Examen de Test et Vérification
2020–2021

— Durée : 2h
— Type : ExamManager
— Rendu : une archive contenant les sources Java à déposer dans le dossier `rendu-test-verif`, et, pour les réponses aux autres questions, un fichier PDF à déposer dans le même dossier
— Tous documents autorisés.
— Toutes vos affaires (sacs, vestes, *etc.*) doivent être placées à l'avant de la salle.
— Aucun téléphone ne doit se trouver sur vous ou à proximité, même éteint.
— Les déplacements et les échanges ne sont pas autorisés.
— Aucune question ne peut être posée aux enseignants, posez des hypothèses en cas de doute.

Test (10 points)

Tests en boîte noire

Lights Out est un jeu de réflexion qui se joue sur une grille de cases lumineuses. Au début du jeu, certaines cases sont allumées, et d'autres éteintes. En appuyant sur l'une des cases, elle bascule d'état (*i.e.*, elle passe d'allumée à éteinte, ou d'éteinte à allumée), ainsi que les 4 cases adjacentes (*i.e.*, celles au-dessus, au-dessous, à gauche et à droite), comme illustré à la figure 1.

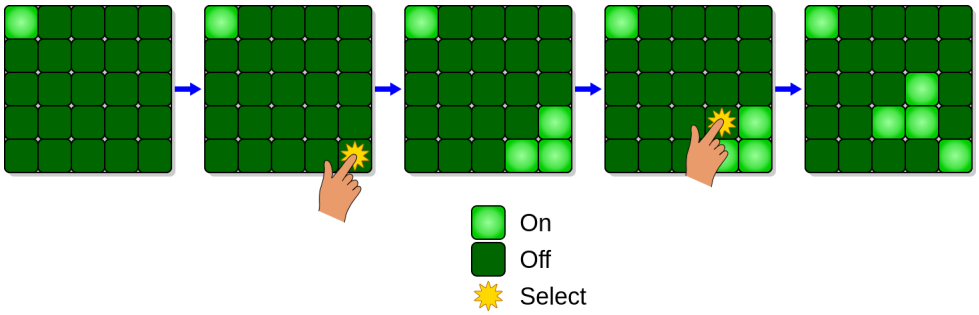

FIGURE 1 – Jeu *Lights Out* (source Wikipédia)

Soit une méthode `void basculement(boolean[][] grille, int x, int y)` permettant de basculer l'état d'une case lumineuse, ainsi que celui de ses 4 cases adjacentes. Le paramètre `grille` représente la grille de cases lumineuses (`true` pour une case allumée, `false` pour éteinte), et les paramètres `x` et `y` correspondent aux coordonnées de la case lumineuse appuyée par le joueur. Si les coordonnées sont hors des dimensions de la grille, une exception de type `ArrayIndexOutOfBoundsException` est levée.

1. Donnez un ensemble complet de cas de test pour tester la méthode. (3pts)
2. Écrivez, dans une classe de test JUnit, les méthodes de test correspondant à vos cas de test (pour compiler ces tests, une version minimale de la méthode testée est requise). (2pts)

Tests en boîte blanche

Soit la méthode `doubleChar` suivante qui revoie `true` si la chaîne de caractères passée en paramètre contient deux caractères consécutifs identiques, `false` sinon. Par exemple, `doubleChar("cool")` retourne `true`, alors que `doubleChar("test")` retourne `false`.

```
1  boolean doubleChar(String s) {
2    if (s.equals(""))
3      return false;
4    int i = 0;
5    while (i < s.length() - 1 && s.charAt(i) != s.charAt(i + 1))
6      i++;
7    if (i == s.length())
8      return false;
9    else
10     return true;
11 }
```

1. Construisez le graphe de flot de contrôle de la méthode, en indiquant à quoi correspondent chaque nœud et chaque arête dans le code. (2pts)
2. Donnez un ensemble minimal de cas de test qui couvre tous les chemins élémentaires du graphe. (2pts)
3. La méthode est erronée. À l'aide de vos cas de test, identifiez et corrigez la faute. (1pt)

Vérification (10 points)

Soit la méthode suivante qui renvoie un nouveau tableau dont le contenu est un décalage circulaire du tableau passé en paramètre selon un pas donné. Le pas correspond au nombre de cases vers la droite dont chaque élément est décalé, en considérant que les deux extrémités du tableau communiquent (tout élément « débordant » la fin du tableau est réintroduit par le début du tableau). Par exemple, `decalage([1,7,0,5,14], 3)` renvoie le tableau `[0,5,14,1,7]`. Nous supposerons que le pas est strictement inférieur à la taille du tableau.

```
1  int[] decalage(int[] t, int pas) {
2    int[] d = new int[t.length];
3    int i = 0;
4    while (i < pas) {
5      d[i] = t[i + t.length - pas];
6      i = i + 1;
7    }
8    while (i < t.length - pas) {
9      d[i + pas] = t[i];
10     i = i + 1;
11   }
12   return d;
13 }
```

Une faute a été introduite dans cette méthode. Utilisez l'outil de vérification Why3 [1] pour prouver la correction totale de la méthode, une fois la faute identifiée et corrigée.

1. Écrivez la pré-condition de la méthode. (1pt)
2. Écrivez la post-condition de la méthode. (2.5pts)
3. Écrivez l'invariant de la première boucle. (2pts)

1. Avant de lancer l'outil de vérification Why3, commencez par exécuter la commande `why3 config --detect` afin qu'il détecte les prouveurs installés (*e.g.*, Alt-Ergo).

4. Écrivez la fonction variant de la première boucle. (0.5pt)
5. Écrivez l'invariant de la seconde boucle. (2.5pts)
6. Écrivez la fonction variant de la seconde boucle. (0.5pt)
7. Corrigez la faute. (1pt)