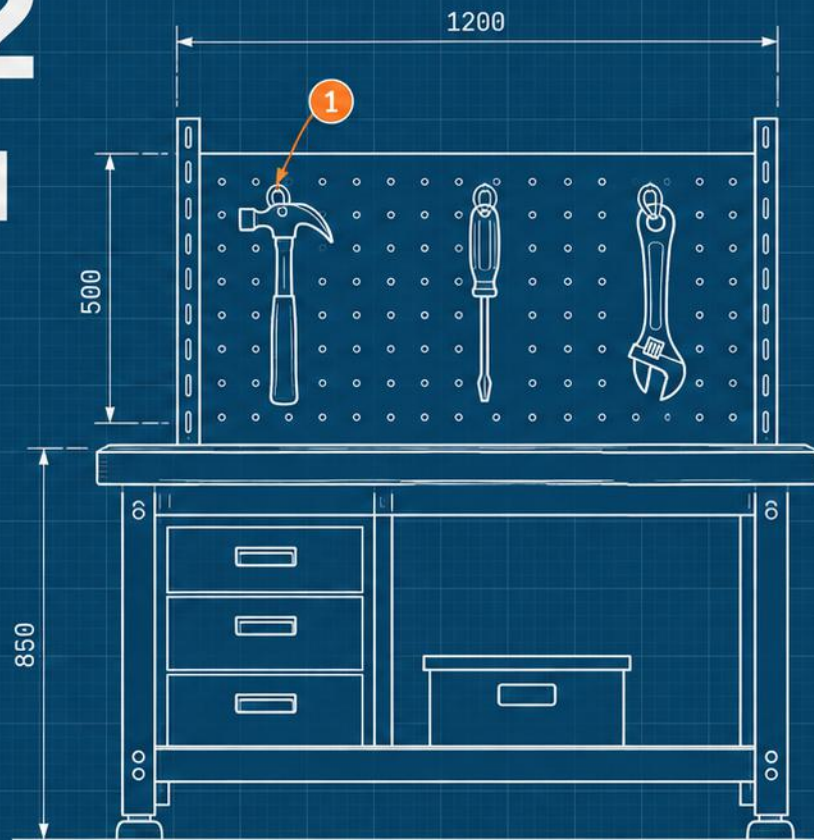


Bloc P00 · chapitre 2

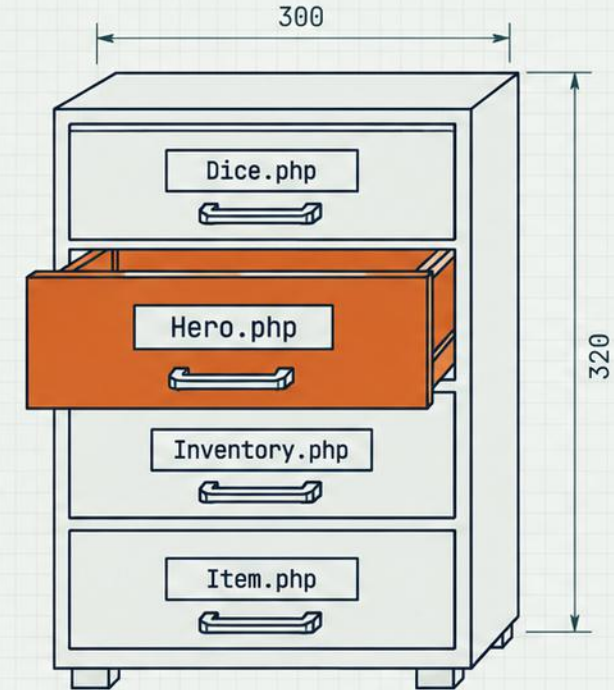
L'atelier : Composer, Pest et la CI

Un dépôt, des tests déjà écrits, une coche verte sur GitHub.



Le fichier porte le nom de la classe

```
1  poo-katas-26/  
2  |— composer.json  
3  |— phpunit.xml  
4  |— src/  
5  |   |— Dice.php           → class Dice  
6  |   |— Hero.php          → class Hero  
7  |   |— Inventory.php     → class Inventory  
8  |   |— Item.php          → class Item  
9  |— tests/  
10 |   |— Pest.php  
11 |   |— Niveau1Test.php  
12 |   |— Niveau2Test.php  
13
```



- Au chapitre 1, Hero et Dice tenaient dans un seul fichier. Avec quinze classes, deux personnes ne peuvent plus y travailler sans conflit.
- Les tests ne suivent pas les classes mais les niveaux : un fichier NiveauNTest.php par kata.

