Tale NSI Projet « jeu du pharaon »

Règles du jeu

Le pharaon est un jeu de cartes qui se joue avec 7 cartes par joueur. **Le but du jeu est d'intégrer ses cartes dans des figures**, de manière à ce qu'il reste au plus une carte inoccupée, et de valeur au plus 5. Dans ce jeu, on cherche à éviter les « points » qui sont des points de pénalité.

Les figures possibles sont : des **brelans** (3 cartes de même niveau), des **carrés** (4 cartes de même niveau), des **suites** (3 cartes successives de la même famille)

A chaque tour, le joueur a le choix entre piocher au talon ou ramasser la dernière carte de la défausse. Il intègre la carte choisie dans ce jeu, puis doit ensuite rejeter sur la défausse une carte de son choix.

L'As vaut 1 point, le 2 vaut 2 points, et ainsi de suite jusqu'au 10 qui vaut 10 points. Les honneurs (Valet, Dame, Roi) valent chacun 10 points. **Quand des cartes sont intégrées à une figure, leur valeur n'est pas comptée.**

Quand le joueur a réussi à constituer deux figures, et qu'il lui reste une seule carte, par exemple de valeur 3, il annonce « Pharaon 3 », et le jeu s'arrête. Son ou ses adversaire(s) font alors le décompte des points de leur propre jeu *(sauf les cartes qui sont déjà intégrées à des figures).*

BONUS : on convient que l'As de Cœur peut remplacer n'importe quelle carte

Exemple :

Voici une fin de partie possible :
Joueur A : **Roi Pique / Roi Trefle / Roi Cœur, 5 – 6 – 7 de carreau, 4 de trefle**. Le joueur a « pharaon 4 » (un brelan, une suite, et il reste un 4) Il reporte 4 points sur sa feuille de scores.

Joueur B : quand son adversaire a déclaré pharaon, il lui reste en main : **Dame Pique / Dame Trefle / Dame Carreau, 7 Cœur / 7 pique / 6 pique / 9 trefle**
Le joueur B totalise 29 points (on ne compte pas les dames qui sont intégrées dans un brelan)

Si le joueur A avait eu le quatrième Roi au lieu du 4 de trèfle, il aurait eu « pharaon 0 ». C'est le meilleur score possible.

Stratégie

A chaque tour, le joueur a deux leviers de stratégie

- **Piocher / ramasser** : le joueur évalue le potentiel de la carte défaussée par son adversaire juste avant lui. Si elle lui semble intéressante (elle va permettre de faire une figure), il la ramasser, sinon il choisira plutôt de piocher au talon.

Exemple : si le joueur a déjà une carte de même niveau que celle de la défausse, il a probablement intérêt à la prendre, puisque ça lui fait une paire, donc un premier pas vers un brelan. Idem si elle suit immédiatement une de ses cartes. Et bien sûr, si elle lui permet de compléter un brelan ou un carré ou une suite, il n'y a pas à hésiter. Sinon, on pioche.

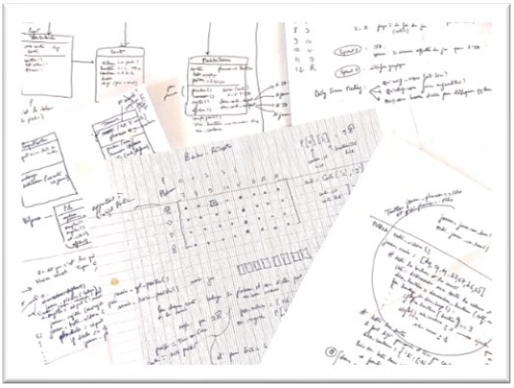
- **Rejeter** : une fois qu'il a pioché ou ramassé, le joueur doit à son tour défausser une carte sur la pile. Le choix de cette défausse est déterminant : il doit évaluer dans son jeu la carte la moins intéressante (celle qui est le moins susceptible d'entrer dans une figure)

Exemple : pour le joueur B dans le cas décrit plus haut, la carte la moins intéressante est le 9 de trèfle : en effet, la paire de 7 peut évoluer vers un brelan, et l'ensemble 6-7 de pique vers une suite.

