MINI PROJET LIBRE

Ceci est **un devoir libre**, vous pouvez **si souhaitez que je regarde votre travail ou que je corrige certaines choses, déposer vos fichiers sur le site**

www.prepabellevue.org/PC/Informatique.

Les documents fournis se trouvent dans ce **dossier Projet sur le site**

Pensez à mettre votre nom sur vos fichiers si vous les déposez.

Je vous conseille enfin de travailler en équipe (2/3)

1. IMAGES 3D ANAGLYPHES

Penser à mettre votre nom *NOM_anaglyphe.py* si vous déposez un fichier.

L'objectif est de reconstruire des images 3D sur un écran standard en utilisant des images anaglyphes et les lunettes adaptées.

La photographie ou le cinéma 3D s'appuie sur deux prises de vues quasi-identiques, à une distance de 6.5 cm l'une de l'autre (l'appareil photo est décalé de 6.5cm entre les deux prises, ou deux objectifs permettent d'enregistrer deux photos décalées simultanément). Cette distance correspond approximativement à l'écartement des deux yeux.

Pour que chaque oeil reçoive la bonne photo, donnant l'illusion du 3D, il existe différentes technique : l'émission de deux images polarisées et les lunettes à verres polarisés, l'émission de deux images successive et les lunettes bloquant alternativement les verres gauche et droit, et enfin la technique anagl

Les lunettes anaglyphes sont constituées de deux verres filtrant des couleurs différentes pour l'oeil gauche et l'oeil droit. La photo anaglyphe est la superposition des deux photos, où seules les couleurs transmises par les verres des lunettes sont présentes.

Une série d'images vous est fournie (dans le **dossier anaglyphe sur le site**):

- *bleu.jpg : couleur observée à travers la lentille bleu en observant du blanc,*
- *rouge.jpg : couleur observée à travers la lentille rouge en observant du blanc,*
- *blanc.jpg : couleur blanche observée pour les deux tests précédents,*
- *chateau_gauche.jpg : photographie d'un château en ruine prise à la position de l'oeil gauche,*
- *chateau_droit.jpg : photographie d'un château en ruine prise à la position de l'oeil droit.*

Copier/coller le dossier anaglyphe sur votre ordinateur puis le DEZIPPER

Les images rouge.jpg, bleu.jpg et blanc.jpg ont été obtenue en plaçant les lunettes devant un papier blanc éclairé par l'arrière :

Ces images bleu.jpg et rouge.jpg permettront de déterminer dans un premier temps quelles couleurs chaque lentille transmet. Dans un second temps, l'image anaglyphe sera construite à partir des deux images gauche et droite.

1.1. Travailler sur une image

Ouvrez une console python (ou ipython) et placez-vous dans le répertoire du projet

```
from os import chdir
chdir ("C :\\votre chemin »)
Vérifiez que la commande « ls » renvoie bien les noms des images
```

Importer la bibliothèque imageio pour pouvoir charger et enregistrer les images :

```
import imageio as scm
#ou
import matplotlib.image as mpimg
```

Importer l'image bleu.jpg et affichez là pour vérifier le bon déroulement du chargement :

```
im_b=scm.imread("bleu.jpg")
imshow(im_b)
```

Rappels : En appliquant la commande « print » à l'image bleu (print im_b), vous verrez que l'image est stockée sous forme d'une liste de dimension 3 composée d'entiers entre 0 et 255. Les deux premières dimensions sont les coordonnées x et y des pixels, et la troisième dimension contient les niveaux des trois couleurs RGB (red, green, blue), codées sur un octet (soit un entier entre 0 et 255 pour chaque couleur, ce qui permet d'obtenir plus de 16 millions de couleurs différentes).

Ainsi, pour afficher les 3 couleurs RGB du pixel de coordonnées (20,40), taper :
```
im_b[20,40]
```

La taille (en pixels) de l'image peut être obtenue par les commandes im_b.shape[0] et im_b.shape[1] .

Q1. Vérifier que chaque couleur est codée sur un octet. Vérifier que sur le pixel (20,40) la lentille ne laisse pas passer certaines couleurs.

1.2. Analyse du filtrage réalisé par les lentilles

Q2. Proposer un algorithme permettant d'obtenir la moyenne des 3 couleurs à travers le filtre de la lentille.