Le nom de famille des classes

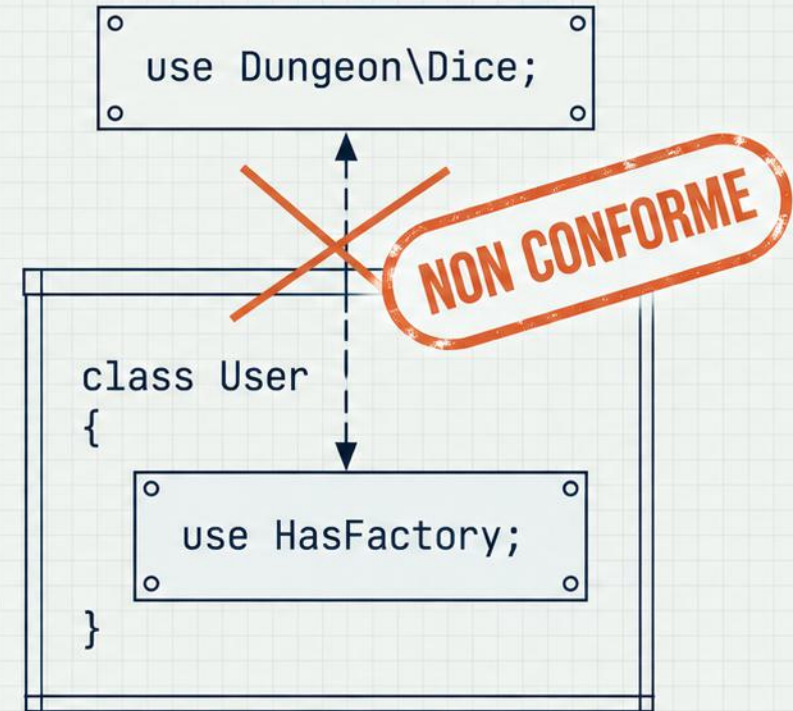
```
1 <?php
2
3 declare(strict_types=1);
4
5 namespace Dungeon;
6
7 class Dice
8 {
9     // ...
10 }
```

```
1 require __DIR__ . '/vendor/autoload.php';
2
3 use Dungeon\Dice;
4 use Dungeon\Hero;
5
6 $hero = new Hero('Arthur');
7 echo 'Le dé du donjon a ', Dice::d6()->sides, ' faces.', PHP_EOL;
```

- Votre projet et Laravel peuvent chacun contenir une classe Collection. Le namespace est un préfixe qui les distingue.
- Le nom complet est `Dungeon\Dice`. Ailleurs, `use` en haut du fichier importe ce nom, et on utilise ensuite le nom court.
- Cet import ne charge aucun fichier. Le chargement est le travail de l'autoload.

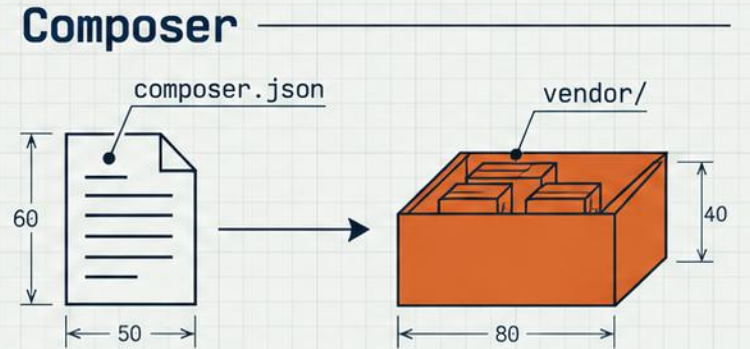
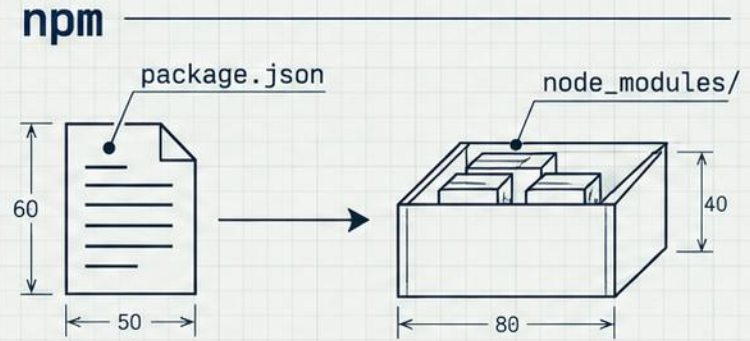
use avant la classe, use dans la classe