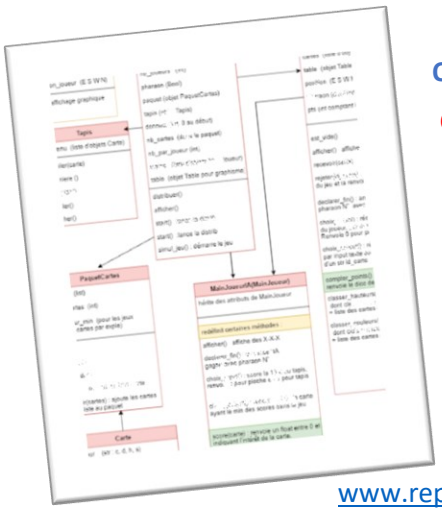
L'objectif de ce projet est de programmer le pharaon : un joueur contre l'ordinateur (une « IA »)

Cadre de travail, organisation

Pour votre projet, vous devrez **utiliser en permanence 5 outils** :



OUTIL 1 : du papier et un crayon pour réfléchir au brouillon, esquisser les classes, dérouler rapidement le fonctionnement d'un bout de programme…



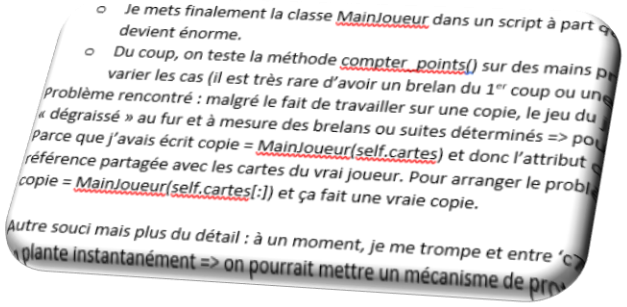
OUTIL 2 : un diagramme des classes décrivant l'architecture de votre application. Vous pouvez utiliser l'outil diagrams.net *(accessible depuis un Google Drive par exemple)*

OUTIL 3 : un éditeur de code comme Pyzo. En parallèle, vous avez aussi la possibilité d'utiliser une plate-forme collaborative en ligne, comme le site www.replit.com *(accessible avec un compte Google, ou alors possibilité de création d'un compte)*. Chaque classe devra être dans un script séparé, et testée individuellement dans ce script avec un « if __name__ == '__main__'. Les classes devront être bien documentées dans le code

```
class MainJoueur:
    '''
    attributs :
    @cartes : une liste (éventuellemer
    @IA : Booléen indiquant si le joue
    @table : un objet de classe Table
    @position : str indiquant la posit
E, W)
    @pharaon : un Booléen indiquant si
pas

    méthodes :
    @afficher
    @recevoir(carte)
    @rejeter(carte)
    @classer_hauteurs() qui renvoie ur
```

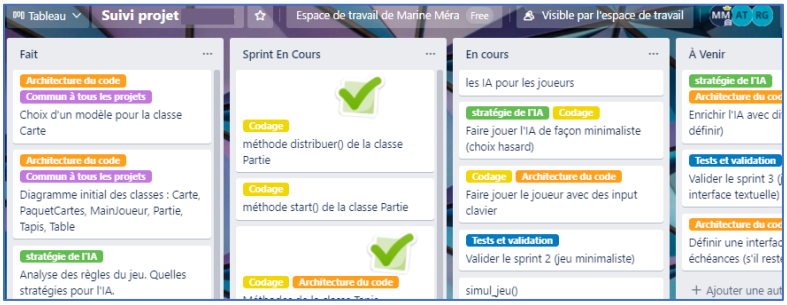
```
#test
if __name__ == '__main__':

    carte1 = Carte('s', '2')
    carte2 = Carte('s', '3')
    carte3 = Carte('s', '4')
    carte4 = Carte('h', '7')
    carte5 = Carte('s', '7')
    carte6 = Carte('d', '7')
    carte7 = Carte('c', 'K')
    l = [carte1, carte2, carte3, ca
                 carte5, carte6, car

    jeu_test = MainJoueur([carte1,
                 carte5, carte6, car
    print(jeu_test.compter_points(
```

