NOEL Arnaud
SANZOVO Victor

Rapport A31/32 : Choix de conception

Première Partie :

L'architecture de logiciel que nous avons choisis correspond au MVC. Le logiciel est organisée 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 des changements survenus sur Echec. Les différentes modifications demandant 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 créées par pieceFactory, faisant appelle à la Factory Method : nous pouvons donc instancier chaque Pièce depuis la classe Echec, sans qu'Echec n'ait besoin de connaitre 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 entrainer l'utilisation d'un booléen ou d'un entier pour identifier la couleur d'une pièce.

Deuxième Partie :

Pour le calcul des mouvements possibles, plutôt que d'utiliser un tableau d'entier en 2 dimensions, ce qui montrait certains inconvénients, nous avons décidé de créer une map, ayant un entier comme identifiant et un tableau d'entier comme valeur. Chaque clé contient le déplacement réalisé en faisant l'un des mouvements possibles, et la map contient la liste des mouvements possibles de la pièce sélectionnée.

NOEL Arnaud
SANZOVO Victor

C’est la classe Echec qui va vérifier elle-même si il est possible de faire un rock durant le calcul de tous les mouvements possibles.

Une nouvelle méthode searchMouvementpossibleEchec() renvoie un tableau de booléen vérifiant quels pions il est encore possible de bouger en étant en situation d’échec, encapsulant la difficulté qui serait sinon dans la méthode vérifiant les échec et mat.

Echec contient aussi une méthode assurant la promotion des pion, permettant d’échanger le pion par une tour, un cavalier, une dame ou un fou au choix, avec la dame comme choix par défaut. La fonction pionPromu() agit après la promotion, et permet de changer le nom de la 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 prévus au départ, 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 mit en échec lorsqu’un roi est mit en échec. On identifie aussi la couleur du joueur.

Nous avons rencontré une certaine difficulté avec les pièces du jeu sur la vué, aussi nous avons encapsulé cette difficulté en leur créant une classe à part dans la vue. Elle hérite de J_Button et dispose de ses propres propriétés en plus : des coordonnés x et y et un nom.