```python
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 11 08:41:16 2023

@author: Ran XIN
"""

import matplotlib.pyplot as plt
import random
# from enum import Enum

firstPartie = True

########################################
# Carte
########################################
class Carte:
    def __init__(self, figure, couleur, valeur):
        self.figure     = figure;
        self.couleur    = couleur;
        self.valeur     = valeur;

    def __str__(self):
        return "{} de {} : valeur {}".format(self.figure.upper(), self.couleur.upper(), self.valeur)

    # afficher une seule carte
    def plot(self, axes=None):
        # import img
        if self.valeur < 10: # pour les cartes de 1 a 9
            imgNom = "cartes/test-0{}-{}-img.png".format(self.valeur, self.couleur.lower());
        else: # pour les cartes 10, valet, dame et roi
            fig = self.figure.upper()
            options = {'10'     : '10',
                       'VALET'  : 'V',
                       'DAME'   : 'D',
                       'ROI'    : 'R'};
            imgNom = "cartes/test-{}-{}-img.png".format(options[fig], self.couleur.lower());
        img = plt.imread(imgNom)
        # set axes
        if axes is not None:
```

```
            axes.axes.clear()
            imgplot = axes.imshow(img)
            axes.axis('off')
        else:
            plt.clf()
            imgplot = plt.imshow(img)
            plt.axis('off')
        return imgplot



########################################
# PaquetCartes
########################################
class PaquetCartes:
    # creer un paquet vide de cartes
    def __init__(self):
        self.listeCartesDuPaquet = []

    # imprimer toutes les cartes dans le paquet
    def __str__(self):
        return "\n".join(c.__str__() for c in self.listeCartesDuPaquet)

    # retourner le longueur de la liste des cartes (nombre )
    def __len__(self):
        return len(self.listeCartesDuPaquet)

    # melanger toutes les cartes dans ce paquet
    def melanger(self):
        random.shuffle(self.listeCartesDuPaquet);

    # supprimer la premiere carte de ce paquet
    def tirerCarte(self):
        if len(self.listeCartesDuPaquet) > 0:
            carteRetire = self.listeCartesDuPaquet[0]
            self.listeCartesDuPaquet.pop(0);
            return carteRetire;
        else:
            return None;
```

```
    # ajouter la carte a la fin de ce paquet
    def ajouterCarteDansPaquet(self, carte):
        self.listeCartesDuPaquet.append(carte)

    # retourner la somme des valeurs de ce paquet
    def getValeurDuPaquet(self):
        val = 0;
        for c in self.listeCartesDuPaquet:
            val += c.valeur;
        return val;

    # supprimer toutes les cartes
    def clearPaquet(self):
        for c in self.listeCartesDuPaquet:
            c.remove()


    def plot(self,fig=None,left=0,bottom=0,width=0.2,height=0.2,shift=0.05):
        # affiche l'ensemble des cartes du paquet en les décalant
        if fig == None: fig=plt.figure()
        for c in self.listeCartesDuPaquet:
            axes=fig.add_axes([left,bottom,width,height])
            c.plot(axes)
            left+=shift
            if left >0.9:
                left=0
                bottom +=0.15



########################################
# JeuCartesBlackJack
########################################
class JeuCartesBlackJack(PaquetCartes):
    def __init__(self, N):
        super(JeuCartesBlackJack, self).__init__()
        self.jeuCartesEnsemble = PaquetCartes()
        carteFigure     = ('as', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'Valet', 'Dame', 'Roi')
        carteCouleur    = ('Carreau', 'Coeur', 'Pique', 'Trefle')
        carteValeur     = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10)
```

```
        i = 0
        m = 0
        n = 0
        for i in range(0,N): # loop de nombre de paquets de 52 cartes utilises
            for m in range(0,4): # loop de couleur