OUTIL 4 : un carnet de bord *(type document Word ou pad ou GoogleDoc pour la version collaborative)* où vous « tracerez » au fur et à mesure l'historique de votre projet : les idées, les problèmes rencontrés, les solutions trouvées, les modifs faites à la structure du code…



Ce document vous sera précieux pour rédiger le rapport final !

OUTIL 5 : un tableau de suivi de projet (par exemple Trello), qui vous permettra de vous répartir le travail, visualiser l'avancement des tâches (en cours, à venir, déjà effectuées) au moyen de « post-its ».



*Il est normal qu'il y ait des **aller-retours permanents** notamment entre le diagramme des classes et le code : AVANT de coder, il faut avoir un plan de structure en tête, et APRES avoir codé : on se rend compte d'un tas de choses qui conduisent à modifier cette structure initiale. Puis on re-code, on re-teste, on re-modifie… et au fur et à mesure des aller-retours : le projet aboutit à sa forme finale.*

Tableau synoptique des sprints, planning et barème

Prévu sur 5 semaines

Pendant le projet :

- Les heures de classe permettront à l'équipe de réfléchir ensemble, se répartir les tâches, valider ce qui a déjà été fait, débloquer avec l'enseignant les points difficiles.

- **Prévoir également un temps de travail en-dehors des cours** (deux à trois heures par semaine semblent réalistes) pour que chacun avance de son côté sur les tâches qui lui sont échues.

Livrables	**Jalon 1**	**Jalon 2**	**Jalon 2bis**	**Jalon 3**	**Jalon 4**	**Jalon 5**
Echéance	2h	2h	2h	4h	2h	2h
Barème	2 pts	4 pts	4 pts	4 pts	4 pts	2 pts

Description des sprints et livrables

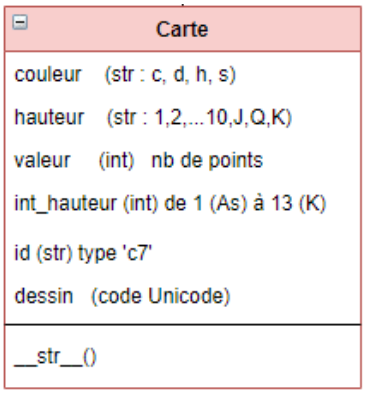
SPRINT 1	*Jalon 1*	Réponses aux questions p4. Classes Carte, Tapis et PaquetCartes opérationnelles, diagramme des classes construit
SPRINT 2	*Jalon 2*	Classe MainJoueur créée avec 6 méthodes opérationnelles. Classe Partie : méthode start() complétée. Diagramme des classes à jour
	Jalon 2bis	Tableau p6 complété. MainJoueurIA créée. Méthodes choix_input(carte) et choix_output() avec IA jouant au hasard. Méthode simul_jeu() basique avec quelques tours de jeu sans compter les points, robuste aux erreurs de saisie. Diagramme des classes à jour.
SPRINT 3	*Jalon 3*	Méthode compter_points(). Méthode simul_jeu() finalisée mais toujours avec IA jouant au hasard. Diagramme des classes à jour.
SPRINT 4	*Jalon 4*	Amélioration IA : méthode score 1ère version, tableaux de tests complétés, analyse et rectification des erreurs (document texte environ 1 page recto-verso), méthode score corrigée, validation de nouveaux tests. Version finale du jeu avec IA « intelligente ».
SPRINT 5	*Jalon 5*	Ajout de la visualisation graphique avec Tkinter. Réponse aux questions p 9, méthodes affiche_cartes et mise_a_jour complétées, test du module table, branchement du module sur le jeu pour visualisation graphique.