- `use Dungeon\Dice;` en haut du fichier, avant la classe : un **import de namespace**.
- `use HasFactory;` à l'intérieur d'une classe : **un trait**, du code copié dans la classe, vu au chapitre 7.
- **Pour les distinguer**, regardez si le mot se trouve avant la classe ou à l'intérieur. C'est la confusion la plus fréquente à l'arrivée sur Laravel.



Le gestionnaire de dépendances : npm, pour PHP

- Composer télécharge les bibliothèques dont votre projet a besoin, note leurs versions exactes, et charge vos classes automatiquement.
- Là où npm lit `package.json` et remplit `node_modules/`, Composer lit `composer.json` et remplit `vendor/`.
- Les bibliothèques viennent de Packagist, l'annuaire public des paquets PHP. Chaque paquet porte un nom éditeur/paquet : `pestphp/pest`, `laravel/framework`.
- Une **dépendance** : une bibliothèque écrite par quelqu'un d'autre et utilisée par votre projet.



Tout tient dans un fichier

composer.json
src/
tests/
pest.php

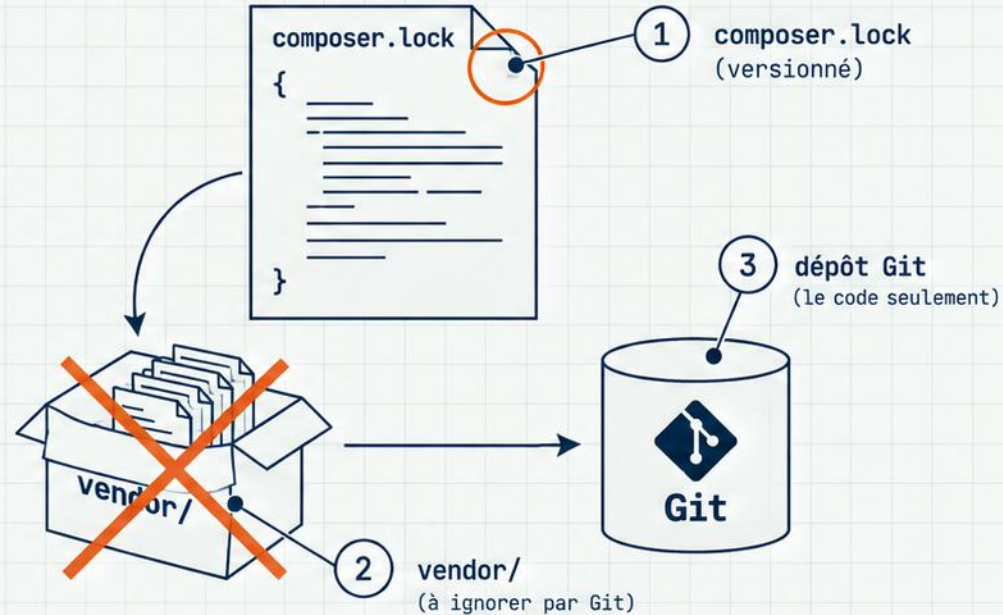
```
1 {
2     "name": "opmvpc/poo-katas-26",
3     "require": {
4         "php": ">=8.4"
5     },
6     "require-dev": {
7         "pestphp/pest": "^3"
8     },
9     "autoload": {
10         "psr-4": {
11             "Dungeon\\": "src/"
12         }
13     },
14     "scripts": {
15         "test": "pest"
16     }
17 }
```

- **require** : ce dont le projet a besoin pour tourner. PHP 8.4 minimum, pour private(set) et les hooks.
- **require-dev** : ce dont vous avez besoin pour développer, ici les tests.
- **^3** : la version 3 et ses mises à jour compatibles. 3.4 et 3.9 acceptées, 4.0 non.
- **scripts** : **composer test** lancera **pest**.

