

Tests Unitaires de Gaëtan Corin

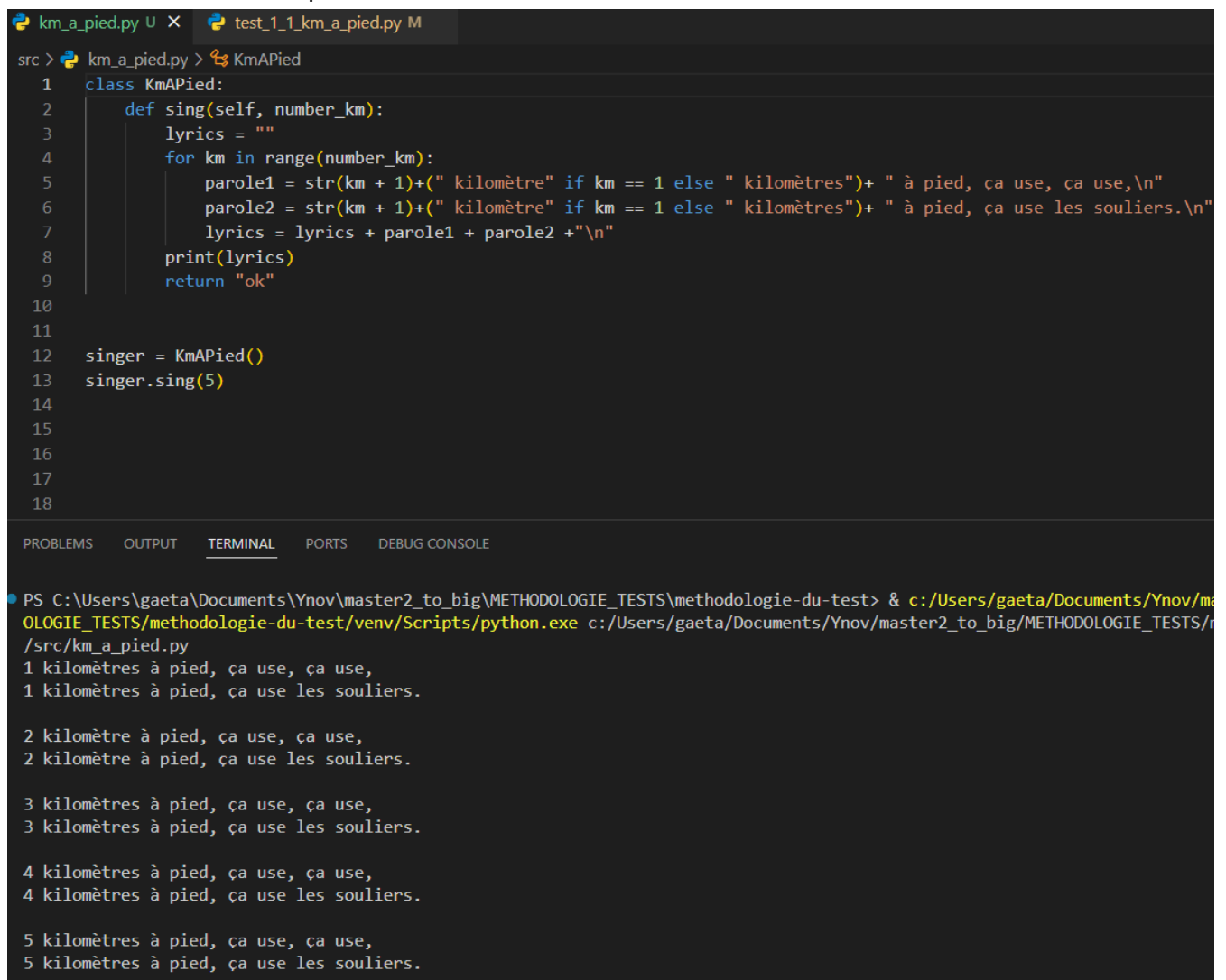
1km à pieds:

Je commence par créer la fonction qui crée la chanson.