Q3. Faire la même moyenne sur les couleurs blanche et rouge pour vérifier que le blanc présente bien toutes les couleurs et identifier les couleurs filtrées par la lentille rouge.

***Q4.** Déterminer quelles couleurs il faut sélectionner pour l'image réservée à l'oeil gauche et celles pour l'image réservée à l'oeil droit.*

1.3. Création d'images anaglyphes

L'image anaglyphe est une composition des deux images gauche et droite. Les niveaux des couleurs transmises par la lentille de gauche sont pris dans l'image gauche, et de même les niveaux des couleurs transmises par la lentille de droite sont pris dans l'image droite. Pour copier une image im1 dans une nouvelle variable, utiliser : im_anag=im1.copy().

Q5. Proposer un algorithme permettant de créer l'image anaglyphe « im_anag » puis vérifier le rendu 3D à l'aide des lunettes.

Q6. Mettre l'algorithme sous la forme d'une fonction ayant pour argument les deux images gauche et droite et renvoyant une image anaglyphe.

Pour enregistrer l'image obtenue, utiliser la commande : scm.imsave("anaglyphe.jpg",im_anag).

Les images du château en ruine ont été prises par Marcel Cador à l'aide d'un appareil disposant de deux objectifs distants de 6.5 cm (écartement des yeux). L'utilisation de ces images est autorisée dans le cadre des renseignements d'informatique mais tout autre utilisation est interdite sans avis. D'autres images anaglyphes sur http://www.mcbat.net

2. JEU DE POKER : TEXAS HOLD'EM

On se propose de modéliser ici un jeu de Poker. Tout d'abord, celui-ci contient 52 cartes, réparties en 4 couleurs (pique, cœur, carreau, trèfle), chacune contenant les cartes numérotées de 1 à 10 ainsi que le valet, la dame et le roi.

Les couleurs seront codées par les entiers de 0 à 3, et les valeurs par les entiers de 0 à 12.

Pour l'affichage des cartes on pourra utiliser les listes suivantes :

```
Couleur=[u'\u2661',u'\u2662','♣', '♠']
Valeur=[str(k) for k in range(2,11)]+["J","Q","K","A"]
```

Pour le hasard on importe la fonction randint du module random :

```
from random import randint
n=randint(1,13) # choisit un nombre entier aléatoirement entre 1 et 13
```

Nous allons modéliser de manière simplifiée le Texas Hold'em, version la plus répandue actuellement. Pour ce faire, nous allons utiliser des listes, des boucles et des fonctions simples mais il faut être cosncient qu'un tel, jeu dans une version plus complète serait faite à l'aide de la programmation orientée objet (class) d'autant plus si nous voulions faire une version multi joueur en ligne … Si vous êtes intéressés vous trouverez des modules Python qui permettent de faire des jeu de poker beaucoup plus évolués avec interfaces graphiques (`PyPokerEngine`, `PyPokerGUI`)

Voici les règles générales du Texas Hold'em :

- *Deux cartes, ne pouvant être vues que par le joueur qui les reçoit, sont distribuées à chaque joueur*
- *Le croupier révèle cinq cartes - trois d'un coup, puis une quatrième, puis une cinquième - qui peuvent être utilisées par tous les joueurs pour former la meilleure main de cinq cartes possible*
- *Les joueurs misent chacun leur tour avant et après la révélation de chaque carte. Pour rester dans la main et voir la prochaine carte, tous les joueurs doivent avoir placé le même montant de jetons dans le pot*
- *La meilleure main de poker remporte le pot*

Penser à mettre votre nom ***NOM_poker.py*** si vous déposez un fichier.

2.1. Fonctions préliminaires

Avant de coder le jeu du Texas Hold'em, nous allons coder quelques fonctions simples qui serviront pour le jeu.

Q7. La carte de couleur `i` et de valeur `j` sera modélisée par le couple `(i; j)`. Écrire un programme définissant le jeu de cartes : `Jeu=[[0,0],[0,1],...]`.

Q8. Écrire une fonction renvoyant une main de 2 cartes distinctes, sous forme de liste de 2 couples d'entiers. Ecrire ensuite une fonction `AffichageMain(M)` qui permet d'afficher les 2 cartes.

