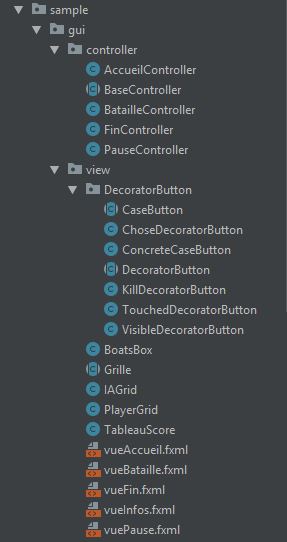
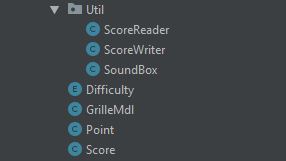
Clément TORTI  
Florent BECOUZE  
G7  
Année 2018/2019

**Conception et programmation Objets Avancées**Université Clermont-AuvergneIUT Info 2A

**Documentation Code :  
Bataille-navale**



(1. sample)  
 1.1. gui  
 1.1.1. controller  
 1.1.1.1. AccueilController  
 1.1.1.2. BaseController  
 1.1.1.3. BatailleController  
 1.1.1.4. FinController  
 1.1.1.5. PauseController  
 1.1.2. view  
 1.1.2.1. DecoratorButton  
 1.1.2.1.1. CaseButton  
 1.1.2.1.2. ChoseDecoratorButton  
 1.1.2.1.3. ConcreteCaseButton  
 1.1.2.1.4. DecoratorButton  
 1.1.2.1.5. KillDecoratorButton  
 1.1.2.1.6. TouchedDecoratorButton  
 1.1.2.1.7. VisibleDecoratorButton  
 1.1.2.2. BoatsBox  
 1.1.2.3. Grille  
 1.1.2.4. IAGrid  
 1.1.2.5. PlayerGrid  
 1.1.2.6. TableauScore  
 1.1.2.7. vueAccueil.fxml  
 1.1.2.8. vueBataille.fxml  
 1.1.2.9. vueFin.fxml  
 1.1.2.10. vueInfos.fxml  
 1.1.2.11. vuePause.fxml  
 1.2. launcher  
 1.2.1. Main  
 1.3. model  
 1.3.1. IAStrategie  
 1.3.1.1. EasyIA  
 1.3.1.2. HardIA  
 1.3.1.3. IA  
 1.3.1.4. IAFactory  
 1.3.1.5. MediumIA  
 1.3.2. Observer  
 1.3.2.1. IObserver  
 1.3.2.2. Partie  
 1.3.2.3. Subject

****

1.3.3. Util  
 1.3.3.1. ScoreReader  
 1.3.3.2. ScoreWriter  
 1.3.3.3. SoundBox  
 1.3.4. Difficulty  
 1.3.5. GrilleMdl  
 1.3.6. Point  
 1.3.7. Score

**(1. sample)**

C’est le dossier global du projet, dans lequel on a une structure MVC : dossier « gui » pour les packages « controller » et « view » et le package « model »

**1.1. gui (graphical user interface)**

C’est le dossier qui possède les packages « controller » et « view ». Dans le dossier, il y a tout ce qui concerne le côté graphique de l’application.

**1.1.1. controller**

C’est le package permettant de rassembler tous les contrôleurs des différentes vues. Chaque contrôleur hérite de la classe abstraite BaseController.

**1.1.1.1. AccueilController**

C’est le contrôleur d’accueil, c’est-à-dire, celui qui va être lancé en premier par la classe Main. A l’initialisation du contrôleur, on lance la musique d’accueil, on créé un ensemble de RadioButton, pour permettre le choix de la difficulté de l’IA et on récupère dans le fichier (représentant notre base de données), les différents scores et on les mets dans la ListView de la vue.

La méthode « start », représente l’action de cliquer sur le bouton « Start » de la vue « vueAccueil ». Elle créée, en fonction du RadioButton choisit, une partie avec la difficulté choisie. Elle « met de côté » la partie actuelle, puis change de « scene » (sans changer de fenêtre) en allant sur la vue de bataille.

La méthode « infos », représente l’action de cliquer sur le bouton « Infos » de la vue « vueAccueil ». Elle ouvre une nouvelle fenêtre (« stage « ) avec la vue d’informations.

**1.1.1.2. BaseController**

C’est la classe abstraite qui est la classe mère de toutes les classes de contrôleurs. Elle permet de « garder en mémoire » le « stage » dans lequel on se trouve, c’est-à-dire, la fenêtre actuelle dans laquelle on est, et de pouvoir changer la « scene » de cette fenêtre, c'est-à-dire la vue affichée. On peut aussi garder en mémoire la partie actuelle (duree, nbCoup, winner, les 2 grilles, …).

La méthode « changeScene », permet de changer de vue sans changer de fenêtre. On modifie donc pour cela la « scene » du « stage » (la vue de la fenêtre). On modifie aussi le contrôleur puisque c’est le contrôleur de la nouvelle vue qui doit prendre le relais.

La méthode « openStage », permet d’ouvrir la seule autre fenêtre de l’application, c'est-à-dire, la fenêtre des informations. Comme pour une fenêtre normale, on charge les éléments de la vue que l’on voudra afficher (FXMLLoader + Parent avec la méthode load() ), on lui ajoute un titre, une « scene » (la vue) et on l’affiche. Ici, nous n’avons pas besoin de changer de contrôleur puisqu’il n’y aura pas d’action à réaliser sur cette fenêtre (pas de bouton). Nous fermerons la fenêtre avec la croix, pour pouvoir revenir à la page d’accueil.

La méthode « getCurrGame » retourne la partie actuelle et « setCurrGame » met à jour la partie actuelle en la gardant en mémoire.

