Projet : la réussite des alliances

1 Organisation du projet

Ce projet se compose d'un sujet (ce document), de quelques fichiers à télécharger ainsi que d'une activité avec évaluation automatique sur Caseine. Cette dernière permet de tester les fonctions demandées dans la partie 4 à l'exception de `reussite_mode_manuel` et `lance_reussite`. Parmi les extensions proposées dans la partie 5, seule `meilleur_echange_consecutif` est évaluée automatiquement, mais vous êtes encouragés à essayer de faire et tester les autres par vous-mêmes.

Votre programme devra être clair, élégant et commenté. Pensez en particulier à utiliser des noms de variables explicites. Toutes les fonctions (y compris celles non demandées) devront être documentées avec une Docstring.

N'hésitez pas à consulter la documentation en ligne de Python 3 :

`https://docs.python.org/3/library/index.html`

2 Le thème du projet : la réussite des alliances

Dans ce projet, nous nous intéressons à une réussite peu connue qui se joue avec un jeu de 32 ou de 52 cartes : la réussite des alliances. On commence par donner les règles de cette réussite.

On mélange les cartes, on obtient un tas de cartes qui va constituer la pioche. On prend les cartes une à une dans la pioche et on les pose côte à côte de gauche à droite. Quand on a placé trois cartes, on considère les cartes 1 et 3 (en les numérotant de gauche à droite). Si elles sont de même couleur ou de même valeur (ce qu'on appellera une alliance), on prend la carte située entre elles (ici la carte numéro 2) et on la pose sur la carte 1 (sinon on ne fait rien et on continue à piocher). C'est ce qu'on va appeler un "saut". On n'a donc potentiellement plus que deux tas.

Par exemple, si les 3 premières cartes piochées sont dans l'ordre un 8 de cœur, un 7 de carreau et un roi de trèfle, aucun saut n'est possible :

$$8\heartsuit \quad 7\diamondsuit \quad R\clubsuit$$

Par contre, si les 3 premières cartes piochées sont dans l'ordre un 8 de cœur, un 7 de carreau et un roi de cœur :

$$8\heartsuit \quad 7\diamondsuit \quad R\heartsuit$$

on devra faire "sauter" le 7 de carreau sur le 8 de cœur avant de piocher la carte suivante, et on obtiendra :

$$7\diamondsuit \quad R\heartsuit$$

On continue alors à piocher des cartes et à les poser une par une sur la droite de la réussite. Dès qu'il y a une alliance entre la dernière carte posée et l'antépénultième, le tas placé entre les deux est décalé et posé sur le tas à sa gauche ("saut"). Ainsi à tout moment de la partie, on a des tas de cartes, posés de gauche à droite. Pour un tas donné, le nombre de cartes du tas importe peu dans la mesure où il n'y a que la carte visible sur le dessus du tas qui compte pour la suite de la partie. On parlera donc de tas, même si le tas en question est composé d'une seule carte.

Par exemple, si on continue la partie suivante, en supposant qu'on pioche successivement l'as de cœur, le 10 de carreau et le 9 de pique, aucun saut ne sera possible, et on aura :

$$7\diamondsuit \quad R\heartsuit \quad A\heartsuit \quad 10\diamondsuit \quad 9\spadesuit$$

On pioche donc la carte suivante, un 9 de carreau :

$$7\diamondsuit \quad R\heartsuit \quad A\heartsuit \quad 10\diamondsuit \quad 9\spadesuit \quad 9\diamondsuit$$

Ici un saut est possible car le tas du 9 de pique est situé entre deux carreaux, et on pose donc le tas du 9 de pique sur le tas du 10 de carreau :

$$7\diamondsuit \quad R\heartsuit \quad A\heartsuit \quad 9\spadesuit \quad 9\diamondsuit$$

Parfois, lorsqu'on fait un saut, cela rend d'autres sauts possibles, ce n'est pas le cas ici. Si au moins un autre saut est possible, il faut le faire avant de piocher la carte suivante. Si plusieurs sauts sont possibles, il faut décider lequel on fait en premier. On décide de faire toujours en premier le saut le plus à gauche (ce qui peut en entraîner un autre encore plus à gauche).