Q9. Écrire une fonction renvoyant une liste de `n` mains de 2 cartes distinctes, toutes prélevées dans le même jeu. Le nombre de mains `n` est donc supposé inférieur ou égal à 10.

Je vous propose ici d'utiliser des `input` et `affichageMain` suivi de nombreux `print` pour faire un affichage « joueur par joueur »

```
input("Main Joueur "+str(m+1))
AffichageMain(M)
input()
print()
print()
print()
print()
print()
print()
print()
print()
print()
print()
```

Pour améliorer l'affichage, je vous donne une PISTE pour améliorer l'affichage des cartes, TOUJOURS très SIMPLIFIE. Ce n'est pas utile de l'utiliser sauf si vous avez fini de coder le jeu complet ! (Au passage vous remarquerez que les caractères spéciaux ne s'écrivent pas avec pyplot.text ☹… donc n'hésitez pas à peaufiner si vous avez du temps à perdre ces vacances ☺

```
import matplotlib.pyplot as plt

def AfficheCarte(position,val,coul):
    [x,y]=position
    plt.plot([x,x+3,x+3,x,x],[y,y,y+5,y+5,y],color='k', linewidth=3)
    if coul=='♡':
        plt.text(x+.1,y+.1,val,color='r', size=16)
        plt.text(x+.1,y+4.5,val,color='r', size=16)
        plt.text(x+2.7,y+.1,val,color='r', size=16)
        plt.text(x+2.7,y+4.5,val,color='r', size=16)
        plt.text(x+.7,y+2.3,'coeur',color='r', size=24)
    elif coul=='♢':
        plt.text(x+.1,y+.1,val,color='r', size=16)
        plt.text(x+.1,y+4.5,val,color='r', size=16)
        plt.text(x+2.7,y+.1,val,color='r', size=16)
        plt.text(x+2.7,y+4.5,val,color='r', size=16)
        plt.text(x+.5,y+2.3,'carreau',color='r', size=24)
    elif coul=='♣':
        plt.text(x+.1,y+.1,val,color='k', size=16)
        plt.text(x+.1,y+4.5,val,color='k', size=16)
        plt.text(x+2.7,y+.1,val,color='k', size=16)
        plt.text(x+2.7,y+4.5,val,color='k', size=16)
        plt.text(x+.6,y+2.3,'trèfle',color='k', size=24)
    elif coul=='♠':
        plt.text(x+.1,y+.1,val,color='k', size=16)
        plt.text(x+.1,y+4.5,val,color='k', size=16)
        plt.text(x+2.7,y+.1,val,color='k', size=16)
        plt.text(x+2.7,y+4.5,val,color='k', size=16)
        plt.text(x+.7,y+2.3,'pique',color='k', size=24)

plt.figure()
plt.axis([-5,6,-6,3])
AfficheCarte([-3,-5],'2','♡')
AfficheCarte([1,-5],'3','♢')

plt.figure()
plt.axis([-5,6,-6,3])
AfficheCarte([-3,-5],'9','♣')
AfficheCarte([1,-5],'V','♠')
```

On va s'intéresser maintenant à une main complète, afin d'évaluer la force d'une main et ainsi de comparer les différentes mains possibles à la fin du jeu et de determiner le gagnant.

Q10. Écrire tout d'abord une fonction `Tableau(M)` recevant une main de 5 cartes et renvoyant deux listes : l'une de longueur 4 contenant le nombre de cartes de chaque couleur présentes dans la main, et l'autre de longueur 13 contenant le nombre de cartes de chaque valeur contenues dans la main.

Les combinaisons gagnantes d'une main sont, par ordre de force décroissante, la suite royale, le carré, le full, la couleur, la suite, le brelan, la double paire, la paire.

Les couleurs (pique, coeur, carreau, trèfle) ne sont jamais utilisées pour évaluer la force d'une main au holdem mais pour généraliser on les met quand même !

