# **Sudoku / Morpion**

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

* compétence travaillée
* compétence travaillée

**Remarque**: ce dojo a été présenté en 2 niveaux (morpion: facile / sudoku : difficile)

Le principe de résolution du dojo est très similaire, le sudoku demande juste un peu plus de travail.

Ce dojo est de difficulté assez élevée, à donner en fin de session.

## Sudoku

**Enoncé du Dojo :**

Le but est de vérifier si une grille de sudoku, totalement remplie et donnée en entrée est correcte ou non.

Règles du sudoku : https://fr.wikipedia.org/wiki/Sudoku#R.C3.A8gles\_de\_base

En d’autres termes, il faut vérifier si chaque ligne et chaque colonnes et chaque bloc de 9 cases (3\*3) ont toutes une seule fois les chiffres de 1 à 9. Si c’est le cas, on renvoit true, sinon false.

Remarque: de base, le jeu est un carré de 9\*9 mais il peut exister des variantes avec des grilles de toutes taille tant que cette taille correspond à n²\*n² (par ex: 4\*4, 16\*16, 25\*25, etc), il peut donc être intéressant de gérer une taille de grille variable.

**Correction :**

**<?php**

*/\*\**

*\* Created by PhpStorm.*

*\* User: sylvain*

*\* Date: 28/03/17*

*\* Time: 14:02*

*\*/*

*// 1h10*

**namespace** wcs;

**class** Sudoku

{

**protected $grid**;

**private $size**;

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

{

$this->**grid** = $grid;

$this->**size** = *count*($this->getGrid());

}

**public function** getGrid() :**array**

{

**return** $this->**grid**;

}

**public function** setGrid($grid)

{

$this->**grid** = $grid;

**return** $this;

}

*/\*\**

*\* verification des lignes, colonnes et blocs un a un.*

*\* Si checkPart renvoie false à un moment, on arrête et on renvoie false*

*\*/*

**public function** check() :bool

{

**for** ($i = 0; $i < $this->**size**; $i++) {

**if** (!$this->checkPart($this->getLine($i))) {

**return false**;

}

**if** (!$this->checkPart($this->getCol($i))) {

**return false**;

}

**if** (!$this->checkPart($this->getBlock($i))) {

**return false**;

}

}

**return true**;

}

*/\*\**

*\* on vérifie que la partie à vérifier (ligne col ou block) ne contient bien que des chiffres de 1 à 9*

*\*/*

**private function** checkPart($part)

{

$range = *range*(1, $this->**size**);

*sort*($part);

**return** $part === $range;

}

*/\*\**

*\* récupération de la ligne $num*

*\*/*

**private function** getLine($num)

{

**return** $this->getGrid()[$num];

}

*/\*\**

*\* récupération de la colonne $num*

*\*/*

**private function** getCol($num)

{

**for** ($i = 0; $i < $this->**size**; $i++) {

$col[] = $this->getGrid()[$i][$num];

}

**return** $col;

}

*/\*\**

*\* récupération du block de $this->size^2 numéro $num*

*\*/*

**private function** getBlock($num)

{

$blockSize = *sqrt*($this->**size**);

$x = ($num) % $blockSize;

$y = *floor*($num / ($blockSize));

**for** ($i = $blockSize \* $x; $i < $blockSize \* ($x + 1); $i++) {

**for** ($j = $blockSize \* $y; $j < $blockSize \* ($y + 1); $j++) {

$block[] = $this->getGrid()[$i][$j];

}

}

**return** $block;

}

}

**Tests :**

**class** SudokuTest **extends** TestCase

{

**public function** testLine()

{

$sudokuChecker = **new** \wcs\Sudoku(

[

[3,2,1,5,8,6,7,9,4],

[9,6,7,3,4,1,5,8,2],

[4,8,5,9,7,2,1,6,3],

[1,3,8,2,9,7,4,5,6],

[2,9,6,4,3,5,8,1,7],

[5,7,4,6,1,8,3,2,9],

[7,5,3,8,2,9,6,4,1],

[6,4,9,1,5,3,2,7,8],

[8,1,2,7,6,4,9,3,5],

]

);

$this->assertTrue($sudokuChecker->check());

}

**public function** test44True()

{

$sudokuChecker = **new** \wcs\Sudoku(

[

[1,2,3,4],

[3,4,1,2],

[2,1,4,3],

[4,3,2,1],

]

);

$this->assertTrue($sudokuChecker->check());

}

**public function** test44False()

{

$sudokuChecker = **new** \wcs\Sudoku(

[

[1,2,3,4],

[2,3,4,1],

[3,4,1,2],

[4,1,2,3],

]

);

$this->assertFalse($sudokuChecker->check());

}

## ------------------------------------------------------------------------

## Morpion

**Enoncé du Dojo**

Pour ce second dojo, c’est une grille de morpion complète qui vient en entrée (3\*3 classique mais possibilité de gérer des tailles variable n\*n si on souhaite davantage challenger les élèves)

Le but est d’analyser la grille et de dire qui est le gagnant (celui qui a les crois, les cercles ou s’il y a égalité)

**Correction**

**class** Morpion

{

**private $grid**;

*/\*\**

*\* Sudoku constructor.*

*\** ***@param*** *$grid*

*\*/*

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

{

$this->**grid** = $grid;

$this->**size** = *count*($this->getGrid());

}

*/\*\**

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

*\*/*

**public function** getGrid()

{

**return** $this->**grid**;

}

*/\*\**

*\** ***@param*** *mixed $grid*

*\*/*

**public function** setGrid($grid)

{

$this->**grid** = $grid;

**return** $this;

}

**public function** check()

{

**for** ($i = 0; $i < $this->**size**; $i++) {

$checks[] = $this->checkPart($this->getLine($i));

$checks[] = $this->checkPart($this->getCol($i));

}

$checks[] = $this->checkPart($this->getDiags()[1]);

$checks[] = $this->checkPart($this->getDiags()[2]);

**foreach** ($checks **as** $check) {

**if** ($check) {

**return** $check;

}

}

**return 'No winner'**;

}

**public function** checkPart($part)

{

**for** ($i = 0; $i < $this->**size**; $i++) {

$cross[] = **'x'**;

$circle[] = **'o'**;

}

**if** ($part === $cross){

**return 'cross wins'**;

} **elseif** ($part === $circle) {

**return 'circle wins'**;

}

**return false**;

}

**public function** getLine($num)

{

**return** $this->getGrid()[$num];

}

**public function** getCol($num)

{

**for** ($i = 0; $i < $this->**size**; $i++) {

$col[] = $this->getGrid()[$i][$num];

}

**return** $col;

}

**public function** getDiags()

{

**for** ($i = 0; $i < $this->**size**; $i++) {

$diags[1][] = $this->getGrid()[$i][$i];

$diags[2][] = $this->getGrid()[$this->**size** - $i-1][$i];

}

**return** $diags;

}

}

Tests :

**class** MorpionTest **extends** TestCase

{

public function testMorpionEq()

{

$morpion = new Morpion(

[

['o','x','o'],

['x','x','o'],

['o','o','x'],

]

);

$this->assertEquals('No winner', $morpion->check());

}

public function testMorpionOWin()

{

$morpion = new Morpion(

[

['o','o','o'],

['x','x','o'],

['x','o','x'],

]

);

$this->assertEquals('circle wins', $morpion->check());

}

}