Pour illustrer ces sauts en cascade, imaginons par exemple qu'à un moment de la partie, on ait le jeu suivant :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad A\spadesuit \quad V\clubsuit \quad D\clubsuit \quad D\spadesuit$$

Ici, il n'y a toujours aucun saut possible. On pioche la carte suivante, un 7 de trèfle, et on la place à droite :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad A\spadesuit \quad V\clubsuit \quad D\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Ce 7 de trèfle est un trèfle tout comme la carte situé deux rangs avant lui, donc on prend le tas de la dame de pique qui est entre le 7 de trèfle et la dame de trèfle et on le pose sur le tas de la dame de trèfle. Après ce saut, on obtient :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad A\spadesuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Ici c'est intéressant car on a deux sauts possibles : on peut faire sauter le tas du valet de trèfle ou le tas de la dame de pique. Mais si on pose d'abord la dame de pique sur le valet de trèfle on ne pourra plus faire l'autre saut, d'où l'idée de toujours commencer par le saut le plus à gauche. Donc on fait sauter la carte (ou le tas de cartes) situé entre l'as de pique et la dame de pique, et c'est donc le tas du valet de trèfle qui sera posé sur l'as de pique :

$$9\clubsuit \quad V\diamondsuit \quad 10\heartsuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Maintenant, avant de faire l'autre saut qu'on avait repéré, il faut vérifier s'il n'y en a pas un nouveau plus à gauche. On repart du début du jeu à gauche pour chercher les sauts possible. Ici un nouveau saut est apparu entre le valet de carreau et le valet de pique. On pose donc le tas du 10 de cœur sur le valet de carreau :

$$9\clubsuit \quad 10\heartsuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Le prochain saut à faire est de poser le tas du 10 de cœur sur le 9 de trèfle :

$$10\heartsuit \quad V\clubsuit \quad D\spadesuit \quad 7\clubsuit$$

Et maintenant le premier saut à faire est celui que nous avions repéré il y a quelques étapes déjà : on pose le tas de la dame de pique sur le valet de trèfle.

$$10\heartsuit \quad D\spadesuit \quad 7\clubsuit$$

Ainsi sans piocher d'autre carte après le 7 de trèfle, on est passé d'un jeu à 8 tas à un jeu à 3 tas.

Ici aucun autre mouvement n'est possible. On pioche la carte suivante et on continue la réussite.

La partie s'arrête quand on a épuisé la pioche et qu'on ne peut plus faire diminuer le nombre de tas. On compte alors le nombre de tas restants à la fin de la réussite. On a gagné si le nombre de tas est inférieur à un certain seuil fixé par la règle du jeu. On testera plusieurs seuils dans ce projet. On a toujours au moins 2 tas à la fin (et le dernier n'a qu'une carte), c'est la fin "parfaite" mais c'est suffisamment rare pour qu'on soit plus tolérant pour estimer que la réussite est gagnée, surtout si on joue avec 52 cartes.

3 Représentation informatique de cette réussite

Dans ce projet, on va programmer cette règle de réussite et faire plusieurs études pour estimer par exemple la probabilité de gagner une partie.

Pour modéliser cette réussite, on va considérer qu'à tout moment de la partie, la réussite est modélisée par une liste de cartes qui sont les cartes visibles sur les dessus des tas. Ainsi le nombre de tas à la fin de la partie sera tout simplement le nombre d'éléments de cette liste.

Pour coder une carte dans le programme, on va utiliser un dictionnaire à deux clés : `'valeur'` et `'couleur'`.

Pour la valeur de la carte, un as sera représenté par `'A'`, un deux par l'entier `2`, un trois par l'entier `3` et un dix par `10`. Pour les figures, on écrira respectivement `'V'`, `'D'` et `'R'` pour valet, dame et roi.

Pour la couleur, on aimerait utiliser l'initiale majuscule de chaque couleur (Cœur, Pique, Carreau et Trèfle) mais comme cœur et carreau ont la même initiale, on choisit la lettre `'K'` pour représenter carreau.

Ainsi le dictionnaire `{'valeur' :7, 'couleur' : 'P'}` représentera le 7 de pique.

La liste `[{'valeur':7, 'couleur':'P'}, {'valeur':10, 'couleur':'K'}, {'valeur':'D', 'couleur':'K'}]` correspondra à l'état suivant de la réussite :

7♠ 10♢ D♢

La pioche sera également modélisée par une liste de cartes, la carte à l'indice 0 étant la première carte à piocher.