Sprint 1 : les classes Carte, PaquetCartes et Tapis

L'objectif de ce premier sprint est de disposer de 3 classes essentielles au déroulement du jeu : pour modéliser une carte, un paquet de carte, et le tapis (ou la « défausse »)

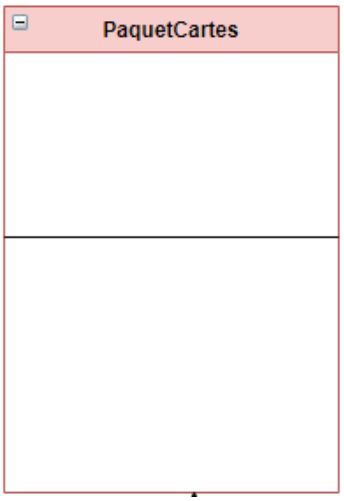
Classe Carte

Voici la spécification de la classe $Carte$. Le code associé se trouve dans le script $carte\_paquet.py$. Ouvrez ce script, et répondez aux questions suivantes :



- Où est définie la variable DICO_HAUTEURS ?
- Quel est le code Unicode en hexadécimal pour le symbole du 7 de carreaux ? Et pour la Dame de Pique ? ……
  ……
- Compléter le code du constructeur de la classe Carte en rajoutant les attributs manquants.
- Que se passerait-il si on utilisait $print()$ pour un objet de classe $Carte$ et qu'on n'avait pas défini de méthode $\_\_str\_\_()$?

  ……
- Quelle instruction écrire pour créer la variable $carte1$ représentant le Valet de Cœur.
  ……
- On suppose qu'on dispose d'une liste ℓ de cartes. Ecrire une instruction utilisant la méthode sort pour trier cette liste par attribut $int\_hauteur$ décroissant
  ……
- Créer indépendamment de la classe une petite fonction utilitaire $affiche\_jeu(liste\_cartes)$ qui permet d'afficher en ligne les cartes d'une liste d'objets cartes. Elle vous sera utile pour les tests



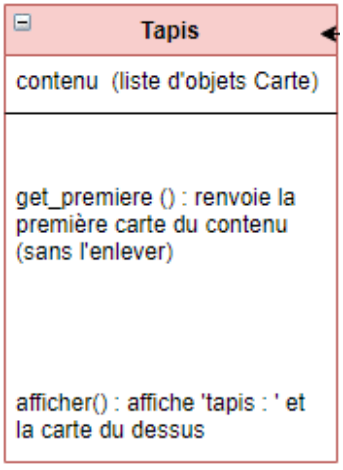
Classe PaquetCartes

Vous trouverez également dans le script le code (incomplet) de la classe PaquetCartes.

- Utilisez-le pour écrire les spécifications de cette classe sous forme d'un tableau comme ci-dessus. Créez le diagramme et ajoutez les 2 classes.
- Complétez dans le script le code des méthodes de la classe

Tests du module carte_paquet.py

- Vérifiez le bon fonctionnement du module $carte\_paquet.py$ avec les tests en bas du script.
- Quel argument ajouter dans la fonction $print()$ pour que les cartes soient affichées les unes à côté des autres, séparées par un tiret, et non plus les unes en-dessous des autres ?
  ……
- Ajouter un test pour vérifier le fonctionnement de la méthode $remplir(cartes)$



Classe Tapis

- Quelle structure classique en informatique permet de représenter le fonctionnement du tapis au pharaon ? ……
- Rajouter alors dans les spécifications de la classe Tapis les 3 méthodes classiques de cette structure, puis les coder dans le script $tapis.py$
- Compléter les tests à la fin du script. Ajoutez la classe Tapis au diagramme

Jalon 1 et livrables : classes Carte, Tapis et PaquetCartes opérationnelles, diagramme des classes construit