**1.1.1.3. BatailleController**

C’est la classe contrôleur de la vue « vueBataille ». A l’initialisation, on lance la musique du jeu, puis on configure les grids. Pour cela, on configure les boutons (les cases de chaque grille de jeu), en mettant les bonnes décorations. Ensuite, on attache les 2 grilles et le contrôleur (BatailleController) à la partie, c'est-à-dire qu’ils vont dans une liste (dans la classe abstraite « Subject »), qui garde tous les éléments qui devront être mis à jour après chaque action. On donne par la suite, la partie aux 2 grilles. Puis on récupère les messages dans les « ObservableList » (on bind les listes des messages de la partie avec la vue). On fini par initialiser le timer (le chrono du jeu). Pour se faire, on définit une tâche que le timer doit réaliser : on incrémente le compteur de « duree » d’une Partie et on lance une méthode permettant d’afficher le nouveau compteur (compteur modifié). Pour finir, on attache au timer, la tâche, qu’il devra réaliser en boucle.

La méthode « pause » est appelée via le bouton de la vue « vueBataille » et permet d’arrêter le timer et la musique de jeu. Et on change de « scene » (et de contrôleur) pour aller sur la vue de pause.

La méthode « finish » est appelée quand la partie est terminée (gagnée ou perdue). Elle permet d’arrêter le timer et de changer de « scene » (et de contrôleur) pour aller sur la vue de fin (avec le résultat de la partie).

La méthode « update », permet de « mettre à jour » la partie, en regardant si la partie est finie ou si un bateau est détruit. Si c’est le cas, alors on met à jour la bonne « boatsBox », en disant qu’un bateau est détruit (en changeant son image).

**1.1.1.4. FinController**

Cette classe représente le contrôleur de la vue « vueFin ». Lors de l’initialisation, on regarde si le joueur a gagné ou perdu et on affiche le message correspondant avec la durée de la partie et le nombre de coups du joueur. Petite particularité : si le joueur a perdu, on supprime le « TextField » où normalement on doit donner notre nom pour enregistrer notre performance.

La méthode « accueil » est activée lorsque l’on clique sur le bouton « Retour à l’accueil »  de la vue « vueFin ». On coupe le son du jeu et on remet le son de la page d’accueil. Si le joueur a gagné (et a rempli son nom), on l’enregistre dans le fichier texte qui nous sert de base de données. Pour finir, on change de « scene » (et de contrôleur) pour aller sur la vue d’accueil.

**1.1.1.5. PauseController**

C’est le contrôleur en charge de la vue « vuePause ». Pour initialiser, on change juste la musique du jeu pour la musique d’accueil.

La méthode « resume » est appelée quand on clique sur le bouton « Retour », ce qui nous permet de revenir au jeu, en arrêtant la musique d’accueil et en changeant la « scene » (et de contrôleur) pour aller sur la vue du jeu.

La méthode « quit » est lancée lors du clique sur le bouton « Quitter », ce qui nous permet de revenir à la page d’accueil et de mettre à zéro la partie. On arrête, ici aussi, la musique d’accueil et on change de « scene » (et de contrôleur) pour aller sur la vue d’accueil.

**1.1.2. view**

C’est le package permettant de rassembler tous ce qui est en rapport avec la vue (ce que voit l’utilisateur).

**1.1.2.1. DecoratorButton**

Ce dossier permet de gérer le design des boutons (ceux représentant les cases de la bataille) en fonction de leurs états, c’est-à-dire, s’ils représentent une case vide, une case avec un bateau dessous ou si le bateau est coulé.

**1.1.2.1.1. CaseButton**

ati

**1.1.2.1.2. ChoseDecoratorButton**

xcr

**1.1.2.1.3. ConcreteCaseButton**

bgfv

**1.1.2.1.4. DecoratorButton**

L ;,

**1.1.2.1.5. KillDecoratorButton**

N,

**1.1.2.1.6. TouchedDecoratorButton**

nb

**1.1.2.1.7. VisibleDecoratorButton**

nhtfgv

**1.1.2.2. BoatsBox**

njgv

**1.1.2.3. Grille**

Rg

**1.1.2.4. IAGrid**

N gh

**1.1.2.5. PlayerGrid**

Ngv

**1.1.2.6. TableauScore**

hng

**1.1.2.7. vueAccueil.fxml**

nh

**1.1.2.8. vueBataille.fxml**

,jhng

**1.1.2.9. vueFin.fxml**

gbf

**1.1.2.10. vueInfos.fxml**

,k

**1.1.2.11. vuePause.fxml**

jgnfb

**1.2. launcher**

C’est le package permettant de rassembler la classe « Main », celle qui va être lancée au tout début du programme.

**1.2.1. Main**

N gf

**1.3. model**

C’est le package permettant de rassembler tous les classes en rapport avec ????

**1.3.1. IAStrategie**

Ce dossier contient l’ensemble des classes, gérant la difficulté de jeu de l’IA.

**1.3.1.1. EasyIA**

vcn

**1.3.1.2. HardIA**

gtref

**1.3.1.3. IA**

gte

**1.3.1.4. IAFactory**

gtre

**1.3.1.5. MediumIA**

rtefc

**1.3.2. Observer**

Ce dossier ???? contient les classes permettant de gérer le pattern « Observateur » ????

**1.3.2.1. IObserver**

fdc

**1.3.2.2. Partie**

asz

**1.3.2.3. Subject**

U^po

**1.3.3. Util**

Fd

**1.3.3.1 ScoreReader**

Fg

**1.3.3.2. ScoreWriter**

Ryjk

**1.3.3.3. SoundBox**

kyu

**1.3.4. Difficulty**

Mo^po

**1.3.5. GrilleMdl**

bnc

**1.3.6. Point**

Rhgf

**1.3.7. Score**

,nhbg