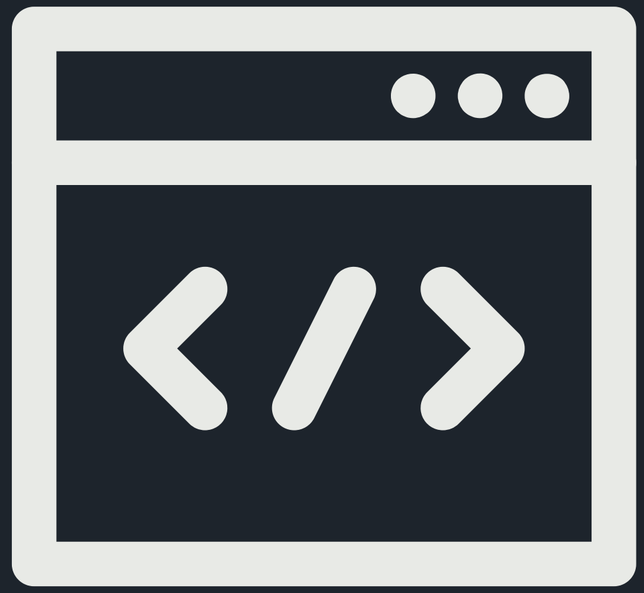
PROJET BLACK JACK

(matter)

(MATTHÉO X 2, VICTOR ET EDGAR)



1er Jour : Travail sur PC (PAPIER- CRAYON)

LES DIFFÉRENTES CLASSES :

CARTES

PAQUET DE CARTE

JOUEUR

LES FONCTIONS PRINCIPALES :

GENERER CARTE()

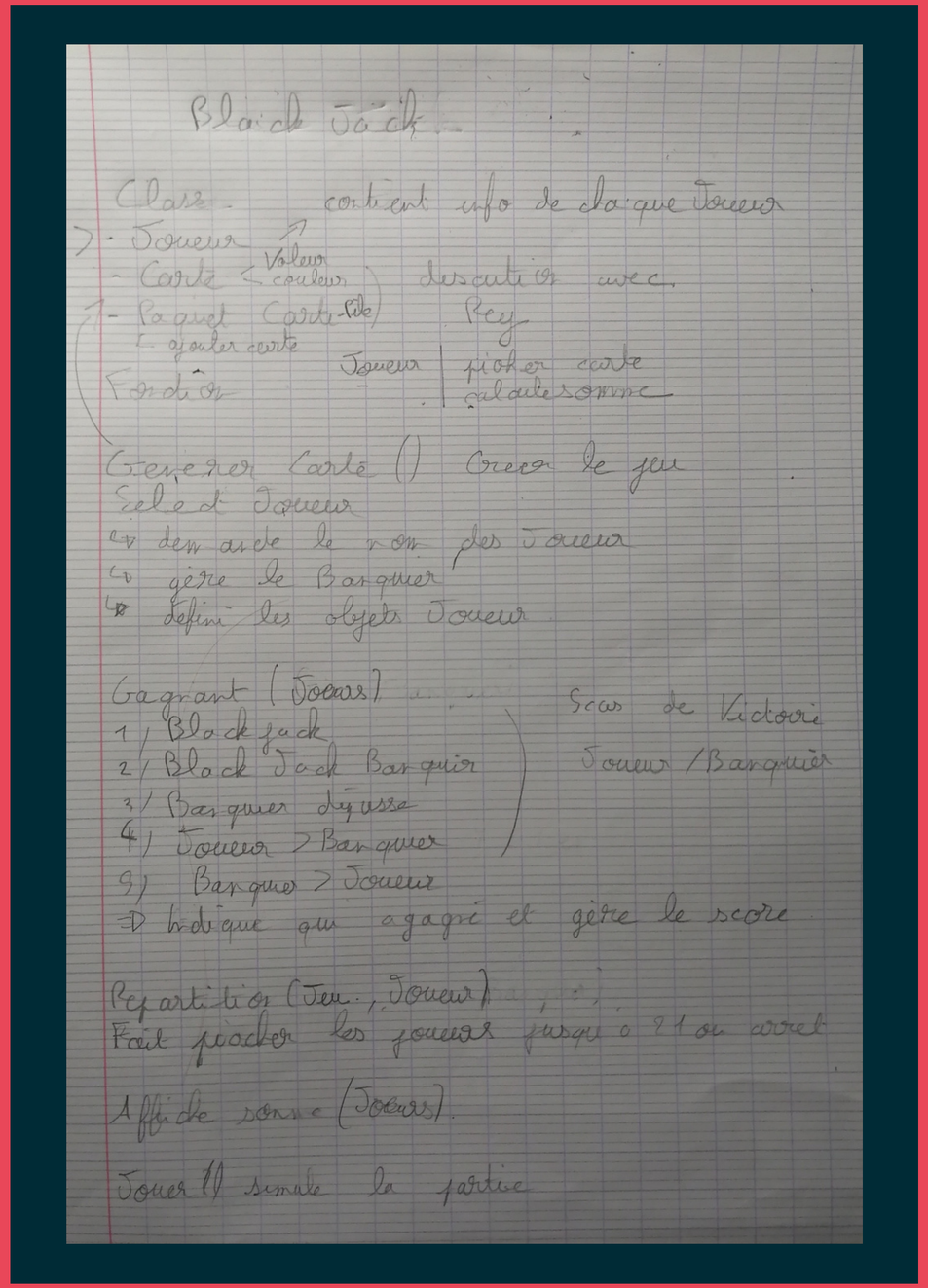
SELECT JOUEUR()