La force d'une main est définie comme valant 8 en cas de suite royale, etc jusqu'à 1 pour la paire et 0 sinon. Soit :

- *une suite royale (cinq cartes de même couleur et de valeurs consécutives, de couleurs quelconques) → 8*
- *un carré (quatre cartes de même valeur) → 7*
- *un full (un brelan et une paire) → 6*
- *une couleur (cinq cartes de même couleur) → 5*
- *une suite (cinq cartes de valeurs consécutives, de couleurs quelconques) → 4*
- *un brelan (trois cartes de même valeur) → 3*
- *une double paire (2 * 2 cartes de même valeurs) → 2*
- *une paire (2 cartes de même valeurs) → 1*
- *rien (on appelle cela une hauteur) → 0*

Q11. Écrire une fonction `Force(M)` qui renvoie la force d'une main ainsi que la valeur associée. Pour les suites, les couleurs et la hauteur , on renverra la plus grande valeur associée.

Je vous conseille de créer 2 petites fonctions bien utiles `NbOc(a,L)` qui compte le nombre d'occurrences de `a` dans `L` et `dernier(nb,L)` qui renvoie l'indice de la dernière position d'une valeur `nb` dans la liste `L`. On pourra aussi utiliser la fonction python `max(L)`.

Si on teste par exemple :

```
Combinaisons=["Hauteur", "Paire", "Double paire", "Brelan", "Suite", "Couleur","Full"
, "Carré", "Suite royale"]

M=[[1,11],[2,11],[3,11],[0,11],[1,10]]
f=Force(M)
AffichageMain(M)
print("Combinaison :",Combinaisons[f[0]], Valeur[f[1]])
#Combinaison : Carré K

M=[[0,1],[0,3],[0,8],[0,9],[0,2]]
#Combinaison : Couleur J

M=[[0,1],[1,1],[0,8],[2,8],[3,8]]
#Combinaison : Full 10

M=[[0,1],[1,1],[0,8],[2,8],[3,7]]
#Combinaison : Double paire 10

M=[[0,1],[1,10],[0,12],[2,8],[3,8]]
#Combinaison : Paire 10

M=[[3,6],[3,4],[3,5],[3,7],[3,8]]
#Combinaison : Suite royale 10

M=[[0,6],[3,11],[2,5],[3,7],[3,8]]
#Combinaison : Hauteur K
```

2.2. Texas Hold'em

Attention, nous allons faire ici une version simplifiée du jeu, sans *joueur computer*. Pour faire jouer l'ordinateur, il faudrait introduire des calculs de statistiques, en fonction de sa probité de gagner (dépend de sa main avec 2 cartes et les 5 communautaires) l'ordinateur mise et ceci est très complexe à programmer. Bref, on fait SIMPLE !

Initialisation

Q12. Commencer par demander le nb de joueur (`input`)

Q13. Créer une fonction `initialisation(n)`, `n` étant le nombre de joueurs, qui renvoie 3 listes :

`argent` la liste de la somme de départ (1000 par exemple) pour chaque joueur,
`mises` liste de mises (à 0) pour chaque joueur,
`listeJoueur` afin d'afficher les joueurs (`J 1, J 2, J 3 ...`) en association avec les listes `mises` et `argent`.

Jeu

A partir d'ici il est difficile de poser des questions précises pour structurer le jeu. Je vais plutôt vous décrire la structure du Texs Holdem avec les étapes importantes et chacun fera le jeu correspondant à son interprétation.

Je vous conseille cependant de mettre toute la suite du programme dans un `while jeu` afin de demander en fin de partie si on continue ou non …

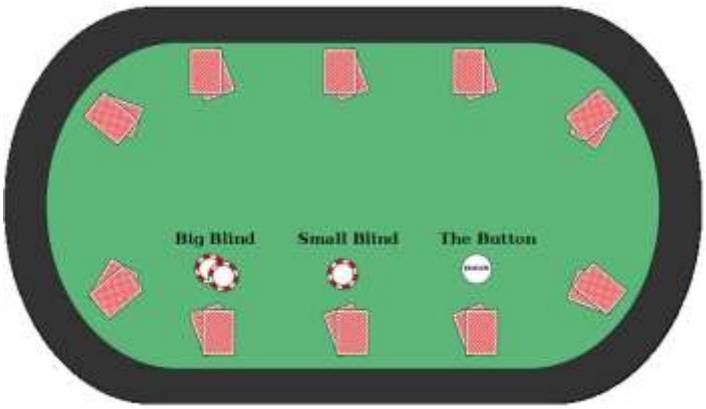
Le Texas Holdem se joue sur une table entre 2 et 10 joueurs. Le but est simple : remporter le plus d'argent possible, un pot à la fois !
Vous remportez un pot en ayant la meilleure main, ou en faisant coucher tous les autres joueurs avant l'abattage.