Sprint 2 : quelques tours de jeu sans les points, IA basique

Le premier jalon a été franchi : vous disposez des scripts carte_paquet.py et tapis.py qui sont testés et opérationnels, et vous avez la description des classes correspondantes. ***L'objectif de ce nouveau sprint est de construire une version très simplifiée du jeu. L'IA jouera de façon très basique (en choisissant une carte au hasard dans son jeu). Le joueur, lui, doit pouvoir piocher / ramasser / rejeter par des entrées au clavier.***

1. Méthodes de base pour MainJoueur et Partie :

Deux nouvelles classes interviennent maintenant :

- **La classe $\boldsymbol{MainJoueur}$**, qui va décrire le joueur et sa main (son jeu). Elle contient la position du joueur*(Nord, Sud, Est ou Ouest : au pharaon, comme il n'y a que 2 joueurs, ce sera Sud pour le joueur et Nord pour l'IA)*, et les cartes qu'il a en main.

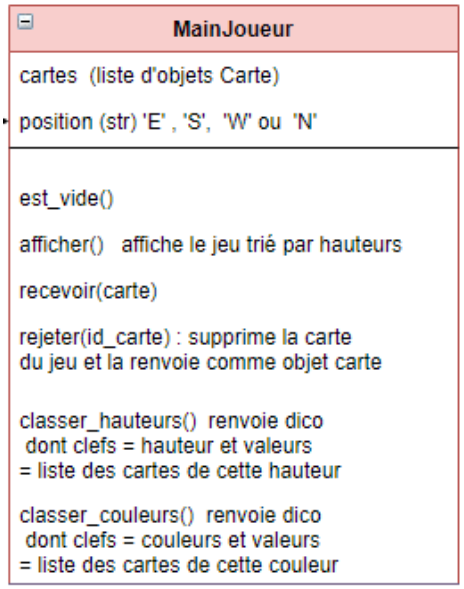
- **La classe $\boldsymbol{Partie}$**, qui est la classe « de plus haut niveau » dans notre application :
  - C'est elle qui dispose des informations générales du jeu : combien il y a de joueurs, qui est le « donneur » (celui qui distribue), combien de cartes il faut distribuer, etc.
  - C'est elle qui va initialiser le jeu, avec sa méthode **start()**, en appelant les constructeurs des autres classes *(PaquetCartes, MainJoueur, Tapis)* , en distribuant les cartes
  - C'est elle enfin qui va lancer le jeu, avec sa méthode principale : **simul_jeu()**

Première ébauche de la classe MainJoueur

Voici ci-contre une première spécification de la classe $MainJoueur$.

Créer un script $mainjoueur.py$ et implémenter cette classe avec les attributs et méthodes indiqués.

N'oubliez pas les tests à la fin du script, pour valider chaque méthode.



Première ébauche de la classe Partie

Ouvrir le script $partie.py$ et compléter les lignes manquantes.
Lancer le test final pour valider le fonctionnement.

Mise à jour du diagramme des classes

A partir du script $partie.py$, donnez les spécifications de la classe Partie. Ajouter les classes Partie et MainJoueur au diagramme des classes.

Jalon 2 et livrables : classe MainJoueur : les 6 méthodes ci-contre opérationnelles.
Classe Partie : start() opérationnelle. Diagramme des classes à jour

2. Notion d'héritage : la classe MainJoueurIA

L'IA est un joueur avec une main tout comme le « vrai joueur », mais il a des spécificités particulières :

Pour **choisir** quelle carte à rejeter ce sera différent :

- une entrée clavier (ou un clic canevas) pour le joueur
- un calcul interne pour l'IA, qui va être amené à « scorer » ses cartes pour savoir laquelle est la moins intéressante. C'est une méthode qui lui sera spécifique (le joueur fait ça dans sa tête).

De même, pour savoir si on ramasse la carte de la défausse ou si on pioche plutôt une carte au talon, la façon de choisir sera différente entre le joueur et l'IA.