On va maintenant pouvoir écrire les fonctions permettant de faire jouer l'ordinateur à cette réussite.

4 Travail demandé

Dans cette partie, on a réparti les différentes fonctions à écrire dans plusieurs catégories. Autant que possible, testez vos fonctions au fur et à mesure que vous les écrivez.

Attention : pour toutes les fonctions demandées dans ce projet, quand il y a des listes en argument, ces listes ne doivent pas être modifiées par la fonction, sauf si c'est explicitement demandé. N'hésitez pas à faire une copie de la liste si nécessaire.

4.1 Affichage de la réussite

4.1.1 Fonction `carte_to_chaine`

Pour afficher la réussite sur la console Python, on souhaite avoir un affichage plus explicite que `"DK"` par exemple pour la dame de carreau. On va afficher les cartes de façon assez similaire à ce qui est fait dans les exemples de la partie 2.

Pour cela, on doit savoir représenter les caractères spéciaux ♡ ♠ ♢ ♣. En fait, en Python, chaque caractère correspond à un entier qui code ce caractère. Pour savoir quel est le code de la lettre `'a'` par exemple, il faut faire afficher `ord(a)`. Vous trouverez alors que l'entier associé au caractère `'a'` est 97. Réciproquement, pour obtenir le caractère correspondant à un entier donné, il faut utiliser la fonction `chr`. Ainsi par exemple `chr(97)` retournera `'a'`. Voici les codes qui vous seront utiles :

Caractère	Code
♠	9824
♡	9825
♢	9826
♣	9827

Ainsi, par exemple, pour créer la chaîne de caractères permettant d'afficher la dame de carreau, il faudra faire : `'D' + chr(9826)`

Ecrire la fonction `carte_to_chaine` qui prend en argument un dictionnaire représentant une carte, et retourne une chaîne de caractères permettant l'affichage de la carte correspondante, sur 3 caractères, avec éventuellement un espace à gauche. Un espace est nécessaire pour toutes les cartes qui ne sont pas un 10. Attention, cette fonction ne produit aucun affichage.

Exemples : (les espaces ont été matérialisés par le caractère ␣)
- `carte_to_chaine({'valeur':7, 'couleur':'P'})` retourne la chaîne "␣7♠"
- `carte_to_chaine({'valeur':10, 'couleur':'K'})` retourne la chaîne "10♢"
- `carte_to_chaine({'valeur':'R', 'couleur':'C'})` retourne la chaîne "␣R♡"

4.1.2 Fonction `afficher_reussite`

Ecrire une fonction `afficher_reussite` qui prend en argument une liste de cartes correspondant à l'état de la réussite à un moment donné et affiche la réussite. On utilisera la fonction `carte_to_chaine`. On affichera les différentes cartes sur une même ligne, séparées par un espace (en plus de celui éventuellement mis par `carte_to_chaine`).

Exemple :
`afficher_reussite([{'valeur':7, 'couleur':'P'}, {'valeur':10, 'couleur':'K'}, {'valeur':'A', 'couleur':'T'}])` affichera "␣7♠␣10♢␣␣A♣" suivi d'une ligne entièrement blanche.

4.2 Entrées / Sorties avec des fichiers

4.2.1 Fonction `init_pioche_fichier`

Cette fonction lit une suite de cartes dans le fichier dont le nom est passé en argument, et renvoie la liste de cartes correspondante. On suppose que les valeurs dans le fichier sont correctes, c'est-à-dire qu'elles correspondent toutes à une carte existante, et qu'il y a le bon nombre de cartes (32 ou 52). Dans le fichier, on a la liste des cartes dans l'ordre où elles seront piochées, les cartes étant séparées par des espaces. Pour chaque carte, il est écrit sa valeur puis sa couleur, séparées par un tiret. Par exemple, si on a "7-K", cela correspondra au 7 de carreau et "D-P" représentera la dame de pique. Pour cette question, un fichier d'exemple de pioche `data_init.txt` est fourni sur Caseine.

4.2.2 Fonction `ecrire_fichier_reussite`

On souhaite pouvoir sauvegarder dans un fichier l'ordre des cartes de la pioche d'une réussite donnée, par exemple pour mémoriser le mélange qui a donné le meilleur résultat parmi plusieurs mélanges.