GAGNANT()

REPARTITION()

AFFICHER_SOMME()

JOUER()



```python
from lesPilescorrige import *

class Paquet_Carte :
    '''Creer un object Paquet_Carte qui contient toutes les cartes possible du jeu sous formes de pile (self.tas) soit 4x52cartes

    Valeur:

    tas : Pile de taille 208 qui va contenir les informations de chaques cartes du jeu


    méthode:

    (note : cette class utilise la classe Pile et les méthodes qui vont avec du fichier lesPilescorrige.py)

    - ajouter_carte permet de rajouter une carte de son choix en dernière position de la pile
    - retirer_carte permet de retirer la dernière carte de la pile
    - affichage renvoie les informations de la cartes ( valeur et couleur )

    '''

    def __init__ (self):
        '''initialise self.tas avec une pile définit de taile 208 (nombre total de cartes)'''

        self.tas = Pile(52*4)

    def ajouter_carte(self, carte):
        '''prend un paramètre carte et rajoute la carte choisi dans la liste'''

        self.tas.empile(carte)


    def retirer_carte(self):
        '''Retire la dernière carte de la pile et renvoie ses informations (valeurs et couleurs)'''

        return self.tas.depile()



    def __str__(self):
        ''' Permet l'affichage str des informations du Paquet de carte'''

        self.tas.affichePile()

        return ""
```

class

LA CLASSE PAQUET DE CARTE

Regroupe toutes les
Cartes dans une pile

```python
from Paquet_Carte import *

class Carte :
    '''Creer un object carte avec pour chaques carte définie les informations valeurs et couleurs

    Informations :

    -valeur  : int contient la valeur de la carte (avec l'as qui par default vaut 1, et le valet, la dame et le roi qui valent respectivement 11,12 et 13)
    -couleur : chaine de caractère qui contient le symbole de la carte ( "pique" / "trefle" / "carreau" / "coeur" )

    '''

    def __init__(self,valeur,couleur):
        '''Prend deux paramètres valeur et couleur et initialise l'object carte, avec les valeurs donnés'''

        self.valeur = valeur
        self.couleur = couleur

    def __str__(self):
        ''' Permet l'affichage str des informations de la carte'''

        return str(self.valeur) + " de " + str(self.couleur)
```

class

LA CLASSE CARTE

Permet de redistribuer les valeurs des cartes d'une façon directe et peu coûteuse

LA CLASSE JOUEUR

```python
from Paquet_Carte import *

class Joueur:
    '''Le doc string prennait trop de place et encore il manque du code'''

    def __init__(self, pseudo):
        ''' Prend un paramètre pseudo et initialise toutes les informations du joueur, avec notamment le pseudo indiqué par le joueur'''

        self.nom = pseudo
        self.liste_carte = []
        self.somme = 0
        self.score = 0
        self.arret = False

    def calcul_somme(self):
        ''' Calcul la somme des cartes que le joueur possède et renvoie la nouvelle valeur dans somme'''

        self.somme=0
        si_as = []

        for element in self.liste_carte:

            if element[0] >= 10:
                self.somme += 10
            else:
                if element[0] == 1:
                    si_as.append(element[0])
                else:
                    self.somme += element[0]

        for element in si_as:

            if self.somme <= 10:
                self.somme += 11
            else:
                self.somme += 1


    def pioche(self,monjeu):
        '''Fait piocher une carte au joueur et rajoute les informations de la carte piocher dans liste_carte'''

        ajouter = monjeu.retirer_carte()
        self.liste_carte.append(ajouter)
```

