Exercice 1

Q1

Un défaut est une partie du code source qui ne respecte pas la spécification.

Une défaillance est un comportement du système testé qui ne respecte pas la spécification.

Un défaut est susceptible de déclencher une défaillance, et une défaillance est toujours causée par un défaut.

Q2

Le test unitaire consiste à tester séparément chaque "unité" du logiciel, une unité étant typiquement une fonction ou une méthode.

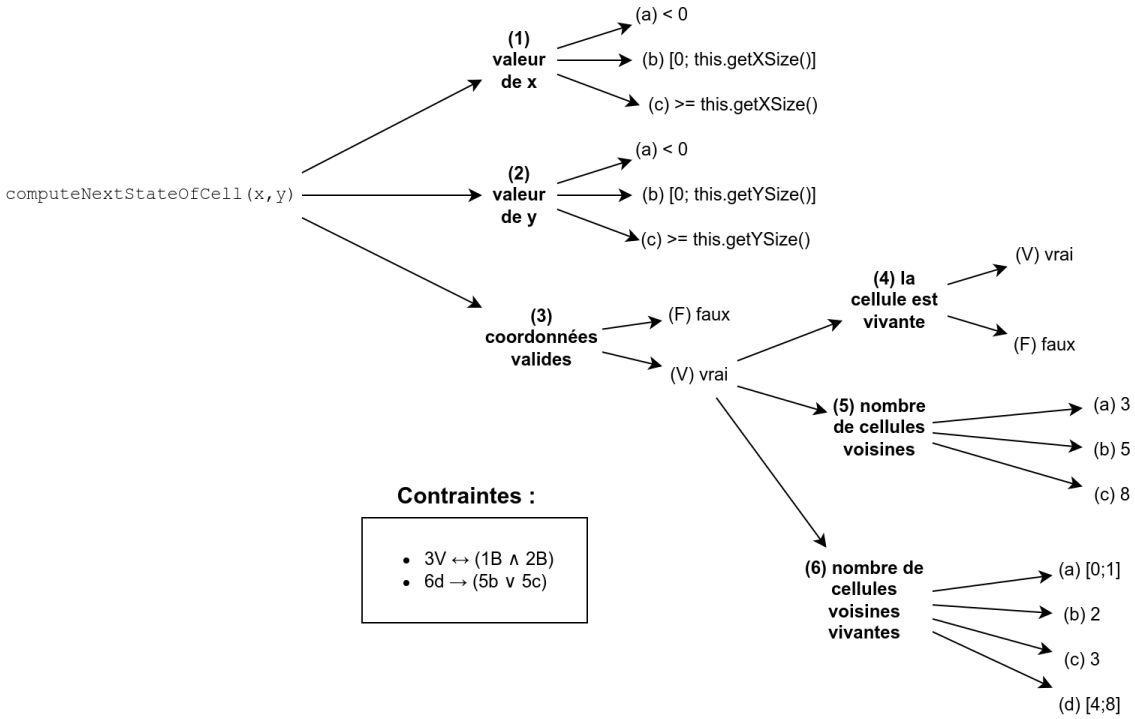
L'intérêt principal de cette pratique est de faciliter la localisation des défauts dans le code source, car si le test d'une unité échoue, il est plausible que le défaut soit localisé dans cette unité.

Exercice 2

Q3

- Entrées :
  - x : int
  - y : int
  - this : GameOfLife
    - getSizeX()
    - getSizeY()
    - getCellState(x,y), avec x dans [0 ; getSizeX()-1] et y dans [0 ; getSizeY()-1]
- Sorties :
  - result : boolean

Q4



Q5

	(1)	(2)	(3)	(4)	(5)	(6)
$C1$	B	B	V	V	C	B
$C2$	**A**	B	F	–	–	–
$C3$	**C**	B	F	–	–	–
$C4$	B	**A**	F	–	–	–
$C5$	B	**C**	F	–	–	–
$C6$	B	B	V	**F**	C	B
$C7$	B	B	V	V	**A**	B
$C8$	B	B	V	V	**B**	B
$C9$	B	B	V	V	C	**A**
$C10$	B	B	V	V	C	**C**
$C11$	B	B	V	V	C	**D**

Q6

Pour `getCellState()`, on notera plus simplement la liste des les coordonnées des cellules vivantes de la grille.

	getSizeX()	**getSizeY()**	**getCellState()**	**x**	**y**	**result**	**IllegalArgExc**
$C1$	5	5	**(3,3)**,(3,2),(2,3)	3	3	true	
$C2$	5	5		-1	3		×
$C3$	5	5		10	3		×
$C4$	5	5		3	-1		×
$C5$	5	5		3	10		×
$C6$	5	5	(3,2),(2,3)	3	3	false	
$C7$	5	5	**(0,0)**,(0,1),(1,0)	0	0	true	
$C8$	5	5	**(0,3)**,(0,4),(1,3)	0	3	true	
$C9$	5	5	(3,3)	3	3	false	
$C10$	5	5	**(3,3)**,(3,2),(2,3),(4,3)	3	3	true	
$C11$	5	5	**(3,3)**(3,2),(2,3),(4,3),(4,4)	3	3	false	

Exercice 3

Q7

Les *capteurs* posent un problème de contrôlabilité, car les valeurs qu'ils produisent peuvent varier d'un test à l'autre, et on peut difficilement forcer un capteur à rendre une valeur particulière.

Ces problèmes peuvent être résolu par la création de doublures pour les capteurs.

Q8

Oui, il faut une doublure de la classe *ReaSensors*, en créant une doublure de type *Sensors*.

Cette doublure pourra être configurée pour rendre des valeurs constantes en cas d'appel aux opérations *readWindDirection*, *readWindSpeed* et *readLuminosity*.

Q9

```java
@Test
public void testDecide() {
    // Given
    Sensors fakeSensors = mock(Sensors.class);
    when(fakeSensors.readWindSpeed()).thenReturns(12);
    when(fakeSensors.readWindLuminosity()).thenReturns(200);
    when(fakeSensors.readWindDirection()).thenReturns(Direciton.NORTH);
    HomeAutomation homeAutomation = new HomeAutomationImpl(fakeSensors);

    // When
    PlannedShuttersAction result = homeAutomation.decide();

    // Then
    assertEquals(PlannedShuttersAction.CLOSE, result);
}
```