Pour pouvoir avoir un retour de la chanson, je ne peux pas simplement faire des prints, car je ne pourrais pas tester le résultat.

Je vais donc créer la chanson en tant que grand string, puis printer le string.

Je divise la chanson en 2 phrases:



```
km_a_pied.py U X test_1_1_km_a_pied.py M
src > km_a_pied.py > KmAPied
1 class KmAPied:
2     def sing(self, number_km):
3         lyrics = ""
4         for km in range(number_km):
5             parole1 = str(km + 1)+(" kilomètre" if km == 1 else " kilomètres")+ " à pied, ça use, ça use,\n"
6             parole2 = str(km + 1)+(" kilomètre" if km == 1 else " kilomètres")+ " à pied, ça use les souliers.\n"
7             lyrics = lyrics + parole1 + parole2 + "\n"
8         print(lyrics)
9         return "ok"
10
11
12 singer = KmAPied()
13 singer.sing(5)
14
15
16
17
18
```

PROBLEMS OUTPUT **TERMINAL** PORTS DEBUG CONSOLE

```
PS C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test> & c:/Users/gaeta/Documents/Ynov/master2_to_big/METHODOLOGIE_TESTS/venv/Scripts/python.exe c:/Users/gaeta/Documents/Ynov/master2_to_big/METHODOLOGIE_TESTS/src/km_a_pied.py
1 kilomètres à pied, ça use, ça use,
1 kilomètres à pied, ça use les souliers.

2 kilomètre à pied, ça use, ça use,
2 kilomètre à pied, ça use les souliers.

3 kilomètres à pied, ça use, ça use,
3 kilomètres à pied, ça use les souliers.

4 kilomètres à pied, ça use, ça use,
4 kilomètres à pied, ça use les souliers.

5 kilomètres à pied, ça use, ça use,
5 kilomètres à pied, ça use les souliers.
```

La fonction marche mais je ne suis pas satisfait car je vais avoir des difficultés à séparer les différentes phrases lors des tests.