La structure du Texas Hold'em peut être divisée en trois parties :

- *La mise en place et le lancement*
- *Les tours de mises*
- *L'abattage*

La première étape est de désigner le joueur qui commencera avec le « bouton du donneur ». Le Holdem se joue avec ce que l'on appelle un « **dealer rotatif** », ce qui veut dire qu'un joueur agira comme donneur pour une main, puis ce sera au tour du joueur suivant une fois cette main terminée, et ainsi de suite.

Ici le bouton ne donnera pas physiquement les cartes, elle restera vue comme le donneur officiel pour la main en cours, avec les blindes et le déroulement du jeu s'orientant ainsi par rapport à lui.



Mettre les blindes

Maintenant que vous avez un donneur, vous devez donc poser les blindes.

Il y a deux « blinds » en hold em, une petite et une grosse.

Le joueur directement après le dealer pose la petite blinde.

Le joueur directement après celui qui a posé la petite blinde, pose la grosse blinde (le double de la petite), et on verra que le premier joueur à parler est celui directement après la grosse blinde.

➔ *Vous devez calculer la petite blind et grosse blind à partir du nombre de parties* `modulo` *nbj*

Déroulement du jeu et Tours de mises au Holdem

➔ *Commencer par distribuer 2 cartes aléatoires à chaque joueur ;*

Si vous voulez vraiment jouer à plusieurs, l'idée est de noté ses cartes à l'abris des regards des autres puis de laisser l'autre joueur demander sa main, et ainsi de suite et à la fin de la distribution, on peut tous regarder l'écran …
J'imagine même pour l'utilisation d'une figure avec le tapis central, un deuxième écran pour avoir d'un côté les commandes dans le shell et de l'autre le tapis de jeu … mais bon ça c'est vraiment si vous avez rien à faire ces vacances !

Le Préflop

Lorsque tous les joueurs ont reçu leurs cartes, on entre dans la phase de mises pré-flop.

Le tour de mises préflop démarre par le joueur suivant la grosse blinde. Celui-ci a trois options :

- ***Se coucher (Fold) :*** *Le joueur ne paie rien. Il attendra la main suivante pour pouvoir rejouer.*
- ***Suivre (Call, Payer) :*** *Le joueur égalise le montant de la grosse blinde pour rentrer dans le coup.*
- ***Relancer (Raise) :*** *Le joueur relance d'un montant au moins égal* ***à deux fois*** *la grosse blinde.*
- ***Check :*** *un joueur qui a déjà mis la mise la plus haute n'a pas besoin de rajouter plus d'argent pour rester dans le coup*

➔ *La mise sera enregistrée dans la liste* `mises`*. Quand un joueur se couche, sa mise sera ajoutée au pot et on mettra un -1 à son indice dans la liste des mises.*
➔ *Il faut aussi vérifier qu'un joueur a suffisamment d'argent pour miser !*

La relance est toujours le montant d'une mise en plus du montant de la mise précédente. Par exemple si la grosse blinde est de 20, et que le premier joueur à parler veut relancer, il met au minimum un total de 60 (la grosse blinde + une mise additionnelle). Si le joueur d'après veut sur-relancer, il mettra au minimum un total de 100 (la mise d'avant + une mise de plus égale à 2 fois la grosse blinde).