composer install suit composer.lock

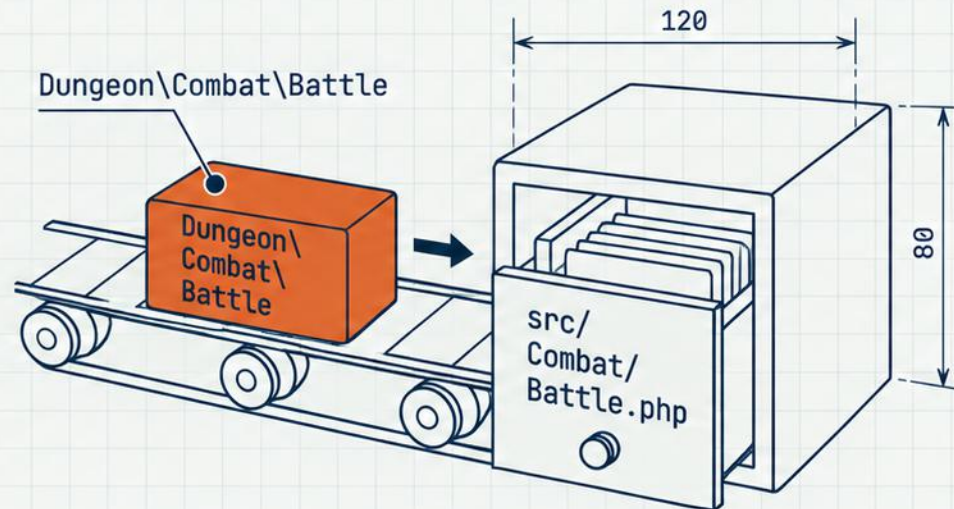
- | | | |
|---|-------------------------------|---|
| 1 | <code>composer install</code> | # installe ce que dit composer.lock |
| 2 | <code>composer update</code> | # cherche des versions plus récentes et réécrit composer.lock |

- `composer.lock` note la version précise de chaque paquet installé. Il est versionné avec le code : mêmes versions chez vous, chez votre binôme et sur GitHub.
- Pour les exercices fournis, toujours `composer install`.
- `vendor/` ne va jamais dans **Git** : c'est du code téléchargé, que `composer install` reconstruit. Le `.gitignore` du dépôt s'en occupe.



Le namespace est le chemin

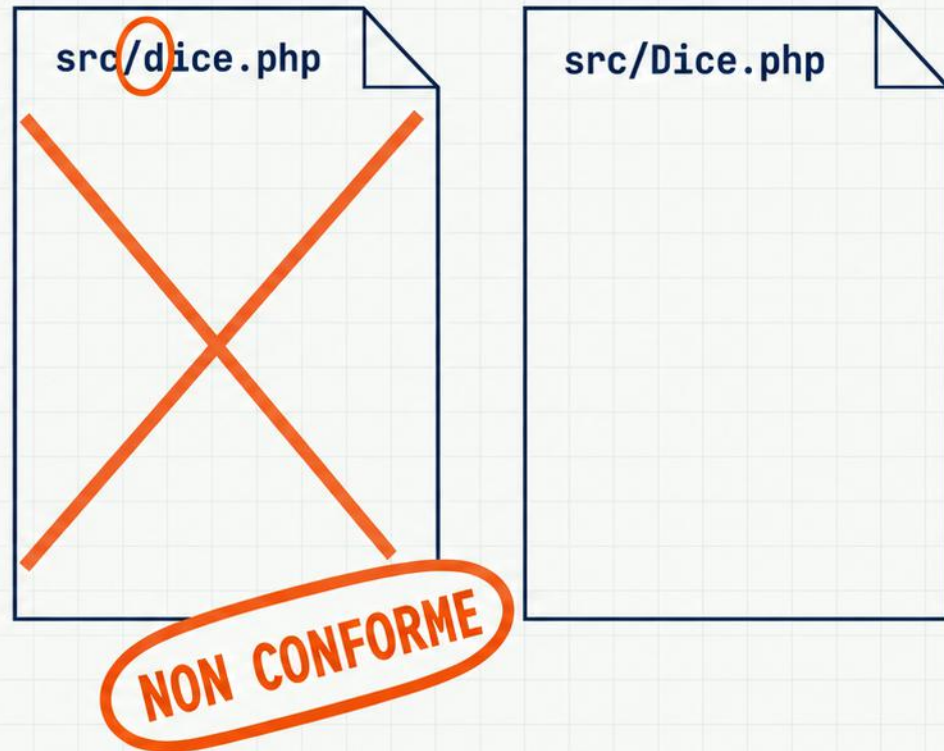
Classe	Fichier attendu
Dungeon\Dice	src/Dice.php
Dungeon\Hero	src/Hero.php
Dungeon\Combat\Battle	src/Combat/Battle.php



