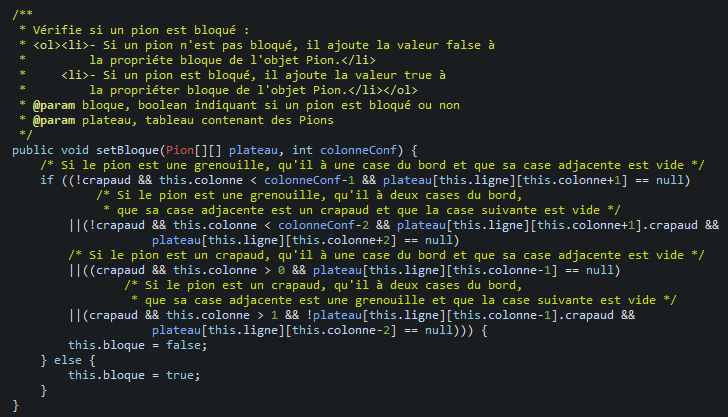


Cette fonction est appelée pour déplacer un pion théoriquement non bloqué. Cependant, cette fonction vérifie quand même ce boolean premièrement pour les tests unitaires et deuxièmement pour éviter de potentiels cas d’erreur et des vérifications inutiles. Cette fonction vérifie premièrement le type de pion ainsi que la colonne sur laquelle ce trouve le pion. Cela est nécessaire pour éviter les erreurs de déplacements si le pion se trouve à une extrémité du plateau. Ensuite, cette fonction ajoute ou enlève 1 ou 2 à la valeur de la colonne du pion en fonction de si sa case adjacente et libre ou non ou que sa case adjacente est du même type que celui-ci.

On remarque que les arguments sont tous deux issues d’une instance d’un objet plateau. Les vérifications sur l’initialisation de ces propriétés étant déjà faite préalablement, il n’y a aucun risque d’erreur de null pointer exception en appelant cette fonction.

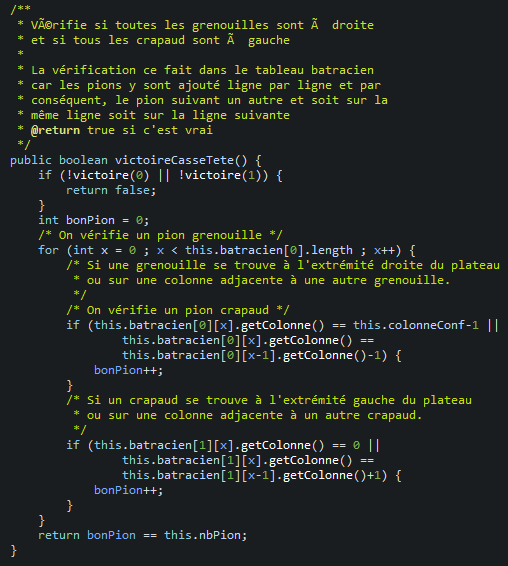
Nous avons choisi cette structure car elle permet d’éviter un maximum de cas d’erreur et qu’elle est facilement maintenable. Par exemple, nous avions au début codé cette fonction pour qu’un pion puisse passer par-dessus un pion de type différent. Notre professer référent nous ayant indiqué qu’un pion ne pouvez chevaucher qu’un pion du même type, il ne nous à fallut que quelques minutes pour corriger ce problème. Cette fonction est donc facilement maintenable.



La structure de ce code est très semblable à celle de la fonction précédente à la différence qu’il n’y à qu’une seule boucle if. Cette boucle vérifie tous les cas dans le quel un pion de n’importe quel type est bloqué ou non.

Comme pour la fonction précédente, les arguments de cette fonction sont supposés non null pour éviter les cas d’erreur.

Nous avons choisi cette structure pour les mêmes raisons que la fonction précédente.



 Cette fonction est appelée après chaque action d’un joueur en mode casse-tête. Elle vérifie le bon emplacement des pions sur le plateau en fonction des pions contenu dans le tableau batracien de l’instance du plateau en cour. Cela se justifie par le fais que lors de l’initialisation de ce tableau, celle-ci parcours le plateau ligne à ligne ce qui signifie qu’un pion de ce plateau est forcément sur la même ligne ou sur la suivante.

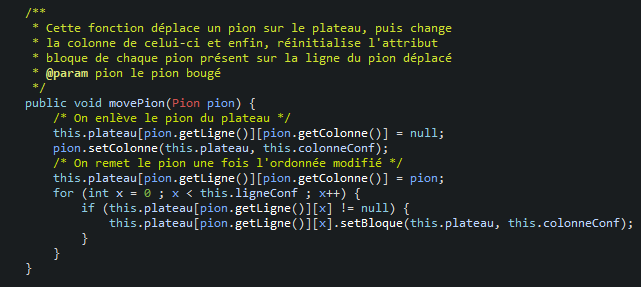
Cette fonction ne peut pas générer d’erreur puisque le tableau batracien est initialisé chaque fois qu’une instance de Plateau est créé.

Cette fonction montre l’intérêt d’avoir une copie des pions du plateau dans un tableau à part. Cette fonction aurait été beaucoup plus longue elle aurait fait les vérifications directement sur le plateau.

 Cette fonction créé un plateau de pion contenant toutes les instances de pions avec leur caractéristique. Ce tableau permet d’effectuer des opérations plus rapidement sur les pions du plateau.

Cette fonction ne peut pas créer d’erreur car nbPion est toujours pair et s’il est égal à 0, les boucle for ne se lancerons pas.

On a décidé de créer ce plateau justement pour simplifier les opérations sur les instances de la classe Plateau.

 Cette fonction est appelée avec comme argument un pion qui a été préalablement été vérifié comme non bloqué. Elle change la position d’un pion avec la fonction setColonne que l’on a vu précédemment et qui modifie ainsi le plateau ainsi que la propriété bloque de chaque pion présent sur la ligne du pion bougé.

