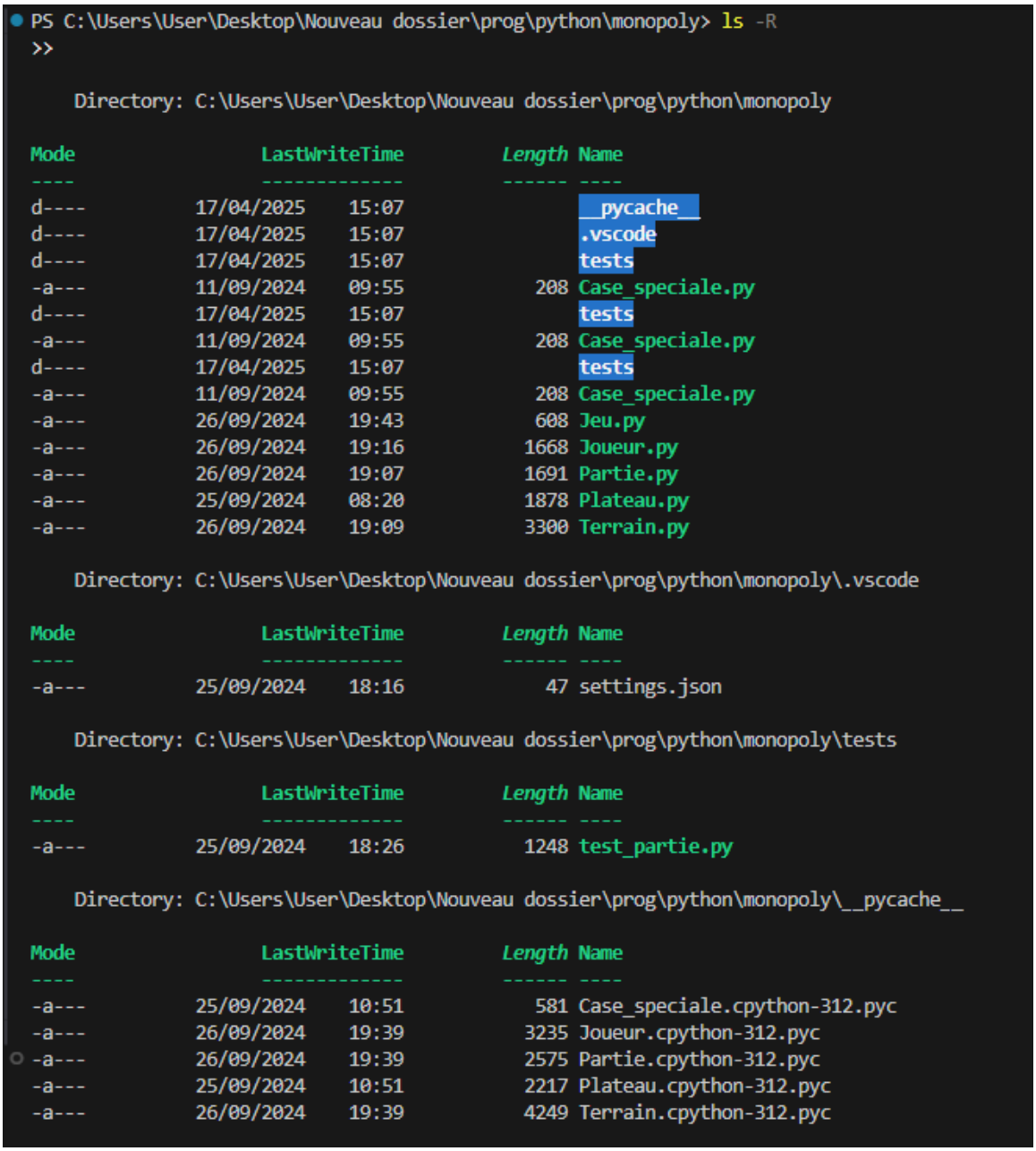
Projet Monopoly

Structure du projet : Organisation complète des fichiers sources, classes, répertoires de test, configuration VSCode, etc.



```
PS C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly> ls -R
>>

    Directory: C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d----          17/04/2025    15:07                __pycache__
d----          17/04/2025    15:07                .vscode
d----          17/04/2025    15:07                tests
-a---          11/09/2024    09:55            208 Case_speciale.py
d----          17/04/2025    15:07                tests
-a---          11/09/2024    09:55            208 Case_speciale.py
d----          17/04/2025    15:07                tests
-a---          11/09/2024    09:55            208 Case_speciale.py
-a---          26/09/2024    19:43            608 Jeu.py
-a---          26/09/2024    19:16           1668 Joueur.py
-a---          26/09/2024    19:07           1691 Partie.py
-a---          25/09/2024    08:20           1878 Plateau.py
-a---          26/09/2024    19:09           3300 Terrain.py

    Directory: C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly\.vscode

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          25/09/2024    18:16             47 settings.json

    Directory: C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly\tests

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          25/09/2024    18:26           1248 test_partie.py

    Directory: C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly\__pycache__

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a---          25/09/2024    10:51            581 Case_speciale.cpython-312.pyc
-a---          26/09/2024    19:39           3235 Joueur.cpython-312.pyc
-a---          26/09/2024    19:39           2575 Partie.cpython-312.pyc
-a---          25/09/2024    10:51           2217 Plateau.cpython-312.pyc
-a---          26/09/2024    19:39           4249 Terrain.cpython-312.pyc
```