                for n in range(0,13): # loop de figure et valeur
                    self.jeuCartesEnsemble.ajouterCarteDansPaquet(Carte(carteFigure[n], carteCouleur[m], carteValeur[n]))
                    # print(Carte(carteFigure[n], carteCouleur[m], carteValeur[n]))
        self.jeuCartesEnsemble.melanger() # melanger les cartes
        # print('Il y a {} de cartes dans ce jeu.'.format(len(self.jeuCartesEnsemble)))

    # afficher toutes les cartes dans le jeu cartes Black Jack
    def plot(self):
        self.jeuCartesEnsemble.plot()



########################################
# Joueur
########################################
class Joueur:
    def __init__(self, prenom, argent, mise=0, etat=None, paquetJoueur=None):
        self.prenom       = prenom
        self.argent       = argent
        self.mise         = mise
        self.etat         = etat
        self.paquetJoueur = paquetJoueur if paquetJoueur is not None else PaquetCartes()


    def __str__(self):
        return '{} : {} Euros Etat: {} \n'.format(self.prenom, self.argent, self.etat)

    def setEtatJoueur(self, nouvelEtat):
        etatJoueur = ['run', 'stop', 'perdu', 'inactif', 'gagne']
        if nouvelEtat in etatJoueur:
            self.etat = nouvelEtat;
        else:
            raise NotImplementedError('Failed to set the player state')

    def ajouterCarteDansPaquetJoueur(self, carte):
```

```
        self.paquetJoueur.ajouterCarteDansPaquet(carte)

    def addArgentJoueur(self, argent):
        self.argent = self.argent + argent

    def clearPaquetJoueur(self):
        self.paquetJoueur.clearPaquet()

    def miseJoueur(self, miseMin):
        mise = int(input('La somme que souhaite miser le joueur : '))
        if mise >= miseMin:
            return mise
        else:
            raise NotImplementedError('Failed to update')

    def action(self):
        actionAutorisee = ['1', '2', 'p', 's']
        print('Vous vouslez faire : ')
        act = input('(tirer 1 ou 2 cartes, p : passer son tour, s : arrêter) \n')
        if act in actionAutorisee:
            return act
        else:
            raise NotImplementedError('Failed to act')

    def plot(self,fig=None,left=0,bottom=0,width=0.2,height=0.2,shift=0.05):
        self.paquetJoueur.plot(fig,left,bottom,width,height,shift)



class Partie:
    def __init__(self, nomDuJeu, listeJoueurs, miseMin):
        self.nomDuJeu       = nomDuJeu
        self.listeJoueurs   = listeJoueurs
        self.miseMin        = miseMin
        self.setEtatJoueurs()
        global firstPartie
        if not firstPartie:
            for j in listeJoueurs:
                j.clearPaquetJoueur
        firstPartie = False
```

```
    def __str__(self):
        tmp = '{} \nIl y a {} joueurs\n'.format(self.nomDuJeu, len(self.listeJoueurs))
        for j in self.listeJoueurs:
            tmp = tmp + j.__str__()
        return tmp

    def setNomDuJeu(self, nomDuJeu):
        self.nomDuJeu = nomDuJeu
    def getNomDuJeu(self):
        return self.nomDuJeu

    def setListeJoueurs(self, listeJoueurs):
        self.listeJoueurs = listeJoueurs
    def getListeJoueurs(self):
        return self.listeJoueurs

    def setMiseMin(self, miseMin):
        self.miseMin = miseMin
    def getMiseMin(self):
        return self.miseMin

    def setEtatJoueurs(self):
        for j in self.listeJoueurs:
            if j.argent >= self.miseMin:
                j.setEtatJoueur('run')
            else:
                j.setEtatJoueur('inactif')

    def clearPaquetJoueurs(self):
        for j in self.listeJoueurs:
            j.clearPaquetJoueur



########################################
# PartieBlackJack
########################################
class PartieBlackJack(Partie):
    def __init__(self, listeJoueurs, miseMin, banque, N=1):
```

```
    # commence par supprimer les cartes de paquets des joueurs et mettre a jour l'etat du joueur
    super(PartieBlackJack, self).__init__('BlackJack', listeJoueurs, miseMin)
