# **Démineur**

**Fait travailler** : *(ex :boucles, conditions, POO...)*

* conditions
* doubles boucles
* POO (getter, setter, construct)

**Enoncé du Dojo :**

Une grille de démineur de taille variable (sous forme de tableaux) est envoyée en paramètre à une classe Demineur. Sur cette grille, les 0 sont des cases vides, les 1 correspondent à des bombes.

Une méthode tryCase($x,$y) permet de tester une case de la grille d’après ses coordonnées. Le retour de la méthode doit être “boom” si ça tombe sur une bombe, le nombre de bombes adjacente sinon. Une erreur doit être retournée si les coordonnées sont en dehors de la taille de la grille)

**Exemples** :

Exemple d’une grille de 4x3

[

[0,0,0,1],

[0,1,**0**,1],

[**1**,0,0,0],

]

si l’on souhaite tester la case 3,2 (en vert), le retour doit être 3 (il y a en effet 3 bombes autour, diagonales comprises)

si l’on souhaite tester la case 1,3 (en rouge), le retour doit être “boom” car nous sommes sur une bombe

*Remarque : au choix on peut commencer à compter les coordonnées à partir de 1 ou de 0 (ce qui peut se révéler plus simple pour le parcours des tableaux)*

**Correction :**

**class** Demineur

{

**private $field**;

**private $size**;

*/\*\**

*\* Demineur constructor.*

*\** ***@param*** *$field*

*\** ***@param*** *$size*

*\*/*

**public function** \_\_construct($field)

{

$this->setField($field);

}

**public function** tryCase($x, $y)

{

$x--;

$y--;

$nbMines = 0;

**if** (**isset**($this->getField()[$y][$x])) {

**if** ($this->getField()[$y][$x] == 1) {

**return 'boom'**;

}

} **else** {

**throw new** \Exception(**'ça sort'**);

}

**for** ($i = ($y - 1); $i <= ($y + 1); $i++) {

**for** ($j = ($x - 1); $j <= ($x + 1); $j++) {

**if** (**isset**($this->getField()[$i][$j])) {

$nbMines += $this->getField()[$i][$j];

}

}

}

**return** $nbMines;

}

*/\*\**

*\** ***@return*** *mixed*

*\*/*

**public function** getField()

{

**return** $this->**field**;

}

*/\*\**

*\** ***@param*** *mixed $field*

*\*/*

**public function** setField($field)

{

$this->**field** = $field;

}

}

**Tests :**

**class** DemineurTest **extends** TestCase

{

**public $demineur**;

**public function** \_\_construct () {

**parent**::*\_\_construct*();

$field = [

[0,1,1,0,0],

[1,1,0,0,0],

[0,1,0,1,0],

[0,1,1,0,1],

[0,1,0,0,0],

];

$this->**demineur** = **new** Demineur($field);

}

**public function** testMine()

{

$this->assertEquals(0, $this->**demineur**->tryCase(5,1));

$this->assertEquals(2, $this->**demineur**->tryCase(4,2));

$this->assertEquals(5, $this->**demineur**->tryCase(3,3));

}

**public function** textBoom() {

$this->assertEquals(**'boom'**, $this->**demineur**->tryCase(2,2));

}

**public function** testKO() {

$this->expectException(Exception::***class***);

$this->**demineur**->tryCase(20,2);

}

**public function** testKO2() {

$this->expectException(Exception::***class***);

$this->**demineur**->tryCase(-1,2);

}

}