Je crée donc des fonctions spécifiques qui vont générer chaque phrase. Cela me permettra de pouvoir les tester de manière séparée.

```
km_a_pied.py U X test_1_1_km_a_pied.py M
src > km_a_pied.py > KmAPied
1 class KmAPied:
2     def sing(self, number_km):
3         lyrics = ""
4         for km in range(number_km):
5             parole1 = self.create_parole1(km)
6             parole2 = self.create_parole2(km)
7             lyrics = lyrics + parole1 + parole2 + "\n"
8         print(lyrics)
9         return lyrics
10
11     def create_parole1(self, km):
12         return str(km + 1) + (" kilomètre" if km == 1 else " kilomètres") + " à pied, ça use, ça use,\n"
13
14     def create_parole2(self, km):
15         return str(km + 1) + (" kilomètre" if km == 1 else " kilomètres") + " à pied, ça use les souliers.\n"
16
17
18 singer = KmAPied()
19 singer.sing(5)

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

OLOGIE_TESTS/methodologie-du-test/venv/Scripts/python.exe c:/Users/gaeta/Documents/Ynov/master2_to_big/METHODOLOGIE_
/src/km_a_pied.py
1 kilomètres à pied, ça use, ça use,
1 kilomètres à pied, ça use les souliers.

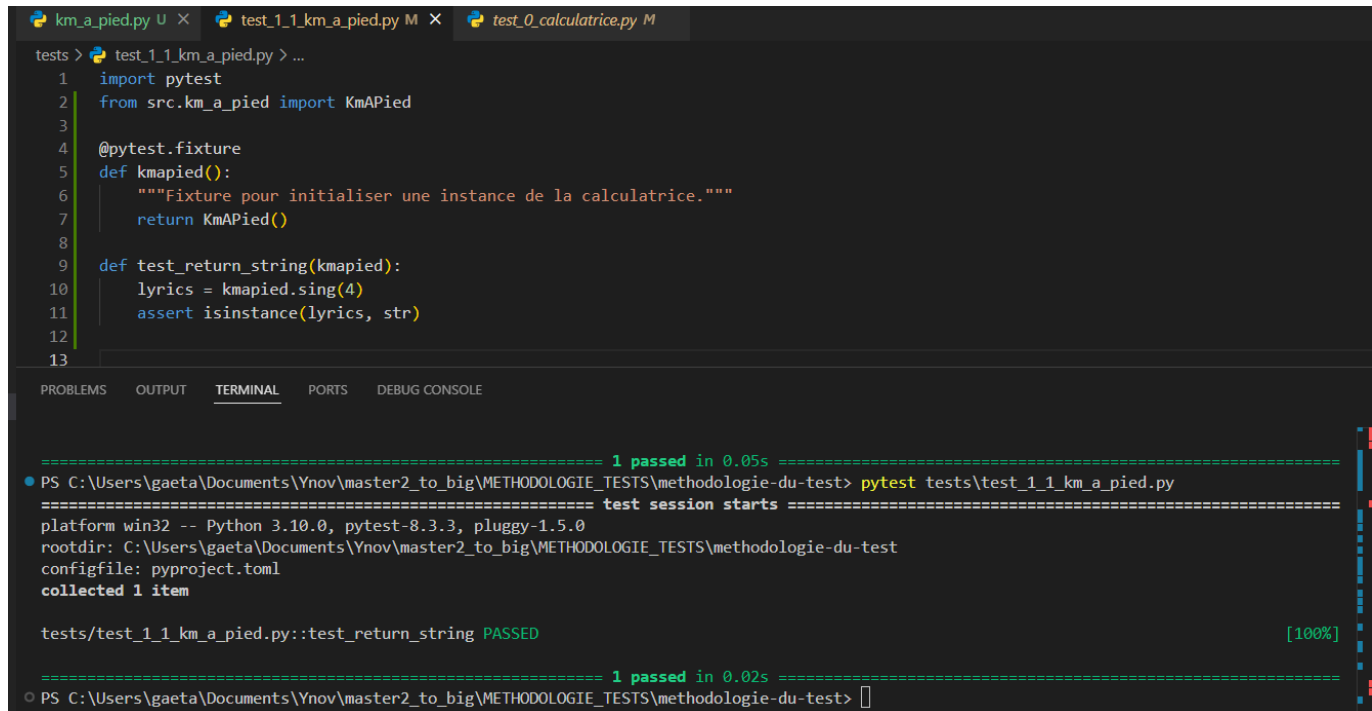
2 kilomètre à pied, ça use, ça use,
2 kilomètre à pied, ça use les souliers.

3 kilomètres à pied, ça use, ça use,
3 kilomètres à pied, ça use les souliers.

4 kilomètres à pied, ça use, ça use,
4 kilomètres à pied, ça use les souliers.

5 kilomètres à pied, ça use, ça use,
5 kilomètres à pied, ça use les souliers.
```

Puis je réalise mon premier test unitaire.
Mon premier test va vérifier que la chanson retourne un string.



```
km_a_pied.py U X test_1_1_km_a_pied.py M X test_0_calculatrice.py M
tests > test_1_1_km_a_pied.py > ...
1 import pytest
2 from src.km_a_pied import KmAPied
3
4 @pytest.fixture
5 def kmapied():
6     """Fixture pour initialiser une instance de la calculatrice."""
7     return KmAPied()
8
9 def test_return_string(kmapied):
10     lyrics = kmapied.sing(4)
11     assert isinstance(lyrics, str)
12
13
```

PROBLEMS OUTPUT **TERMINAL** PORTS DEBUG CONSOLE

```
===== 1 passed in 0.05s =====
● PS C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test> pytest tests\test_1_1_km_a_pied.py
===== test session starts =====
platform win32 -- Python 3.10.0, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test
configfile: pyproject.toml
collected 1 item

tests/test_1_1_km_a_pied.py::test_return_string PASSED [100%]

===== 1 passed in 0.02s =====
○ PS C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test> 
```

(j'ai eu des difficultés pour la ligne de commande "pytest tests\test_1_1_km_a_pied.py" me permettant de lancer uniquement le fichier de test dont j'ai besoin.)