- Une seule ligne, require `__DIR__ . '/vendor/autoload.php';`, et PHP va chercher le fichier d'une classe au moment où vous l'utilisez.
- Avec `"Dungeon\\"`: `"src/"`, Composer remplace `Dungeon\` par `src/`, puis chaque `\` restant par `/`, et ajoute `.php`. Il ne fouille aucun dossier.

Le fichier existe, PHP ne le trouve pas

- 1 Le namespace du fichier ne correspond pas à son dossier.
- 2 Le nom de la classe ne correspond pas au nom du fichier. La majuscule compte : Linux la distingue, Windows non. D'où des erreurs qui n'apparaissent que sur GitHub.
- 3 Vous avez ajouté une correspondance dans `composer.json` sans relancer `composer dump-autoload`.



Un test est un petit programme qui vérifie votre code

- **Pest** est un lanceur de tests : il cherche les fichiers de tests, les exécute un par un, compare ce qu'il obtient à ce qui était attendu, et affiche un rapport. Une coche verte par test réussi, une croix rouge avec l'explication par test raté.
- Sans lui, vous écririez un script avec des `if` et des `echo` pour vérifier chaque comportement à la main.
- Les tests vont dans `tests/`, dans des fichiers qui se terminent par `Test.php`. Pest les trouve tout seul.
- **Pest** est le lanceur utilisé par **Laravel**.



Un nom en français, une vérification

```
1  use Dungeon\Dice;
2
3  test('un dé garde le nombre de faces qu\'on lui donne', function () { void {
4      expect((new Dice(6))->sides)->toBe(6);
5      expect((new Dice(20))->sides)->toBe(20);
6  })->group('niveau-1');
7
8  test('un lancer de dé reste entre 1 et le nombre de faces', function () { void {
9      $dice = new Dice(6);
10
11      for ($i = 0; $i < 100; $i++) {
12          expect($dice->roll())->toBeGreaterThanOrEqual(1)->toBeLessThanOrEqual(6);
13      }
14  })->group('niveau-1');
```

`test('...', function () { ... })` : un nom qui décrit le comportement attendu, et le code qui le vérifie.

`expect($valeur)->toBe($attendu)` : une **assertion**, c'est-à-dire une vérification. Pest en propose une centaine, la documentation les liste.

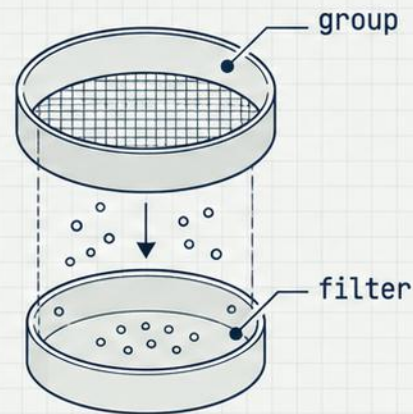
`->group('niveau-1')` : une étiquette, pour ne lancer qu'une partie de la suite.

--group pour le kata du jour, --filter pour le test en cours

1	<code>composer test</code>	<code># toute la suite</code>
2	<code>composer test -- --group=niveau-1</code>	<code># seulement les tests étiquetés niveau-1</code>
3	<code>composer test -- --filter=readonly</code>	<code># seulement les tests dont le nom contient « readonly »</code>
4		
5	<code>vendor/bin/pest --group=niveau-1</code>	<code># la même chose, en couleurs</code>

Les deux tirets seuls, --, disent à Composer « ce qui suit n'est pas pour toi, passe-le à Pest ». Sans eux, --group est ignoré et toute la suite tourne.

