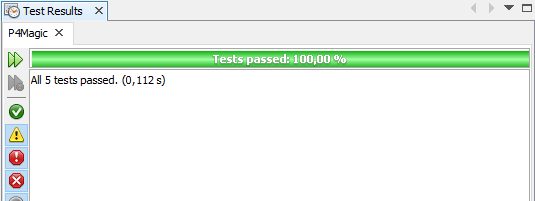
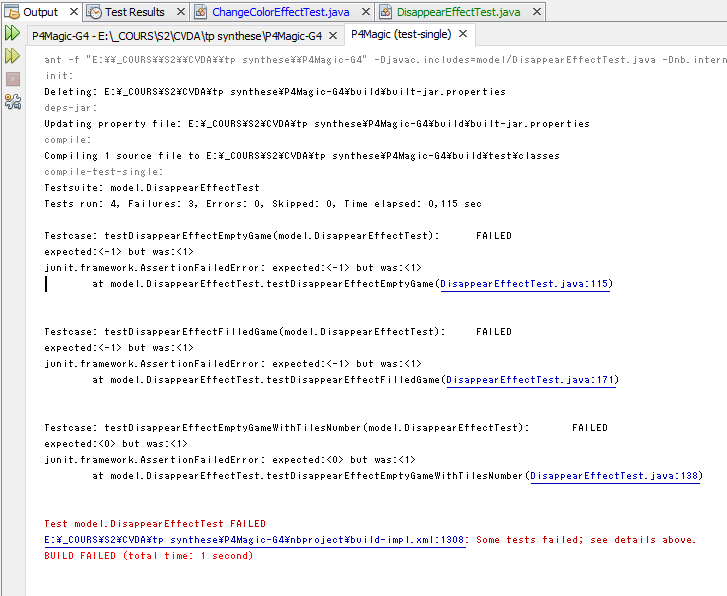
**Compte-Rendu P4-Magic**

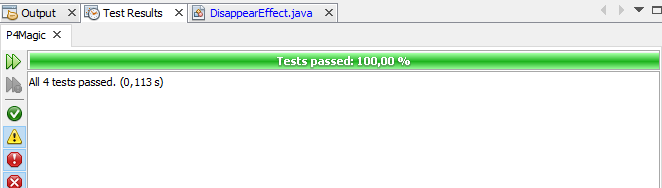
**Q1 –**



**Q2 –**



**Q3 –**



@Override

public void playEffect(int line, int column, Game game) {

game.getBoard().getTileIJ(line, column).setStatus(-1);

}

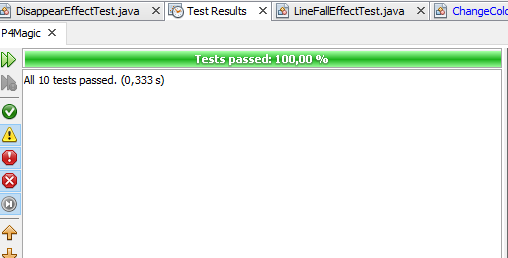
**Q4 –** Fait.

**Q5 –** Commit réalisé.

L’effet était difficile à implémenter, mais nous avons choisi d’exploiter la méthode « playMove » qui existait déjà. Nous avons supprimé la lecture d’effets et avons utilisé le même code pour faire une deuxième méthode : « playMoveNoEffect ». Nous avons ensuite, dans la classe propre à notre effet, choisi de jouer sur toutes les colonnes (sauf celles jouées à l’origine) un pion sans effet. Le code est défini de telle manière que le current player est redéfini avant chaque coup. Il est remis à la bonne valeur à la fin par la méthode playMove appellée de base.

Pour les tests, nous n’en avons fait qu’un seul par choix ; il contient 3 asserts, et est implémenté de telle manière qu’il peut tester les boards vides, pleines ou commencées.

**Q6 –**



**Q7 –**

QL1 :

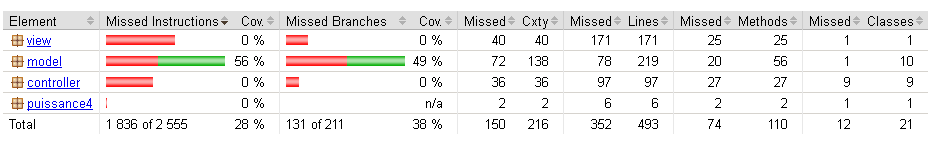
On a décidé d’ajouter un effet qui joue au hasard sur une colonne... En plus, le coup aléatoire peut déclencher des effets ce qui provoque des synergies éventuelles intéressantes.

QL2 :

Nous avons implémenté le code. Ce dernier vérifie, dans différentes configurations (pas de gagnant, combinaison en colonne, combinaison en ligne, combinaison en diagonale) si le gagnant désigné par la méthode « win » est le bon.

QL3 :

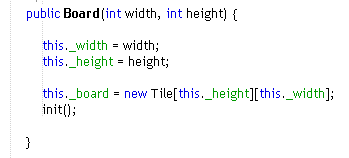
Pour répondre à cette question,nous avons pensé qu’utiliser le plugin JACoCoverage était une bonne idée. En effet ce dernier permet de déterminer quelles parties du code ne sont pas ou sont mal testées.