Permet d'attribuer à Chaque entité*
un Pseudo, une liste de carte
et d'autre information utile

```python
        self.liste_carte.append(ajouter)


    def __str__(self):
        ''' Permet l'affichage str des informations du joueur'''

        info_cartes = ""

        for carte in self.liste_carte:

            temp = carte[0]

            if temp == 1:
                temp = "As"

            if temp == 11:
                temp = "Valet"

            if temp == 12:
                temp = "Dame"

            if temp == 13:
                temp = "Roi"

            info_cartes += str(temp) + " de " + str(carte[1]) + "\n"

        a = str(self.nom) + " : \n" + info_cartes + "Somme : " + str(self.somme) + "\n"


        return a
```

class

ENTITÉ POSSIBLE

Tout est gérer par la fonction select_Joueur()

Joueur : Object Joueur Classique

Bot : Object Joueur avec l'attribut bot == True

Banquier : Object Joueur mis dans une variable à part

```python
def select_joueur(nb_joueur,banque_auto):
    '''deso c trop long
    '''


    if banque_auto == True:
        Nb_bot = 4-nb_joueur
    else:
        Nb_bot = 5-nb_joueur


    # Attribut un nom aléatoire pour les bots
    nom_bot = ["Bot(Jean Bombeur)","Bot(Louis Fine)","Bot(Anne Riz)","Bot(Le Père Spective)","Bot(Jone Yalidai)"]
    L_nom_bot = []  # listes des nom des bots

    for i in range(Nb_bot):
        L_nom_bot.append(nom_bot.pop(randint(0,len(nom_bot)-1)))


    # demande le nom pour chaque joueur à l'utilisateur
    L_nom_joueur= [] # listes des noms de chaques joueurs

    for i in range(nb_joueur):
        L_nom_joueur.append(str(input("joueur " + str(i+1) +" :")))

    print("")


    # Si jamais l'utilisateur ne rentre pas de nom, on lui definie un nom par default

    for i in range(len(L_nom_joueur)) :

        if L_nom_joueur[i] == '':
            L_nom_joueur[i] = "Joueur " + str(i+1)


    # si la banque est gérer par un joueur on attribut aléatoirement le role de Banquier

    if banque_auto == False:
        alea= randint(0,len(L_nom_joueur)-1)
        banque = L_nom_joueur.pop(alea)
        print(banque + " est le banquier. \n")

        Banquier= Joueur("Banquier (" + banque + ")")

    # sinon on créer un nouvelle Object Banquier
    else:
        Banquier = Joueur("Banquier")

    L_joueur = []


    # On definie les 4 Joueurs

    for i in range(len(L_nom_bot)):

        L_joueur.append(Joueur(L_nom_bot[i]))

        L_joueur[i].bot = True

    for i in range(len(L_nom_joueur)):

        L_joueur.append(Joueur(L_nom_joueur[i]))

    return [L_joueur,Banquier]  # on renvoie la liste des 4 Joueurs et le Banquier
```

LA FONCTION SELECT JOUEUR

Attribue un nom pour chaques joueurs, séléctionne un banquier de manière aléatoire, et Creer les objects Joueurs

```python
def gagnant(liste_joueur,Banquier):
    """Prends en paramètres:
    Tacos - Galette - Tacos
    Cette Fonction Vous renvoie un tacos saucisse parfaitement préparer
        - On met la saucisse dans la galette Tacos
        - On roule La galette autour de la gross saucisse avec eventuelement un curdent pour que ça tienne mieux
        - On met la Galette 30secondes au micro onde"""

    joueur_en_jeu = []  # liste de tous les joueurs ayant une somme infèrieur à 21
    liste_gagnant = []  # liste des tous les joueurs qui ont gagné la partie
    blackjack=[]  # liste des tous les joueurs ayant une somme égale à 21 en 2 cartes

    mise = 12
    pot = 4*mise


    # Pour tous les joueurs on effectue plusieurs tests et on attribue chaques joueur à une ou plusieurs listes ci dessus

    for J in liste_joueur:

        if J.somme <=21:
            joueur_en_jeu.append(J)

        if J.somme == 21:
            if J.nb_carte == 2:
                blackjack.append(J)

        if Banquier.somme > 21:
            if J.somme <= 21:
                liste_gagnant.append(J)

        if Banquier.somme <= 21:
            if J.somme >= Banquier.somme:
                if J.somme <= 21:
                    liste_gagnant.append(J)


    a = ""

    # Cas 1 : blackjack (avouez vous avez pas lu les commentaires et docstring depuis le début)
    if len(blackjack) >= 1:
        pot = pot/len(blackjack)

        for J in blackjack:

            J.nb_partie_gagner += 1
            J.nb_blackjack += 1
            J.score += pot
            a += str(J.nom) + " gagne " + str(pot) + " points" + "\n"

        print("Black Jack en 2 cartes: " + "\n" + a)

        return
```

