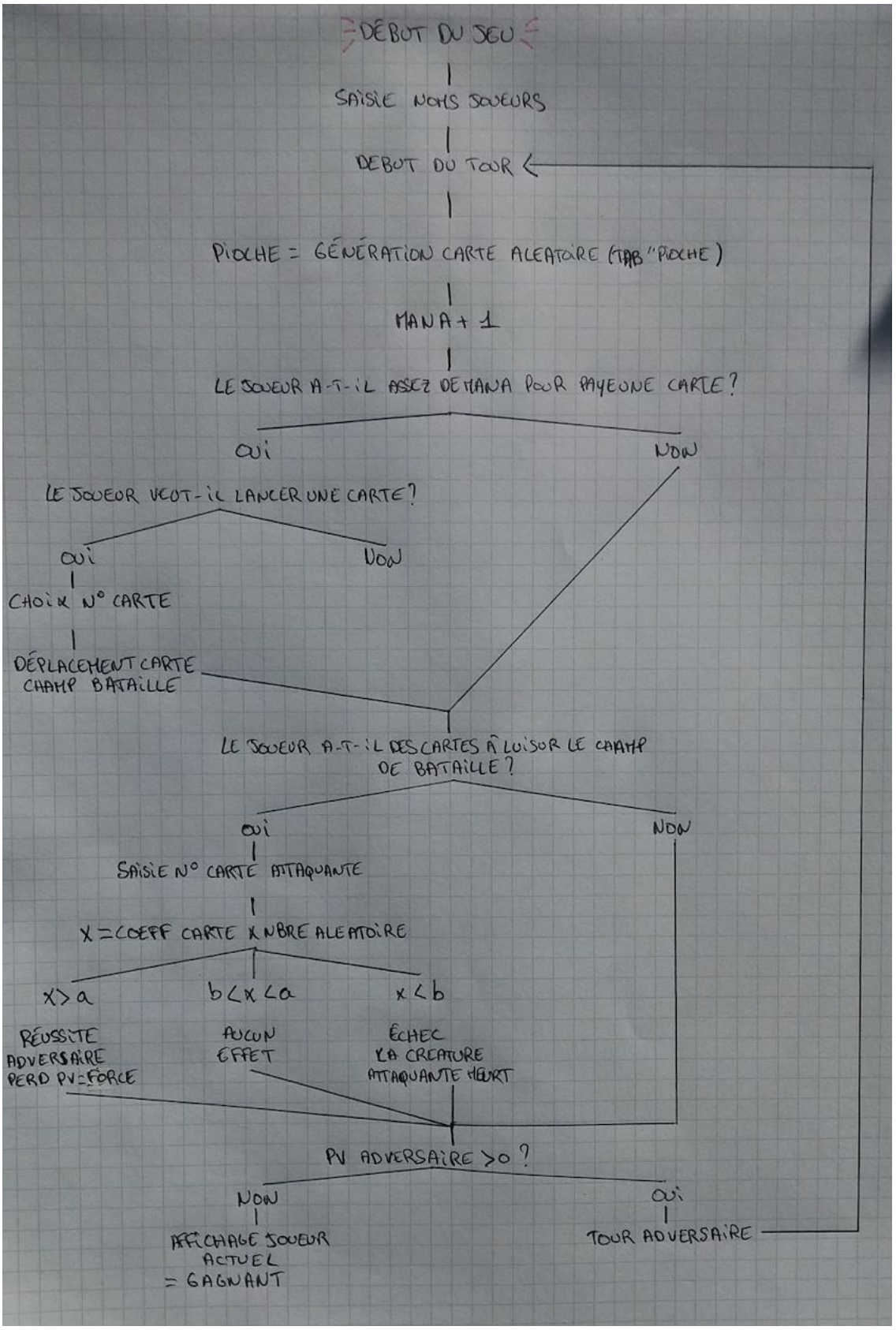
Une fois avoir lancé les créatures que l'on veut, la phase d'attaque peut commencer. On peut attaquer avec 3 créatures maximum. La réussite de l'attaque repose sur le hasard: pour chaque créature attaquante, un numéro aléatoire est généré. En fonction du numéro de probabilité de réussite de la créature, et d'un nombre généré aléatoirement pour l'attaque, celle-ci réussit ou non. Plus la créature est petite, plus elle a de chances de réussir son attaque.

3 cas sont possibles:

- l'attaque réussit, et l'adversaire perd un nombre de PV égal à la force de la créature
- L'attaque échoue, mais il ne se passe rien
- L'attaque est un véritable échec, et non seulement l'adversaire ne perd pas de PV, mais en plus cette créature d'auto-détruit.

Les créatures qui ont survécu sont conservées pour les tours suivant sur le champ de bataille.