Sous PowerShell, le chemin s'écrit `.\vendor\bin\pest`.



Un test rouge est une adresse

FAIL Tests\Niveau1Test

✗ les points de vie ne descendent jamais sous zéro

0.01s

FAILED Tests\Niveau1Test > les points de vie ne descendent jamais sous zéro

Failed asserting that -989 is identical to 0.

at tests\Niveau1Test.php:59

57| \$hero->takeDamage(999);

58|

→ 59| expect(\$hero->hp)->toBe(0);

60| expect(\$hero->isAlive())->toBeFalse();

Tests: 1 failed, 9 passed (417 assertions)

TEST ROUGE

1. Le nom du test : la règle qui est violée.

2. Le message : attendu 0, obtenu -989.

3. La flèche → : la ligne exacte de la vérification.

-989, c'est $10 - 999$: la soustraction n'est pas bornée. Vous n'avez rien deviné, vous avez lu.

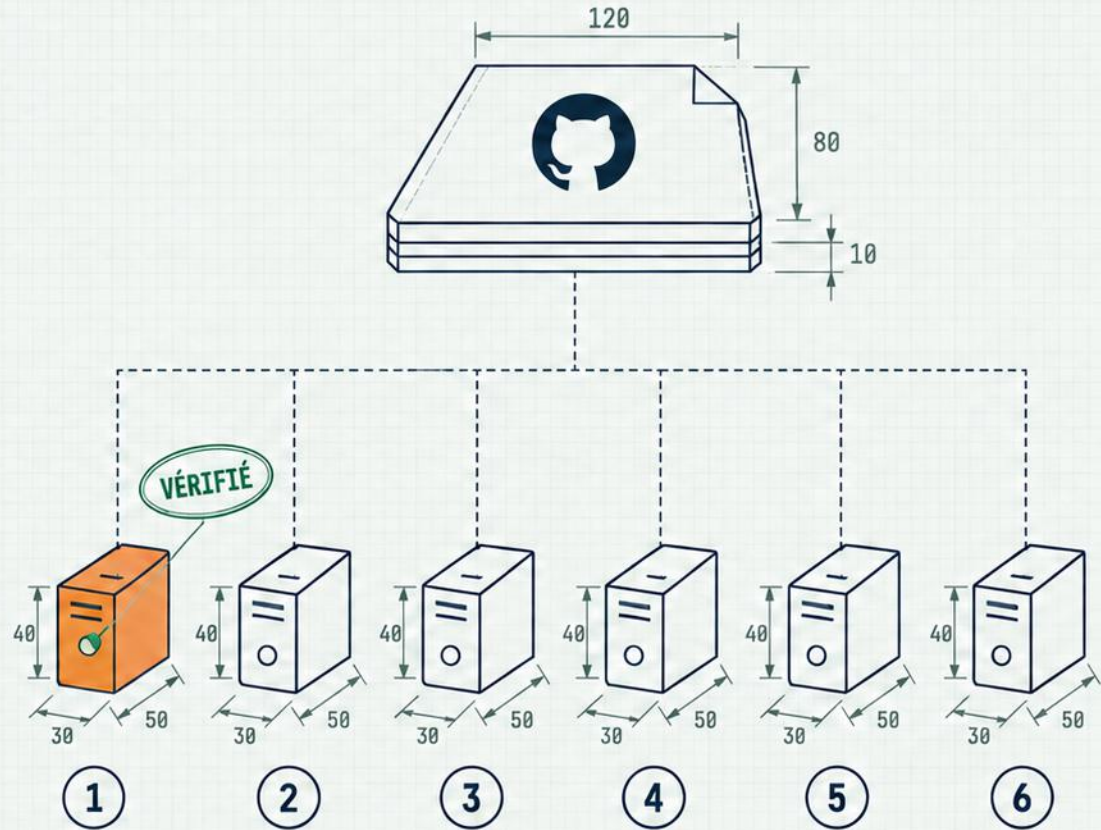
Les tests relancés sur une machine neuve

```
1  name: Tests
2
3  on: [push, pull_request]
4
5  jobs:
6    katas:
7      runs-on: ubuntu-latest
8      strategy:
9        fail-fast: false
10       matrix:
11         groupe: [niveau-1, encapsulation, niveau-2, niveau-3, niveau-4, niveau-5]
12       steps:
13         - uses: actions/checkout@v5
14         - uses: shivammathur/setup-php@v2
15           with:
16             php-version: "8.4"
17         - run: composer install --no-interaction --prefer-dist --no-progress
18         - run: composer test -- --group=${{ matrix.groupe }}
```

- À chaque git push, GitHub relance vos tests sur une machine vierge et affiche une coche verte ou une croix rouge sur votre commit.
- Le fichier est fourni dans `.github/workflows/`, vous n'avez rien à écrire. Une tâche, ou job, est une suite d'étapes exécutées sur une machine.