LA FONCTION GAGNANT

On a fait 5 cas différents de victoire

Cas 1 : blackjack

Cas 2 : blackjack du Banquier

Cas 3 : Le banquier dépasse

Cas 4 : Joueur > Banquier

Cas 5 : Banquier > Joueur

```
    # Tant que les 4 Joueurs ne se sont pas arreter
    while all_arret <= 4:

        for J in liste_joueur:

            if J.somme < 21 and J.arret == False:

                print(J)

                # Si l'Object Joueur est de type humain
                if J.bot == False:

                    stop = False
                    compteur = 0
                    # tant que la valeur renvoyé est incorect
                    while stop == False:

                        reponse = str(input(J.nom + " : Souhaiter-vous continuer à piocher (1 : oui / 0 : non) :"))

                        # Au bout de 3 valeurs incorect on décide par défault que le Joueur ne repioche pas pour éviter une boucle infini
                        if compteur > 3:
                            stop = True
                            reponse = "non"

                        if reponse =='1' or reponse =='oui' or reponse == 'non' or reponse == '0':
                            stop = True

                        else:
                            print("La valeur que vous avez renvoyé est incorrect \n")
                            compteur += 1

                # si c'est un Object de type Bot on utilise la fonction bot pour avoir une réponse
                else:
                    reponse = bot(J)

                if reponse == '0' or reponse == 'non':
                    J.arret = True
                    print(J.nom + " s'arrête de piocher")
                else:
                    if J.bot == True:
                        print(J.nom + " décide de piocher")

                    J.pioche(monjeu)
                    J.calcul_somme()

                    print("")
                    print(J)

                    if J.somme > 21:
                        print(J.nom + " viens de dépasser")


                print("_____ \n")

            else:
                all_arret +=1
```

LA FONCTION REPARTITION

Fait piocher les joueurs jusqu'à ce que le joueur dépasse 21 ou s'arrête

Les différents problèmes rencontrés

```
484  def reset(liste_joueur,Banquier):
485      '''Prend en paramètres
486      - liste_joueur qui contient une liste de
     tous les Objects Joueurs de type (bot ou
     humain)
487      - Banquier qui contient L'Object Joueur de
     type Banquier
488
489      Cette fonction réinitialise Pour chaque
     Object Joueur les attributs liste_carte, arret
     nb_carte, pour pouvoir relancer une partie'''
490
491      for J in liste_joueur:
492
493          J.liste_carte = []
494          J.arret = False
495          J.nb_carte = 0
496
497      Banquier.liste_carte = []
```

Problème n°1 :
Si on recommence une partie, les joueurs ont toujours leurs cartes.

Solution :
Fonction reset.

Les différents problèmes rencontrés

```
31          self.nb_carte = 0

47      def verif_tas(self):
48          '''Renvoie un boolean indiquant s'il y
    a assez de carte dans le tas pour relancer une
    partie'''
49
50          if self.nb_carte < 55: # 55 : on a au
    maximum 11 cartes par joueurs x nombre de
    joueur (5) donc 55 cartes
51
52              return True
53          return False
```

Problème n°2 :
Après un certain nombre de parties, le paquet de carte est vide.

Solution :
Attribut nb carte qui compte le nombre de carte.
Méthode verif tas qui vérifie qu'il y a au moins 55 cartes.

Bonus : comment on a coder les IA?

```
316  def bot(Obj_joueur):
317      '''Prends un paramètre id_joueur, qui contient un Object
     Joueur de type (humain, bot ou Banquier) et renvoie une chaine
     de caractère selon plusieurs critères 'oui' ou 'non' qui indique
     à la fonction repartition si le bot souhaite repiocher'''
318
319      if Obj_joueur.somme <= 14:
320
321          return 'oui'  # indique qu'on souhaite repiocher
322
323      # Si les as valents 11, c'est qu'on peut repiocher sans
     risquer de dépasser pour espérer avoir mieux
324      if Obj_joueur.as_vaut_11 >= 1 and Obj_joueur.somme <= 18:
325
326          return 'oui'  # indique qu'on souhaite repiocher
327
328      return 'non'  # indique qu'on ne souhaite pas repiocher
329
```

MERCI DE NOUS AVOIR

ECOUTÉ