➔ *Pour simplifier ici la relance, nous imposerons donc le* `max des mises + 2* la blind` *(= 20 ici)*

➔ *Si vous voulez quand même pimenter un peu les choses, vous pourrez proposer une dernière option* ***Tapis (All in)*** *où le joueur met tout son argent dans le pot. Mais attention car ici la deuxième condition explicité ci-dessous peut alors ne pas être remplie … il faudra donc prendre en compte ce cas de figure.* ***Dans un premier temps, je vous conseille de ne pas mettre Tapis et vous l'ajouterez si votre programme tourne sans.***

Un tour de mises se termine lorsque deux conditions sont remplies :

- *Tous les joueurs ont eu l'opportunité d'agir.*
- *Tous les joueurs qui n'ont pas jeté leurs cartes ont misé le même montant d'argent dans le pot.*

Un premier exemple de tour de mises :

```
Il y a 5 joueurs à la table :
Joueur 1 - Bouton
Joueur 2 - Petite blinde (10)
Joueur 3 - Grosse blinde (20)
Début du tour de mises
Joueur 4 - Suit la grosse blinde (20)
Joueur 5 - Se couche
Joueur 1 - Suit la grosse blinde (20)
Joueur 2 - Suit la grosse blinde (étant donné qu'il a déjà mis 10 dans le pot, il n'a qu'à rajouter 10, pour un total de 20)
Joueur 3 - Checke (étant donné qu'il a déjà mis ses 20, il n'a pas besoin de rajouter plus d'argent pour rester dans le coup)
Fin du tour de mises
Lorsque le joueur 2 suit la grosse blinde, tous les joueurs ont maintenant le même montant d'argent en face d'eux, mais le joueur 3 (la grosse blinde) n'a pas encore eu la chance de parler, le tour de mises n'est donc pas terminé.
Une fois que le joueur 3 a checké, toutes les conditions sont réunies, et ce tour de mises est donc bel et bien fini.
```

Deuxième exemple de tour de mises :

```
Il y a 5 joueurs à la table :
Joueur 1 - Bouton
Joueur 2 - Petite blinde (10)
Joueur 3 - Grosse blinde (20)
Début du tour de mises
Joueur 4 - Suit la grosse blinde (20)
Joueur 5 - Relance (60)
Joueur 1 - Se couche
Joueur 2 - Se couche
Joueur 3 - Sur-relance (il a déjà mis 20 en tant que grosse blinde - il complète jusqu'à 60, et ajoute une mise pour un total de 100)
Joueur 4 - Se couche (son précédent "suivi" de 20 est maintenant dans le pot)
Joueur 5 - Suit (égalise la mise du joueur 3 pour un total de 100)
Fin du tour de mises
Dans ce scénario, tous les joueurs ont eu la chance d'agir lorsque le joueur 3 fait une sur-relance. Mais tous les joueurs n'ont pas misé la même somme dans le coup.
Une fois que le joueur 4 s'est couché, il ne reste que les joueurs 3 et 5 dans le pot. Lorsque le joueur 5 paye, toutes les conditions sont remplies, et le tour de mises est donc terminé.
```

➔ *En conclusion,* `Tant que` *tous les joueurs n'ont pas parlé OU que les valeurs maximales de mises sont différentes, on continue de demander les actions aux joueurs qui ne sont pas couchées. Toutes les mises seront prises en compte sur la liste* `argent` *(à l'indice correspondant au joueur) et sur la liste* `mise` *(à l'indice correspondant au joueur). Quand un joueur se couche, on notera -1 dans la liste* `mise` *à l'indice correspondant et on augmentera le* `pot`*.*

Je vous conseille alors de faire une petite fonction `testMax(L,argent)` *qui teste si les valeurs maximales de L sont différentes, on renverra* `False` *si tous les max ne sont pas égaux et* `True` *si toutes les valeurs autres que -1 sont égales. Je mets également* `argent` *en paramètre car vous verrez, au fil du jeu que lorsqu'un joueur n'a plus d'argent, il ne faut pas le prendre en compte dans ce teste.*

#par exemple

```
L=[-1,10,20,-1,20,20,-1]
print(testMax(L,argent)) # False
L=[-1,20,20,-1,20,20,-1]
print(testMax(L,argent))  #True
```

Le Flop

Une fois que le tour de mises préflop s'est terminé, *le flop* est distribué.

➔ *Distribuer 3 cartes et les afficher (« au centre de la table »)*

C'est à ce moment que le premier tour de mises post-flop démarre.