En revanche, il y a de nombreux points communs : le fait de recevoir dans son jeu ou de rejeter une carte fixée, de compter les points… fonctionneront de façon identique entre le joueur et l'IA.

On est donc amené à créer une classe MainJoueurIA qui sera **dérivée** de la classe MainJoueur. Pour indiquer cela, on rajoute entre les parenthèses le nom de la classe parente :

```
class MainJoueurIA(MainJoueur):
    '''
    hérite de la classe MainJoueur
```

- Cette classe bénéficiera par **héritage** des attributs et méthodes de la classe MainJoueur.
- Son constructeur fera appel au constructeur de la classe parente, en lui transmettant *self* comme premier argument. Il pourra rajouter de nouveaux attributs en plus.

```
def __init__(self, cartes,  position='N'):
    #on appelle le constructeur de la classe parente
    MainJoueur.__init__(self, cartes, position)
    #ici c'est tout (pas d'attribut spécifique
```

- La classe MainJoueurIA pourra redéfinir certaines méthodes de sa classe parente de façon différente, tout en gardant leur nom : on appelle cela **surcharger** une méthode.
- Elle pourra également avoir des méthodes spécifiques (« rien qu'à elle »).

A faire 1 :

Parmi les méthodes suivantes de la classe MainJoueurIA, indiquez son lien vis-à-vis de la classe MainJoueur. Est-ce une méthode « identique », « nouvelle » ou « surchargée » ?

$choix\_output()$	Renvoie l'id de la carte à rejeter	
$classer\_hauteurs()$	Renvoie un dico avec la liste des cartes par hauteur	
$choix\_input(carte)$	Renvoie 1 si on ramasse la carte sur la défausse, 0 si on pioche au talon	
$compter\_points()$	Analyse le jeu du joueur, compte les points, vérifie s'il y a pharaon	
$afficher()$	Affiche à l'écran les carte du jeu	
$classer\_couleurs()$	Renvoie un dico avec la liste des cartes par couleur	
$rejeter(id\_carte)$	Supprime du jeu et renvoie la carte indiquée	
$score(carte)$	Quantifie l'intérêt de la carte en renvoyant un nombre entre 0 et 1	
$recevoir(carte)$	Ajoute dans le jeu la carte indiquée	

A faire 2 :

Reprendre le script $mainjoueur.py$:

- Rajouter la classe $MainJoueurIA(MainJoueur)$
- Coder les méthodes $choix\_input(carte)$ et $choix\_output()$. On fera pour le moment jouer l'IA de façon très bête : elle tirera au sort si elle doit piocher ou ramasser, elle tirera au sort la carte à rejeter dans son jeu.

Mettre le diagramme des classe à jour !

A faire 3 :

Reprendre le script $partie.py$: écrire une première version très simple de la méthode $simul\_jeu()$

- Quelques tours de jeu seulement (4 ou 5)
- Le joueur et l'IA, tour à tour, piochent ou ramassent, puis rejettent. On voit le jeu du joueur affiché au fur et mesure de son évolution. <u>On ne compte pas les points pour l'instant !</u>

Jalon 2bis et livrables : simul_jeu() dans sa première version simple, mais robuste aux erreurs de saisie. L'IA joue au hasard. Diagramme des classes à jour

3. Sprint 3 : Compter les points

a. Compter les points, déterminer s'il y a pharaon

Rappels :

- Seules les cartes non intégrées à un figure comptent dans les points.
- Avoir pharaon signifie qu'il reste au plus une carte non intégrée à une figure, et que cette carte a une valeur inférieure ou égale à 5.

A chaque tour de jeu, le joueur doit analyser son jeu, regarder s'il a « pharaon » ou non, compter ses points. On est donc amené à **ajouter deux attributs et une méthode** dans la classe MainJoueur :

- pts : un entier dénombrant le nombre de points dans la main
- $pharaon$: un Booléen indiquant si le joueur a pharaon ou pas.
- $compter\_points()$: une méthode qui analyse le jeu du joueur, et met à jour les deux attributs précédents en conséquent.