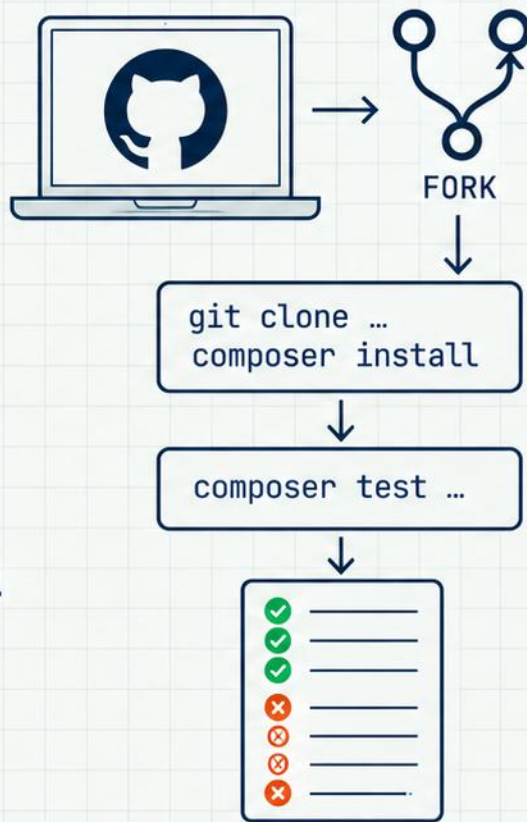
Six machines, une par groupe de tests

- **on:** `[push, pull_request]` : ce qui déclenche l'exécution.
- **runs-on:** `ubuntu-latest` : une machine Ubuntu vierge, créée pour l'occasion et détruite à la fin. Rien de votre ordinateur n'y est. Ce que cela vérifie : le projet s'installe et se teste à partir du seul contenu du dépôt.
- **matrix:** **groupe:** `[...]` : la tâche est lancée six fois en parallèle. Dans l'onglet Actions, six lignes, une par kata.
- Une exécution complète n'est verte que si les six passent. Aujourd'hui, votre objectif est la coche verte de la tâche niveau-1, pas celle du dépôt entier.



Fork, clone, install, test, push

- 1 `php --version` (8.4 minimum) et `composer --version` : les deux doivent répondre.
- 2 Forker `github.com/opmvpc/poo-katas-26` : votre propre copie.
- 3 Sur votre fork, onglet Actions : cliquer sur « I understand my workflows, go ahead and enable them ».
- 4 `git clone https://github.com/VOTRE-PSEUDO/poo-katas-26.git` puis `composer install`.
- 5 `composer test -- --group=niveau-1` : tout est rouge, c'est normal.
- 6 Écrire `d6()`, `d20()` et `roll()` dans `src/Dice.php`, relancer : trois verts, sept rouges.
- 7 `git add -A`, `git commit -m 'Kata 1 : Dice'`, `git push`, puis lire la tâche niveau-1 dans l'onglet Actions.



« À implémenter » : le message indique où travailler

```
1 public static function d6(): self
2 {
3     throw new \LogicException('À implémenter');
4 }
5
```

FAIL Tests\Niveau1Test

✗ Dice::d6() et Dice::d20() fabriquent les dés classiques

0.02s

FAILED Tests\Niveau1Test > Dice::d6() et Dice::d20()...

LogicException

À implémenter

at src\Dice.php:19

→ 19 | throw new \LogicException('À implémenter');

Tests: 1 failed (0 assertions)

TEST ROUGE

Les classes de src/ sont fournies avec leur signature. Le code des méthodes reste à écrire.