Les règles d'un tour de mises d'après le flop sont les mêmes que pré-flop, à deux petites exceptions :

- *le premier joueur à agir est le premier joueur encore en jeu (ceux qui ne sont pas couchés) à la gauche du donneur, vous devez utiliser les positions des blindes et boutons*
- *le premier joueur à parler peut soit **checker,** soit **miser**. Etant donné qu'aucune mise n'a encore été faite, "suivre" est gratuit.*

Une mise au flop est du montant de la grosse blinde. Dans notre exemple, le joueur doit au moins poser 20 pour effectuer une mise.

La Turn

Une fois que le tour de mises du flop est terminé, c'est le *turn*.

➔ *on distribue une nouvelle carte à la suite du flop.*
➔ *Le troisième tour de mises peut alors démarrer, et dans notre programme nous garderons un tour de mises identique à celui du flop (check ou relance du double de la grosse binde)*

La River

En admettant qu'il reste plus d'un joueur toujours en jeu, la *river* est distribuée. Distribuer la river est la même chose que distribuer la *turn*, avec une carte à la suite des autres.

Plus aucune autre carte ne sera distribuée pour cette main.

➔ *Le tour de mises y est identique à celui de la turn.*

L'Abattage

Une fois que le tour de mises de la river est terminé, les joueurs entrent dans la phase du *showdown (abattage*). A ce stade, la **meilleure main remporte le pot.**

➔ *On calcule le pot en ajoutant tous les termes de la liste* `mises` *au* `pot` *précédent, simple !*
➔ *Cette somme est ajoutée à la liste* `argent` *pour l'indice du gagnant, basique !*

Q14. Pour trouver le gagnant vous aurez besoin de créer au préalable une fonction `meilleureAssoc(joueur,centre)` *qui fait toutes les associations des 2 cartes d'un* `joueur` *et de 3 cartes du* `centre` *(communautaire) parmi les 5 et qui renvoie la main qui a la plus grande force(d'où la fonction* `Force` *!)*

Exemple :

```
centre=[[0,6],[3,11],[2,5],[3,7],[3,8]]
Mjoueur=[[2,6],[2,11]]
AffichageMain(Mjoueur)
AffichageMain(centre)
AffichageMain(meilleureAssoc(Mjoueur,centre))
# ♣ 8 ; ♣ K ; ♡ 8 ; ♠ K ; ♣ 7
```

Q15. Ensuite, pour tous les joueurs qui ne sont pas couchés, on applique la fonction `Force(meilleureAssoc(MainJoueur,centre))` *afin de créer la liste des meilleures mains de tous les joueurs en jeu PUIS on compare les valeurs entre elles pour trouver le gagnant de la main. Rappelez-vous que* `Force` *renvoie 2 valeurs qu'il faut comparer de manière hiérarchique.*

Q16. S'il y a égalité pour les valeurs de `Force`*, on divise le pot entre les joueurs…*

Exemple :

```
Joueurs [1, 2, 3, 4, 5] ; mises =[-1, 100, 100, -1, 100]
→ listMMains=[-1,[4,11],[6,5],-1,[3,12]]
→ le Joueur 3 remporte le pot de 330. (les joueurs 1 et 4 se sont couches après avoir mis 10 et 20.
```

Et on recommence … *Pensez à réinitialiser les mises !*

Le bouton de dealer devient alors le joueur suivant, avec les deux joueurs d'après qui poseront à leur tour petite et grosse blindes, respectivement.

Normalement à ce stade vous devriez avoir un jeu qui tourne, il est bien évident qu'une multitude d'améliorations peuvent être apportées …

Par exemple, pour proposer le tapis il faut modifier la fonction `testMax(mises,argent,tapis)` en utilisant une liste `tapis` correspondant aux joueurs

Quand un joueur n'a plus d'argent, je continue à lui distribuer des cartes ! Il faudrait alors modifier la fonction `distribution` ainsi que la manière dont on calcule le gagnant.

Tout ça montre l'intérêt de l'objet car ici on comprend qu'avec des objets dont la taille change en fonction du nombre de joueur en jeu, on pourrait beaucoup plus facilement construire les objets initiaux (*constructeurs self_init*) avec *leurs attributs* et *leurs méthodes* (fonctions) …

Il serait aussi très intéressant de faire une version en ligne où le centre communautaire est vu de chaque ordi et seul le joueur voit son jeu …. Mais ça c'est pour l'école d'ingénieur … ☺

Sur ce, Bonnes vacances !