    self.banque         = banque
    self.paquetCroupier = JeuCartesBlackJack(N).jeuCartesEnsemble
    self.fig            = plt.figure(figsize=(4,3),dpi=120)
    self.run()

def finDePartie(self):
    for j in self.listeJoueurs:
        if j.etat == 'gagne':
            return True
        elif j.etat == 'run':
            return False
        else: continue

def miseAJourArgent(self):
    for j in self.listeJoueurs:
        if j.etat == 'gagne':
            self.banque -= 1.5 * j.mise
            j.argent += 1.5 * j.mise
        elif j.etat == 'perdu':
            self.banque += j.mise
            j.argent -= j.mise
        elif j.etat == 'stop':
            self.banque += 0.5 * j.mise
            j.argent -= 0.5 * j.mise

def plot(self):
    for i,joueur in enumerate(self.listeJoueurs):
        joueur.plot(self.fig,left=0.0,bottom=i*0.25,width=0.2,height=0.2,shift=0.05)
    plt.pause(0.1)

def run(self):
    for j in self.listeJoueurs:
        if j.etat == 'run':
            j.miseJoueur(self.miseMin)
            tCarte = self.paquetCroupier.tirerCarte()
            j.ajouterCarteDansPaquetJoueur(tCarte)
    self.plot()
    while not self.finDePartie():
```

```
            for j in self.listeJoueurs:
                if j.etat == 'run':
                    A = j.action()
                    if A == 's':
                        j.setEtatJoueur('stop')
                    elif A == '1' or A == '2':
                        tCarte1 = self.paquetCroupier.tirerCarte()
                        j.ajouterCarteDansPaquetJoueur(tCarte1)
                        if A == '2':
                            tCarte2 = self.paquetCroupier.tirerCarte()
                            j.ajouterCarteDansPaquetJoueur(tCarte2)
                        val = j.paquetJoueur.getValeurDuPaquet()
                        if val == 21:
                            j.setEtatJoueur('gagne')
                        elif val > 21:
                            j.setEtatJoueur('perdu')
        self.miseAJourArgent()
        sommeArgentJoueur = 0
        for j in self.listeJoueurs:
            sommeArgentJoueur += j.argent
        tmp = 'sommes d argent des joueurs: {}\n'.format(sommeArgentJoueur)
        tmp += 'sommes d argent de la banque : {} \n'.format(self.banque)
        print(tmp)

########################################
# main
########################################
if __name__ == '__main__':
    ########################################
    # Partie 1
    ########################################
    # C = Carte('as','pique',1)
    # print(C)
    # C.plot()

    # C1 = Carte('as','pique',1)
    # C2 = Carte('as','coeur',1)
    # C3 = Carte('Valet','Trefle',10)
    # C4 = Carte('3','Pique',3)
    # P = PaquetCartes()
```

```
# P.ajouterCarteDansPaquet(C1)
# P.ajouterCarteDansPaquet(C2)
# P.ajouterCarteDansPaquet(C3)
# P.ajouterCarteDansPaquet(C4)
# print('Le paquet contient {} cartes et sa valeur est {}'.format(len(P),P.getValeurDuPaquet()))
# print(P)
# carte=P.tirerCarte()
# print('Le {} a été retiré du paquet'.format(carte))
# P.plot()

# jeu=JeuCartesBlackJack(2) # creation d'un jeu de black Jack à partir de 2 jeux de 52 cartes
# jeu.plot() # afficher

########################################
# Partie 2
########################################
# j1=Joueur('Obélix',100)
# j1.ajouterCarteDansPaquetJoueur(Carte('as','pique',1))
# j1.ajouterCarteDansPaquetJoueur(Carte('7','coeur',7))
# j1.setEtatJoueur('run')

# j2=Joueur('Martine',120)
# j2.ajouterCarteDansPaquetJoueur(Carte('as','coeur',1))
# j2.ajouterCarteDansPaquetJoueur(Carte('roi','carreau',10))
# print(j1,j2)

# fig=plt.figure()
# j1.plot(fig,bottom=0.0,width=0.4,height=0.4,shift=0.4)
# j2.plot(fig,bottom=0.5,width=0.4,height=0.4,shift=0.4)

# joueurs=[Joueur('Obélix',80),Joueur('Sarah',200)]
# P=Partie('Black JAck',joueurs,100)
# print(P)

j1=Joueur('Obélix',100)
j2=Joueur('Nathalie',120)
UnePartie = PartieBlackJack([j1,j2],20,1000,1)
UnePartie.run()
```