Classe Joueur : Illustration de la programmation orientée objet avec attributs, méthodes (tirer_de, déplacement, paiement...) et logique métier.

```
PS C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly> cat Joueur.py
>>
from random import randint

class Joueur:

    def __init__(self, nom):
        self.nom = nom
        self.compte = 1500
        self.prop = []
        self.position = (0, 0)

    def __str__(self):
        return f"Le joueur s'appelle {self.nom}, il a sur son compte {self.compte}€."

    def tirer_de(self):
        return randint(1, 6)

    def deplacement(self, nb_cases):
        nb_cases = self.tirer_de()
        print(f"{self.nom} a tiré un {nb_cases}.")

        x, y = self.position
        y = (y + nb_cases) % 6
        x += (y // 6)
        x = x % 4

        self.position = (x, y)
        print(f"{self.nom} se déplace à la position {self.position}.")

    def acheter(self, terrain):
        if terrain.est_achetable() and self.compte >= terrain.prix:
            self.compte -= terrain.prix
            self.prop.append(terrain)
            terrain.prop = self
            print(f"{self.nom} a acheté {terrain.nom} pour {terrain.prix}€. Il lui reste {self.compte}€.")
        else:
            message = f"{terrain.nom} est déjà possédé par un autre joueur." if terrain.prop else f"{self.nom} n'a pas assez d'argent."
            print(message)

    def payer(self, terrain, autre_joueur):
        loyer = terrain.loyer
        if self.compte >= loyer:
            self.compte -= loyer
            autre_joueur.compte += loyer
            print(f"{self.nom} a payé {loyer}€ à {autre_joueur.nom} pour {terrain.nom}.")
        else:
            print(f"{self.nom} n'a pas assez d'argent pour payer le loyer de {terrain.nom}.")
```

Tests unitaires : Fichier de tests automatisés avec Python `unittest`, validation de la méthode `joueur_faillite()` à travers plusieurs cas.

```
PS C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly> cat tests/test_partie.py
>>
import unittest
from Partie import Partie

class TestPartie(unittest.TestCase):

    def setUp(self):
        self.partie = Partie()

    def test_joueur_faillite_no_faillite(self):
        self.partie.joueurs = [
            {'argent': 100},
            {'argent': 50},
            {'argent': 0}
        ]
        self.assertFalse(self.partie.joueur_faillite())

    def test_joueur_faillite_with_faillite(self):
        self.partie.joueurs = [
            {'argent': 100},
            {'argent': -10},
            {'argent': 50}
        ]
        self.assertTrue(self.partie.joueur_faillite())

    def test_joueur_faillite_all_positive(self):
        self.partie.joueurs = [
            {'argent': 100},
            {'argent': 200},
            {'argent': 300}
        ]
        self.assertFalse(self.partie.joueur_faillite())

    def test_joueur_faillite_empty_list(self):
        self.partie.joueurs = []
        self.assertFalse(self.partie.joueur_faillite())

    def test_joueur_faillite_multiple_negative(self):
        self.partie.joueurs = [
            {'argent': -50},
            {'argent': 100},
            {'argent': -30}
        ]
        self.assertTrue(self.partie.joueur_faillite())
```

Partie.py : Coordination des classes, gestion des actions, boucle de jeu, preuve d'un programme fonctionnel modulaire.

```
PS C:\Users\User\Desktop\Nouveau dossier\prog\python\monopoly> cat Partie.py
>>
from Plateau import Plateau
from Terrain import Terrain

class Partie:
            def __init__(self, liste_joueur, Plateau):
                self.liste_joueur = liste_joueur
                self.Plateau = Plateau

            def avoir_joueur_avec_nom(self, nom):
                for joueur in self.liste_joueur:
                    if joueur.nom == nom:
                        return joueur
                return None

            def choix_action(self, joueur):
                pass

            def deplacement(self, joueur):
                nb_cases = joueur.tirer_de()
                joueur.deplacement(nb_cases)
                i, j = joueur.position
                case_arrivee = self.Plateau.avoir_terrain_i_j(i, j)
                print(f"{joueur.nom} a tiré un {nb_cases} et se trouve maintenant à la position {joueur.position} ({case_arrivee.nom}).")
                return case_arrivee

            def traitement_post_deplacement(self, joueur, case):
                pass

            def tour(self, joueur):
                if joueur is None:
                    raise ValueError("Le joueur ne peut pas être None")

                print(f"Tour de {joueur.nom}")
                print()

                self.choix_action(joueur)
                case_joueur = self.deplacement(joueur)
                self.traitement_post_deplacement(joueur, case_joueur)

            def joueur_faillite(self):
                for joueur in self.liste_joueur:
                    if joueur.compte < 0:
                        return True
                return False
```