Pour construire la première version du jeu, on est parti de ce schéma, il nous a permi de mieux visualiser les méthodes à faire, et l'organisation générale de la partie.



On a commenté le code en suivant les règles de la javadoc, ce qui nous a permis d'avoir accès à des fichiers html de tableaux résumant les différents attributs et méthodes utilisées.

En voici un aperçu (toutes nos excuses, les accents n'ont pas été gérés correctement par le fichier HTML):

Method Summary

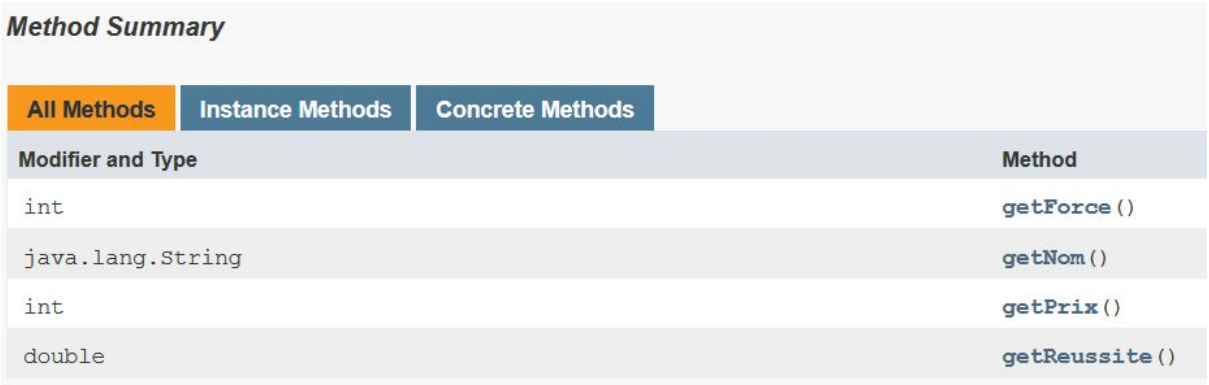
All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	addMana()	MÃ©thode permettant d'ajouter Ã la rÃ©serve de mana du joueur un nombre Ã©gal au numÃ©ro de son tour
void	attaque()	MÃ©thode qui gÃ¨re la phase d'attaque, avec le choix des crÃ©atures attaquantes (maximum 3) la rÃ©partition des blessures suivant une logique alÃ©atoire, mais Ã partir des caractÃ©risques de la crÃ©ature Gestions des exceptions possibles dÃ»es au scanner
void	entreeContinuer()	MÃ©thode permettant d'appuyer sur entrÃ©e pour passer au tour de l'adversaire Gain de visibilitÃ© dans l'affichage
void	genereMainDepart()	MÃ©thode ajoutant 7 crÃ©atures Ã la main (pioche) du joueur
Joueur	getAdversaire()	
int	getMana()	
java.lang.String	getNomJoueur()	
int	getNumTour()	
java.util.ArrayList<Creature>	getPioche()	
int	getPV()	MÃ©thodes get print et set permettant d'accÃ©der aux attributs privÃ©s de la classe
void	payerCreature()	MÃ©thode permettant de payer une crÃ©ature.
void	printChampBataille()	MÃ©thode permettant d'afficher chaque crÃ©ature du champ de bataille du joueur A savoir son numÃ©ro, son nom et sa force
void	printMana()	
void	printPioche()	MÃ©thode permettant d'afficher chaque crÃ©ature de la main (pioche) du joueur A savoir son nÂ°, son nom, son prix et sa force
void	printPV()	
void	setAdversaire (Joueur a)	
void	setMana(int a)	
void	setNumTour(int b)	

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
Joueur	getJoueur1()	MÃ©thodes get permettant d'avoir connaissance des attributs privÃ©s de la classe.
Joueur	getJoueur2()	
void	lancerPartie()	MÃ©thode qui fait tourner la partie tant qu'aucun joueur n'a perdu
void	partieEstTerminee(Joueur joueura, Joueur joueurb)	MÃ©thode permettant de vÃ©rifier si un joueur a perdu.
void	printNomJoueur(Joueur joueur)	MÃ©thode affichant dans le terminal le nom du joueur entrÃ© en paramÃ¨tre.
void	tour(Joueur joueur)	MÃ©thode qui gÃ¨re l'organisation gÃ©nÃ©rale d'un tour
void	tourPartie(double chance)	MÃ©thode dÃ©terminant Ã quel joueur c'est le tour de jouer



Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method
int	getForce()
java.lang.String	getNom()
int	getPrix()
double	getReussite()

Enfin, le fichier audio étant très lourd, on a trouvé une solution grâce à l'outil Git LFS (expliqué dans le readme)