Maintenant, je vais tester chaque phrase séparément pour vérifier le singulier et le pluriel du mot "kilomètre"

```
km_a_pied.py U X test_1_1_km_a_pied.py M X calculatrice.py
tests > test_1_1_km_a_pied.py > test_singulier_parole1
5 def kmapied():
6     # Fixture pour initialiser une instance de la calculatrice.
7     return KmAPIed()
8
9 def test_return_string(kmapied):
10    lyrics = kmapied.sing(4)
11    assert isinstance(lyrics, str)
12
13 def test_singulier_parole1(kmapied):
14    parole1 = kmapied.create_parole1(1)
15    assert "1 kilomètre à pied, ça use, ça use," == parole1
16
17

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

tests/test_1_1_km_a_pied.py::test_singulier_parole1 FAILED

===== FAILURES =====
test_singulier_parole1

kmapied = <src.km_a_pied.KmAPIed object at 0x0000021113B94610>

def test_singulier_parole1(kmapied):
    parole1 = kmapied.create_parole1(1)
    assert "1 kilomètre à pied, ça use, ça use," == parole1
E     AssertionError: assert '1 kilomètre ... use, ça use,' == '2 kilomètre ...se, ça use,\n'
E
E     - 2 kilomètre à pied, ça use, ça use,
E     ? ^
E     + 1 kilomètre à pied, ça use, ça use,
E     ? ^

tests\test_1_1_km_a_pied.py:15: AssertionError
```

j'ai des problèmes a cause de la construction de mes phrases qui se font avec des "km +1", je réalise donc le "km+1" en dehors des fonctions de créations des phrases 1 et 2.

```
class KmAPIed:
    def sing(self, number_km):
        km = Km+1
        parole1 = self.create_parole1(km)
        parole2 = self.create_parole2(km)
        lyrics = lyrics + parole1 + parole2 + "\n"
        print(lyrics)
        return lyrics

    def create_parole1(self, km):
        return str(km)+(" kilomètre" if km == 1 else " kilomètres")+ " à pied, ça use, ça use,\n"

    def create_parole2(self, km):
        return str(km)+(" kilomètre" if km == 1 else " kilomètres")+ " à pied, ça use les souliers.\n"
```

Une fois le test fonctionnel sur le kilomètre au singulier de la parole1, je fais la même chose pour le pluriel.

```
1 import pytest
2 from src.km_a_pied import KmAPied
3
4 @pytest.fixture
5 def kmapied():
6     """Fixture pour initialiser une instance de la calculatrice."""
7     return KmAPied()
8
9 def test_return_string(kmapied):
10     lyrics = kmapied.sing(4)
11     assert isinstance(lyrics, str)
12
13 def test_singulier_parole1(kmapied):
14     parole1 = kmapied.create_parole1(1)
15     assert "1 kilomètre à pied, ça use, ça use,\n" == parole1
16
17 def test_pluriel_parole1(kmapied):
18     parole1 = kmapied.create_parole1(2)
19     assert "2 kilomètres à pied, ça use, ça use,\n" == parole1
20
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
===== test session starts =====
platform win32 -- Python 3.10.0, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test
configfile: pyproject.toml
collected 3 items

tests/test_1_1_km_a_pied.py::test_return_string PASSED
tests/test_1_1_km_a_pied.py::test_singulier_parole1 PASSED
tests/test_1_1_km_a_pied.py::test_pluriel_parole1 PASSED
```