Ecrire pour cela la fonction `ecrire_fichier_reussite` qui prend en argument un nom de fichier `nom_fich` dans lequel il faut écrire la liste des cartes `pioche` fournie en deuxième argument de la fonction. Le fichier ainsi écrit doit pouvoir être lu par la fonction `init_pioche_fichier`. Cette fonction ne renvoie rien.

4.3 Générer une pioche mélangée aléatoirement

4.3.1 Fonction `init_pioche_alea`

Ecrire la fonction `init_pioche_alea`. Cette fonction a un seul argument, optionnel, nommé `nb_cartes` qui vaut 32 par défaut, mais qui peut aussi valoir 52. Cette fonction renvoie une liste contenant toutes les cartes du jeu, et les mélange (pour mélanger une liste, vous pourrez utiliser la fonction `shuffle` du module `random`).

4.3.2 Fonction verifier_pioche

Écrire une fonction `verifier_pioche` : elle prend en argument la liste des cartes constituant la pioche et un entier `nb_cartes` qui vaut 32 ou 52 (argument optionnel, 32 par défaut) et elle vérifie si cette pioche contient bien toutes les cartes d'un jeu à `nb_cartes`, sans doublon. Elle renvoie vrai si c'est le cas et faux sinon.

4.4 Programmer les règles de la réussite des alliances

4.4.1 Fonction `alliance(carte1,carte2)`

Ecrire une fonction `alliance(carte1,carte2)` qui prend en argument deux cartes, et qui renvoie vrai si les cartes ont la même valeur ou la même couleur et faux sinon.

Exemples :

- `alliance({'valeur':7, 'couleur':'P'},{'valeur':7, 'couleur':'C'})` renvoie vrai
- `alliance({'valeur':7, 'couleur':'P'},{'valeur':8, 'couleur':'C'})` renvoie faux.

4.4.2 Fonction `saut_si_possible`

Cette fonction prend en argument une liste `liste_tas` correspondant à la liste des cartes visibles sur les tas de la réussite et un entier `num_tas`. La fonction vérifie s'il est possible de faire sauter le tas d'indice `num_tas` sur le tas qui le précède (selon les règles de la réussite). Si le saut est possible, la fonction le fait. La fonction retourne vrai si le saut a été fait et faux sinon. Elle n'affiche rien.

Attention : cette fonction modifie donc la liste qui lui est donnée en argument.

4.4.3 Fonction `une_etape_reussite`

Cette fonction prend en argument une liste `liste_tas` correspondant aux cartes visibles des tas de la réussite, une liste `pioche` contenant les cartes restant dans la pioche ainsi qu'un argument optionnel booléen `affiche` valant `False` par défaut. Cette fonction doit effectuer une étape de la réussite, c'est-à-dire :

- tirer la première carte de `pioche` et la placer dans `liste_tas` pour qu'elle corresponde au tas le plus à droite de la réussite,
- faire le saut s'il est possible entre la carte qu'on vient de poser et le tas situé deux rangs plus à gauche (c'est-à-dire faire sauter la carte entre ces deux tas),
- si un saut a été fait, il faudra vérifier si cela a rendu possible d'autres sauts. Pour cela on partira de la gauche de `liste_tas` et on fera le premier saut possible. On recommencera cette action tant qu'un changement aura lieu.
- en plus de tout ça, si l'argument `affiche` vaut vrai, la fonction affichera chacun des états de la réussite pour l'étape en cours. C'est-à-dire qu'à chaque changement (pioche ou saut) il faudra faire un affichage de la réussite. Et si `affiche` vaut faux, rien ne sera affiché.

Attention : cette fonction ne retourne rien mais elle modifie les deux listes qui lui sont données en argument (car on a enlevé la première carte de la `pioche` pour la mettre dans `liste_tas`).

Il est recommandé d'utiliser des fonctions auxiliaires pour programmer correctement cette fonction.