A faire :

- ✓ Mettre à jour le diagramme des classes.
- ✓ Reprendre le script $mainjoueur.py$, rajouter les deux attributs précédents, puis coder la méthode $compter\_points()$.
- ✓ Tester la méthode $compter\_points()$ à la fin du script, en préparant quelques mains variées *(il faut couvrir l'ensemble des configurations possibles : brelans, carrés, suites, pharaon ou pas…).*

b. Mise au point de simul_jeu()

Une fois que l'étape précédente est franchie, il est assez simple de modifier la méthode $simul\_jeu()$ dans la classe Partie, pour qu'elle simule vraiment une partie de pharaon. Non plus quelques tours, mais on joue jusqu'à ce que l'un des deux joueurs déclare pharaon.

On ne peut déclarer pharaon qu'au début de son tour (on ne pioche alors pas de carte).

Dans cette version, l'IA joue toujours de façon « très bête » (elle choisit au hasard !) Elle a donc peu de chances de gagner….

```
tapez 1 pour ramasser la carte  s5
tapez 0 si vous préférez piocher au talon : 1
votre jeu :
hQ 🂻-dQ 🂻-cQ 🂷-c7 🂥-s5 🂤-c4 🂤-s4 🂤-d4 🂤-

choisissez la carte à rejeter: (exple h6) : c7
l'IA a rejeté h4
joueur gagne avec pharaon 5
```

Jouer quelques parties pour vérifier le bon fonctionnement du jeu.

BONUS

On peut intégrer la règle particulière suivante : l'As de Cœur peut remplacer n'importe quelle carte.

Jalon 3 et livrables : simul_jeu() finalisée avec IA basique. Diagramme des classes à jour

4. Sprint 4 : amélioration de l'IA

Pour jouer comme un « vrai joueur », l'IA doit réfléchir à deux choses :

- A-t-elle intérêt à ramasser la carte du tapis, ou plutôt à piocher au talon ?
- Quelle carte de son jeu est la moins intéressante, donc susceptible d'être rejetée ?

Ces deux aspects se résument à attribuer un « score » à chaque carte, une note (exple : un float entre 0 et 1) reflétant son intérêt au vu du jeu de l'IA.
On va donc créer une méthode $score(carte)$ qui renvoie un float entre 0 et 1.
Dans quelle classe doit se trouver cette méthode ? ……..

A faire :

- ✓ **Concevoir une première version de la méthode $score(carte)$, et rédiger en parallèle un ou deux paragraphes explicatifs** qui détaillent votre raisonnement (naturellement le code doit être également commenté)
- ✓ Avec le jeu [sK, cK, cJ, h8, d7, d5, s2] : quel serait votre choix face aux tapis suivants ? Ramasser la carte du tapis ? Ou piocher au talon ? Quel est-il de votre IA ? (montrez les résultats des tests)

Carte sur le tapis	Ramasser ou piocher ?	Validation programme ? (IA)	Carte sur le tapis	Ramasser ou piocher ?	Validation programme ? (IA)
hK			**d10**		
c10			**c4**		
d3			**sJ**		
h6			**d6**		
cQ			**s9**		
d1			**c9**		
hK			**h3**		

- ✓ Quel score entre 0 et 1 attribuez-vous "naturellement » aux cartes de chaque jeu ? Et l'IA, quel score a-t-elle attribué avec votre programme ? Quelle carte rejetteriez-vous spontanément ? Quel est le choix de l'IA dans votre programme (montrez les résultats des tests).