puis je fais la même chose pour la parole 2:

```
21 def test_singulier_parole2(kmapied):
22     parole1 = kmapied.create_parole2(1)
23     assert "1 kilomètre à pied, ça use les souliers.\n" == parole1
24
25 def test_pluriel_parole2(kmapied):
26     parole1 = kmapied.create_parole2(2)
27     assert "2 kilomètres à pied, ça use les souliers.\n" == parole1
28
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
PS C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-
===== test session starts =====
platform win32 -- Python 3.10.0, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test
configfile: pyproject.toml
collected 5 items

tests/test_1_1_km_a_pied.py::test_return_string PASSED
tests/test_1_1_km_a_pied.py::test_singulier_parole1 PASSED
tests/test_1_1_km_a_pied.py::test_pluriel_parole1 PASSED
tests/test_1_1_km_a_pied.py::test_singulier_parole2 PASSED
tests/test_1_1_km_a_pied.py::test_pluriel_parole2 PASSED

===== 5 passed in 0.04s =====
```

Je crée ensuite un test qui va compter les retour a la ligne.
Je doit avoir 3 retour a la ligne par kilomètre sur la chanson.

```
29 def test_verify_linebreak(kmapied):
30     km = 3
31     sing = kmapied.sing(km)
32     linebreak_number = sing.count("\n")
33     assert km * 3 == linebreak_number
34
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

15

```
● PS C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test> pytest
===== test session starts =====
platform win32 -- Python 3.10.0, pytest-8.3.3, pluggy-1.5.0
rootdir: C:\Users\gaeta\Documents\Ynov\master2_to_big\METHODOLOGIE_TESTS\methodologie-du-test
configfile: pyproject.toml
collected 6 items

tests/test_1_1_km_a_pied.py::test_return_string PASSED
tests/test_1_1_km_a_pied.py::test_singulier_parole1 PASSED
tests/test_1_1_km_a_pied.py::test_pluriel_parole1 PASSED
tests/test_1_1_km_a_pied.py::test_singulier_parole2 PASSED
tests/test_1_1_km_a_pied.py::test_pluriel_parole2 PASSED
tests/test_1_1_km_a_pied.py::test_verify_linebreak PASSED
```

FizzBuzz:

J'ai créé l'object FizzBuzz avec ces méthodes permettant de transformer les multiple de 3 en fizz, les multiple de 5 en buzz, et les multiple de 3 et de 5 en fizzbuzz.

```
class FizzBuzz:
    def process_numbers(self, number):
        number_transformed = ""
        for nb in range(number):
            number_transformed = number_transformed + self.transform_number(nb+1) + "\n"
        print(number_transformed)
        return number_transformed

    def transform_number(self, number):
        print(number)
        fizzbuzz = self.is_fizzbuzz(number)
        fizz = self.is_fizz(number)
        buzz = self.is_buzz(number)
        if fizzbuzz:
            return "FizzBuzz"
        elif fizz:
            return "Fizz"
        elif buzz:
            return "Buzz"
        else:
            return str(number)

    def is_fizz(self, number):
        return number % 3 == 0

    def is_buzz(self, number):
        return number % 5 == 0

    def is_fizzbuzz(self, number):
        fizz = self.is_fizz(number)
        buzz = self.is_buzz(number)
        return (fizz and buzz)
```

J'attaque mes tests afin de tester le fonctionnement de l'objet "FizzBuzz".

```
@pytest.fixture
def fizzbuzz():
    """Fixture pour initialiser une instance de la calculatrice."""
    return FizzBuzz()

def test_return_string(fizzbuzz):
    string = fizzbuzz.process_numbers(2)
    assert isinstance(string, str)

def test_not_fizz(fizzbuzz):
    assert False == fizzbuzz.is_fizz(2)

def test_is_fizz(fizzbuzz):
    assert True == fizzbuzz.is_fizz(3)

def test_not_buzz(fizzbuzz):
    assert False == fizzbuzz.is_buzz(4)

def test_is_buzz(fizzbuzz):
    assert True == fizzbuzz.is_buzz(5)

def test_not_buzz(fizzbuzz):
    assert False == fizzbuzz.is_buzz(4)