  
On remarque effectivement qu’il y a énormément d’’instructions non testées (il est inutile d’en tester certaines).  
Tester le package view ne me parait pas indispensable dans un premier temps, car le package le plus crucial reste « model ».   
Cependant, Dans model, les methodes non testées sont surtout des getteurs, des setteurs et des toString, donc rien de vraiment très urgent (peu probable de faire des erreurs là-dessus).  
De même pour le main dans le package « puissance4 »; impossible de faire une faute sur 4 lignes.  
Si nous devions établir un ordre de test, ce serait un modèle en V. Nous testerions ainsi les packages les plus annexes avant d 'élargir au fur et à mesure notre zone de test vers les packages importants. Nous commencerions donc avec les tests du package « model », puis ceux de « controller » et enfin « view » (en prenant bien soin de, au sein de chaque package, tester les méthodes et classes importantes en premier lieu).

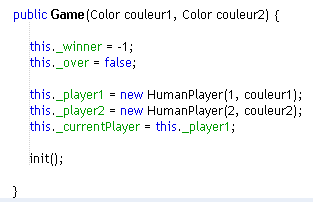
QL4 :

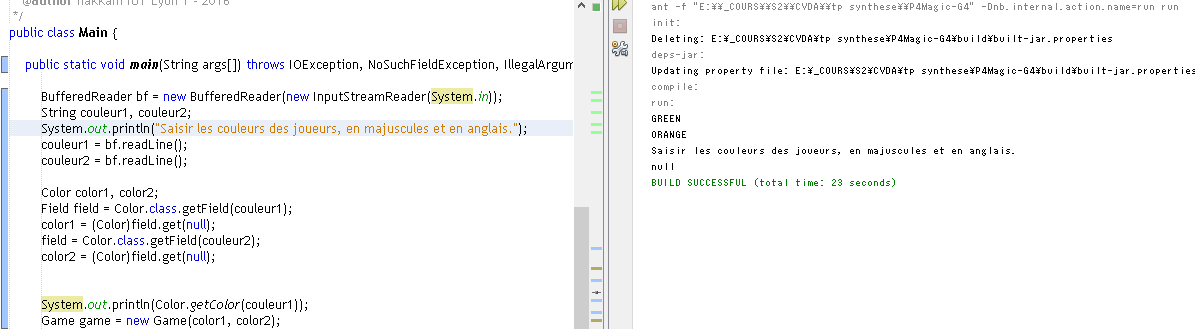
Une mécanique intéressante pour faire cela serait d’ajouter un système de tours. Dans la méthode « playmove », on pourrait ajouter un système d’incrémentation et avec des modulos on pourrait faire un compteur de tours ! cela deviendrait une statistique intéressante au cours du jeu.  
Pour pouvoir jouer à plusieurs joueurs, il faudrait remplacer une très grande partie du code.   
Dans les tests unitaires, on ne pourrait plus simplement vérifier que le jouer est différent ; il faudrait aussi s’assurer qu’il s’agit de la bonne personne.  
Des effets comme le changement de couleur devraient prendre en compte cet ajout de joueur, et répartir (ou pas) les chances d’être la cible de l’effet.

QL5 – QL6 :

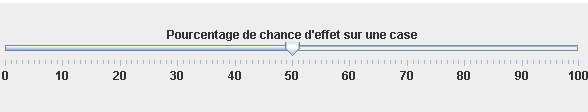
Les paramètres du constructeur de la planche permettent de changer la largeur et la hauteur de la grille.  


Nous avons ajouté un constructeur de partie, pour pouvoir ajouter la couleur des joueurs avant de lancer la partie. Nous avons aussi modifié le main (la sélection de couleurs n’est pas encore intégrée à l’IHM)





Chances d’effet : cette fonctionnalité existe déjà.

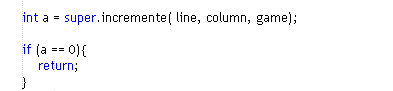


QL7 :

Pour cette question, nous avons ajouté deux variables de classe dans « Effect ». Chaque instance d’effet garde désormais le compte du nombre de fois où elle a été appelée. Pour les effets interminables, il suffit de ne pas changer la valeur de « iterationsMax » ou de la laisser à 0.



Il suffit donc ensuite d’ajouter ce code au début de chaque effet.



NB : on peut aussi ne pas ajouter ce code si on ne veut pas de limite d’itération.

QL8 :

Dans Board on pourrait imaginer un nouvelle méthote getTilesCount( int numJoueur) qui compterait les tuiles du Joueur passé en paramètre plutôt que d’avoir une méthode par joueur (sinon rajouter une nouvelle méthode par joueur en plus).

Dans Game ajouter un getJoueurX() ;

QL9 :

Comment une case peut-elle être morte ?  
Sinon, on peut imaginer un système de chute en cascade ; prenons l’exemple d’une case qui disparait sous d’autres ; on peut faire chuter celles du dessus ! on peut même imaginer que ces jetons puissent faire gagner pendant leur chute (en complétant une ligne).