	Jeu de l'IA (8 cartes dont une à rejeter)								**Carte à rejeter**
Jeu	sK	cK	cQ	cJ	h8	d7	d5	s2	
Votre score									
Score IA									
Jeu	sK	cK	cJ	h8	d7	h6	d5	s2	
Votre score									
Score IA									
Jeu	sK	d8	s6	c6	c2	h2	d2	s1	
Votre score									
Score IA									
Jeu	sK	cQ	dJ	c10	h8	d8	c8	s2	
Votre score									
Score IA									
Jeu	sK	cK	dK	hK	h8	d8	c8	s7	
Votre score									
Score IA									

- ✓ **Si vous constatez des écarts**, ce qui serait logique au moins au début (il est rare d'avoir tout bon du 1er coup…), **il faut corriger cette première version de la méthode $score(carte)$**
  - Cherchez d'où ils viennent : « breakpoints » dans le code, affichage du Workspace, des « print » aux endroits importants, puis faire tourner le débugger jusqu'à comprendre ce qui se passe.
  - Expliquez d'où provenaient les erreurs, tant du fond (des cas à traiter oubliés) que de la forme (des coquilles dans le code par exemple)
  - Expliquez pour deux ou trois de ces erreurs comment vous avez rectifié votre code pour qu'il produise de meilleurs résultats.

- ✓ Préparer au moins 4 autres tests pour vérifier le fonctionnement de cette nouvelle version méthode, (décrire les tests dans un tableau, puis faire des copies d'écran

Jalon 4 et livrables : Jeu version textuelle avec IA « intelligente ». Diagramme des classes à jour. Document analysant le fonctionnement de l'IA, avec tests de validation, explication des éventuelles erreurs majeures, puis rectifications apportées

5. Sprint 5 : visualisation graphique

Dans cette partie, on construit une classe Table qui permettra d'avoir une **visualisation graphique du jeu** sur un Canvas Tkinter. Ce ne sera pas vraiment une « interface », dans le sens où le joueur continuera à entrer ses choix par des saisies clavier. Néanmoins, on verra au fur et à mesure de la partie le Canvas s'actualiser pour illustrer les cartes en main, la défausse…

Ouvrir le script $table.py$

Questions générales

Table

- ➢ Donner les spécifications de la classe Table et la rajouter dans le diagramme des classes.
- ➢ Quelles sont les dimensions du Canvas ? ……
- ➢ Dans le cas du jeu de pharaon :
  - Combien y a-t-il de clefs dans l'attribut $dicos\_images$? …… Quelles sont-elles ? ……
  - Quel est le type des valeurs associées aux clefs précédentes ? ……
  - Combien d'éléments contiennent ces valeurs ? …… Quels sont ces éléments ? ……
- ➢ Quelle est la valeur du paramètre angle dans l'instruction $img\_dos.rotate(angle, expand = 1)$ de la ligne 49 pour la position de l'IA (Nord) ……
  Expliquez le rôle de cette instruction et son intérêt dans l'interface graphique ……
  ……

- Chercher sur Internet le rôle du paramètre $extend = 1$: ……
  ……
  ……

Méthodes affiche_cartes() et mise_a_jour()

- Expliquez à quoi correspondent les variables x_start, y_start, dx, dy (ligne 67), et leur valeur dans le cas où la position est celle du joueur (Sud) ……
  ……
  ……
  ……
  ……

- La méthode $affiche\_cartes()$ est incomplète : écrivez les instructions aux lignes 78 et 83 qui permettent d'afficher le jeu de la main.
- Complétez de même la méthode $mise\_a\_jour()$ pour afficher le talon et la première carte de la défausse.

Tests du module table.py

A la fin du module, dans le if __name__ == '__main__', écrivez des instructions permettant d'afficher le résultat suivant :

Code :

Utilisation de la classe Table dans le jeu

Reprendre le code de la classe $Partie$ pour inclure une visualisation de la table au fur et à mesure du jeu. Le jeu de l'IA doit être caché, sauf à la toute fin du jeu, si c'est elle qui a gagné (dans ce cas, son jeu doit être révélé).

Jalon 5 : réponses aux questions de cette partie, version du jeu avec visualisation graphique fonctionnelle.