NOEL Arnaud
SANZOVO Victor

Rapport A31/32 : Choix de conception

Première Partie :

L'architecture de logiciel que nous avons choisie correspond au MVC. Le logiciel est organisé en trois parties : le Modèle (model), contenant les données, la Vue (view) qui les affiche et le Contrôleur (controller) qui les manipule.

Nous avons mis en place dans notre Controller une classe Facade : elle sélectionne la vue en fonction de ses demandes et en manipulant les données du modèle. Facade permet d'encapsuler la complexité entre le modèle et la vue, n'en rajoutant pas inutilement dans ces deux parties.

Pour limiter la dépendance entre la classe Echec du modèle et la classe echecWindow de la vue, nous avons décidé d'utiliser le design pattern de l'Observer. Nous avons créé une interface echecObserver qui surveille (« observe ») la classe Echec et permet à echecWindow de réagir en fonction des changements survenus sur Echec. Les différentes modifications du controller demandant des changements sur la view sont le choix d'un mouvement par un joueur, une modification dans les mouvements qu'il peut faire ou la mise en échec/échec et mat d'un joueur. Chacune de ces situations correspond à une méthode dans echecObserver.

Les différentes pièces ont plusieurs fonctionnalités et propriétés communes : elles ont chacune la possibilité de capturer une autre pièce, une liste de déplacements possibles, une couleur… Il a été décidé de faire hériter chaque type de pièce d'une super classe « Piece » contenant toutes ces propriétés communes. Cela revient à suivre le principe DRY : si nous n'avons pas mutualisé le code, nous nous serons répété.

Les différents types de pièces sont ensuite créés par pieceFactory, faisant appel à la Factory Method : nous pouvons donc instancier chaque Pièce depuis la classe Echec, sans qu'Echec n'ait besoin de connaître leurs classes concrètes. Pour cela, la classe Echec utilise la méthode setPiece(), faisant appel à notre pieceFactory qui créera toutes les pièces de l'échiquier.

Nous avons décidé de signaler la couleur des Pièces avec un énumérateur Color, contenant deux possibilités : BLACK et WHITE. Cela nous permet d'éviter les confusions qu'aurait pu entraîner l'utilisation d'un booléen ou d'un entier pour identifier la couleur d'une pièce.

NOEL Arnaud
SANZOVO Victor

Deuxième Partie :

Pour le calcul des mouvements possibles, nous avons opté pour une autre solution que celle du premier rendu qui est que dans un premier temps la piece va calculer tous les mouvements qu'elle peut faire dans l'échequier sans se préoccuper si ce mouvement met en échec qui que ce soit et après en second temps c'est la classe Echec qui va récupérer les mouvements de la pièce calculés avant et qui va regarder si le mouvement met le roi de la couleur de la pièce en échec et si elle ne met pas en échec le mouvement est ajouté dans un tableau de boolean en 2d qui représente tous les mouvement possible par la pièce et est renvoyé .

Pour la vérification de l'échec on utilise la fonction verifEchec() on récupère le roi de l'équipe adverse puis on vérifie tous les mouvement possible des pièces de la couleur du joueur qui joue et on regarde si une des pièces à dans le tableau renvoyé la position du Roi en temps que mouvement possible, si oui la pièce est ajouté dans une liste de Piece nommé piecesAttaquant et cette liste est renvoyé à la fin de la fonction

Pour la vérification de l'échec et math on utilise la fonction verifEchecMath(), on récupère en premier lieu la liste des pièce attaquantes si il y a plus de 2 pièces qui attaquent on regarde si le roi peut bouger avec la fonction calculMouvementPossible() et si il ne peut pas alors le roi est en échec et math par contre si il n'y a que une pièce qui attaquent alors on regarde toutes les pièces de la couleurs que le roi attaqué et on regarde si une pièce peut bouger et vu que la fonction calculMouvementPossible() ne peut pas renvoyé de mouvement qui mettrait le roi en échec donc si aucun mouvement n'est renvoyé cela veut dire que le roi est en échec et math.

La classe Echec contient la méthode PromotionPion() assurant la promotion des pions,qui envoie un signal à la EchecWindow et attend que le joueur ait choisi quelle pièce il souhaite que son pion soit promu et ensuite change le pion par la pièce voulut. La fonction pionPromu() permet d'arrêter le wait() de PromotionPion() et donne le nom de la nouvelle pièce. Elle est appelée par la façade. La vue s'occupe d'afficher le menu de choix de la pièce et de changer le visuel de la pièce une fois ce choix effectué.

Chose que nous n'avons pas prévue dans notre premier plan, nous avons finalement créé une classe joueur. Il s'est avéré que c'était finalement une solution plus simple que de se contenter d'un string pour identifier les deux joueurs. On en profite pour identifier le roi correspondant au joueur, ce qui permet de savoir directement quel joueur a été mis en échec lorsqu'un roi est mis en échec. On identifie aussi la couleur du joueur.

Nous voulions enregistré le nom des pièces dans les boutons qui correspond au case de l'échiquier donc nous avons créé une classe Button_piece qui hérite de la classe JButton et ainsi nous avons mis dans la classe le nom et on as utilisé cette opportunité pour donner au bouton l'image correspondante avec la pièce.

L'évolution du score est récupéré quand un mouvement est fait, on regarde si la pièce étant présente dans l'échequier est une pièce si oui on lance la fonction

updateScore() en donnant le score attaché à la pièce (donné à sa création) qui ensuite ajouté au score du joueur qui as mangé la pièce et on change le score affiché sur l'écran.