def test_not_fizzbuzz(fizzbuzz):
    assert False == fizzbuzz.is_fizzbuzz(5)

def test_is_fizzbuzz(fizzbuzz):
    ⚡ assert True == fizzbuzz.is_fizzbuzz(15)
```

return_string verifie que la valeur final retourner de toute la boucle de nombre est bien égale a un string.

is_fizz teste si la fonction qui vérifie que le nombre est un multiple de 3. Il retourne True ou False

is_buzz teste si la fonction qui vérifie que le nombre est un multiple de 5. Il retourne True ou False

is_fizzbuzz teste si la fonction qui vérifie que le nombre est un multiple de 3 et de 5. Il retourne True ou False

Bowling:

Le programme implémenter permet de faire 10 jeux de 1 ou 2 lancers, double les points du prochain lancé en cas de spare, et double les points des 2 prochains lancé en cas de strike. Il ne prend pas en compte le fait de pouvoir jouer au dernier lancé si il fait un spare ou un strike.

```
import random
class Game:
    scoring = 0
    launch = 0
    spare = False
    strike = 0
    bonus_strike = 0
    bonus_spare = 0

    def play_party(self):
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()
        self.play_launch()

    def play_launch(self):
        number_random1 = random.randint(0, 10)
        self.roll(number_random1)
        if number_random1 == 10:
            self.strike = 2
        else:
            number_random2 = random.randint(0, 10 - number_random1)
            self.roll(number_random2)
            if number_random1 + number_random2 == 10:
                self.spare = True
        self.launch += 1
        print(self.scoring)

    def roll(self, pins) -> None:
        print("roll", pins)
        if pins > 10:
```

```

        raise ValueError("Score supérieur a 10 interdit")
    if self.strike > 0:
        self.strike -= 1
        self.bonus_strike += pins
    elif self.spare == True:
        self.spare == False
        self.bonus_spare += pins
    self.scoring += (pins + self.bonus_strike + self.bonus_spare)
    self.bonus_strike = 0
    self.bonus_spare = 0
    pass

def score(self) -> int:
    return self.scoring

```

voici les codes coté tests.

Certains fonctionnes et d'autres non a cause de la manière de comment ceux-ci ont été implémenté

```

from src.bowling_game import Game
import pytest

@pytest.fixture
def game():
    """Fixture pour initialiser une instance de la calculatrice."""
    return Game()

def test_roll_value_error(game):
    with pytest.raises(ValueError):
        game.roll(11)

def test_roll(game):
    game.roll(8)
    scoring = game.score()
    assert 8 == scoring

## Test non fonctionnels
def test_spare(game):
    game.roll(4)
    game.roll(6)
    assert True == game.spare

def test_spare_after_one_roll(game):

```

```

game.roll(4)
game.roll(6)
game.roll(2)
assert False == game.spare

def test_strike(game):
    game.roll(10)
    assert 2 == int(game.strike)

def test_strike_after_one_roll(game):
    game.roll(10)
    game.roll(2)
    assert 1 == game.strike

def test_strike_after_two_roll(game):
    game.roll(10)
    game.roll(2)
    game.roll(3)
    assert 0 == game.strike

```

Le test “test_roll_value_error(game)” permet de vérifier que une valeur de 11 quilles qui tombe est impossible et sort une erreur.

le test “test_roll(game)” permet de vérifier que le score s’incrémente correctement en fonction du nombre de quille qui tombe

le test “test_spare(game)” permet de vérifier que le programme compte bien le spare comme actif si il y a un spare

La fonction “test_spare_after_one_roll(game)” vérifie que l’activation du spare c’est correctement désactivé après un lancé suivant

le test “test_strike(game)” permet de vérifier que le programme compte bien le strike comme actif si il y a un strike

La fonction “test_strike_after_one_roll(game)” vérifie que l’activation du strike n’est pas encore désactivé mais a bien eu une désincrémentation après un lancé suivant

La fonction “test_strike_after_two_roll(game)” vérifie que l’activation du strike est désactivée après 2 lancers.