Trois sur dix : Dice est terminé, Hero reste à écrire

FAIL Tests\Niveau1Test

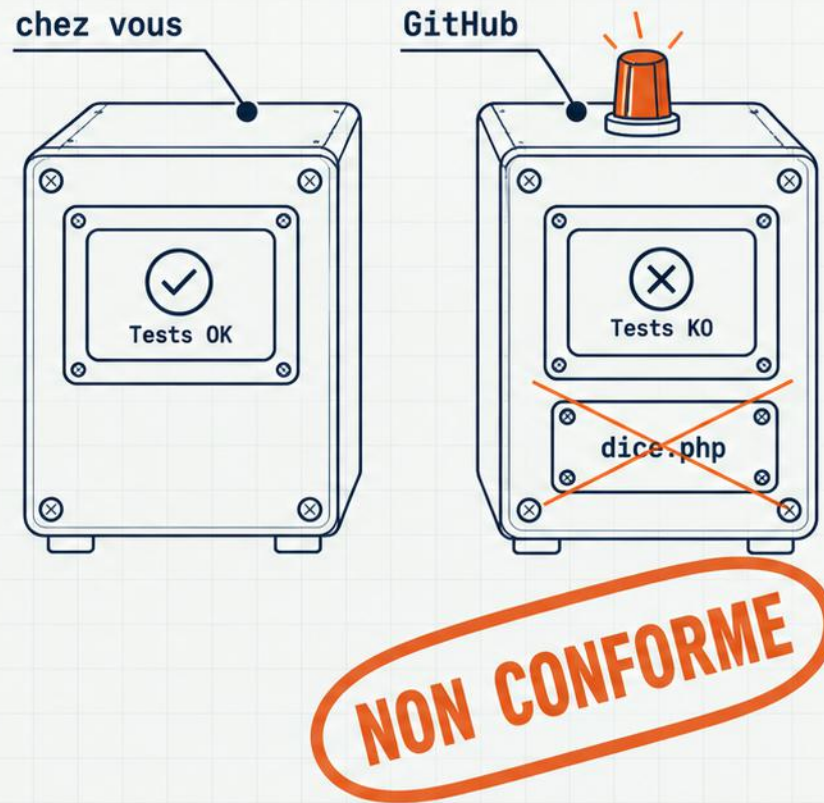
- ✓ un dé garde le nombre de faces qu'on lui donne 0.01s
- ✓ un lancer de dé reste entre 1 et le nombre de faces 0.01s
- ✓ Dice::d6() et Dice::d20() fabriquent les dés classiques VÉRIFIÉ
- ✗ un héros démarre avec tous ses points de vie 0.01s
- ✗ on peut choisir les points de vie et la force à la création
- ✗ takeDamage retire des points de vie
- ✗ les points de vie ne descendent jamais sous zéro
- ✗ heal ne dépasse jamais le maximum de points de vie
- ✗ un héros affiché donne "Arthur (7/10 PV)"
- ✗ le nom du héros est readonly : le réécrire lève une Error

Tests: 7 failed, 3 passed (405 assertions)

Un vert partiel est déjà une information utile.

La différence vient de l'environnement

- GitHub désactive les workflows sur un dépôt forké, par prudence. La première fois que vous ouvrez l'onglet Actions, cliquez sur le bouton vert, une seule fois. Tant que ce n'est pas fait, aucun push ne lance les tests.
- Si la tâche niveau-1 est rouge alors que les mêmes tests passent chez vous : un fichier oublié au moment du commit, ou une majuscule dans un nom de fichier, `src/dice.php` au lieu de `src/Dice.php`. Windows ne fait pas la différence, Ubuntu si.



Kata niveau 1 : Dice et Hero

```
composer test -- --group=niveau-1
```

- **Dice** : `new Dice(int $sides)`, `roll()` dans `[1, sides]`, fabriques `Dice::d6()` et `Dice::d20()`.
- **Hero** : `hp` démarre à `maxHp`, `takeDamage()` jamais sous 0, `heal()` jamais au-dessus de `maxHp`, `isAlive()`, `__toString()` donne exactement "Arthur (7/10 PV)".
- `name` est `readonly`, `hp` et `maxHp` sont `public private(set)` : deux tests vérifient qu'une écriture de l'extérieur lève une `Error`.
- Poussez. La tâche `niveau-1` doit être verte dans l'onglet Actions. Comptez 30 minutes.