4.5 Faire une partie

4.5.1 Fonction `reussite_mode_auto`

Cette fonction prend en argument une liste de cartes `pioche` correspondant à l'ensemble du jeu de cartes déjà mélangé et un booléen optionnel `affiche` ayant les mêmes spécifications qu'à la question précédente. Si `affiche` vaut vrai, on commence par afficher le contenu de la pioche avant de jouer la réussite en affichant les étapes. Cette fonction joue l'ensemble de la réussite en affichant chaque étape si `affiche` vaut vrai. Le second affichage correspond alors à la première carte de la pioche retournée. La fonction ne doit pas modifier la liste `pioche` (vous pouvez en faire une copie) et elle doit retourner la liste des cartes visibles sur les tas restant à la fin de la réussite.

4.5.2 Fonction `reussite_mode_manuel`

On souhaite maintenant laisser l'utilisateur prendre lui-même les décisions comme si c'était lui qui jouait quitte à ce qu'il rate des sauts, mais sans le laisser tricher.

Pour cela, écrire une fonction `reussite_mode_manuel` qui, comme la fonction précédente, prend en argument une liste de cartes `pioche` correspondant à l'ensemble du jeu de cartes déjà mélangé. Ici, il n'y a pas d'argument `affiche` car on veut forcément afficher les étapes pour que le joueur puisse voir où il en est. Par contre, la fonction prendra un argument optionnel `nb_tas_max` (par défaut, réglé à 2) qui fixe le nombre maximum de tas qu'il peut rester à la fin pour que la partie soit déclarée comme gagnante. L'ordinateur proposera un menu au joueur : il lui proposera par exemple de découvrir une carte de la pioche, ou de saisir un saut à faire. Il faudra aussi que le joueur puisse quitter s'il le souhaite. Si le joueur demande à faire un saut non autorisé, l'ordinateur doit refuser de le faire en précisant que le coup est impossible.

Si le joueur abandonne avant d'avoir retourné toutes les cartes, les cartes restant dans la pioche sont découvertes une à une et posées sur la réussite, sans qu'aucun saut ne soit fait. Dans tous les cas, à la fin, la fonction affiche au joueur s'il a gagné ou perdu, et retourne la liste des cartes visibles sur les tas restant à la fin de la réussite.

Cette question est volontairement assez ouverte. A vous de trouver le format qui rendra le jeu le plus agréable, et qui proposera éventuellement d'autres options.

4.5.3 Fonction `lance_reussite`

Écrire une fonction `lance_reussite` qui prend en argument une chaîne de caractères `mode` qui vaudra soit `'manuel'` soit `'auto'`, ainsi que trois arguments optionnels `nb_cartes` (par défaut, 32) et un booléen `affiche` (par défaut, False) et un entier `nb_tas_max` (par défaut, 2). Cette fonction doit mélanger aléatoirement un jeu contenant le bon nombre de cartes et lancer la réussite en mode manuel ou en mode auto. L'argument `affiche` ne sert que pour le mode auto, et l'argument `nb_tas_max` ne sert que pour le mode manuel. La fonction doit renvoyer la liste des tas restants à la fin de la partie.

5 Extensions

Dans cette partie, on propose un certain nombre d'extensions ou d'améliorations possibles. Ici, les instructions sont moins directives que dans la partie précédente, à vous de prendre des décisions pour faire au mieux. Vous pouvez aussi imaginer d'autres extensions.

5.1 Améliorer un mélange en trichant "un peu"

Si vous avez un peu pratiqué cette réussite, vous vous serez rendus compte que l'on a souvent l'impression qu'à pas grand-chose près, en trichant juste un peu, en échangeant discrètement deux cartes consécutives qui ne sont vraiment pas arrivées dans le bon ordre, le résultat aurait été très différent. On va tester si cette impression est justifiée.

Ecrivez une fonction `meilleur_echange_consecutif` qui prend en paramètre la liste des cartes correspondant à la pioche et cherche la meilleure façon d'améliorer cette réussite en faisant un seul échange, entre deux cartes consécutives. Cette fonction a deux valeurs de retour : elle renvoie la liste des cartes correspondant à la pioche donnée en entrée dans laquelle on a fait le meilleur échange possible entre deux cartes consécutives et elle renvoie aussi le nombre de tas que nous fait gagner cet échange (3 par exemple si la réussite initiale donnait 7 tas à la fin et que celle après avoir fait le meilleur échange donne 4 tas à la fin). Si plusieurs échanges conduisent au résultat optimal (nombre de tas minimum), on choisira de faire l'échange qui est le plus loin dans la pioche (le plus grand indice dans la liste `pioche`).

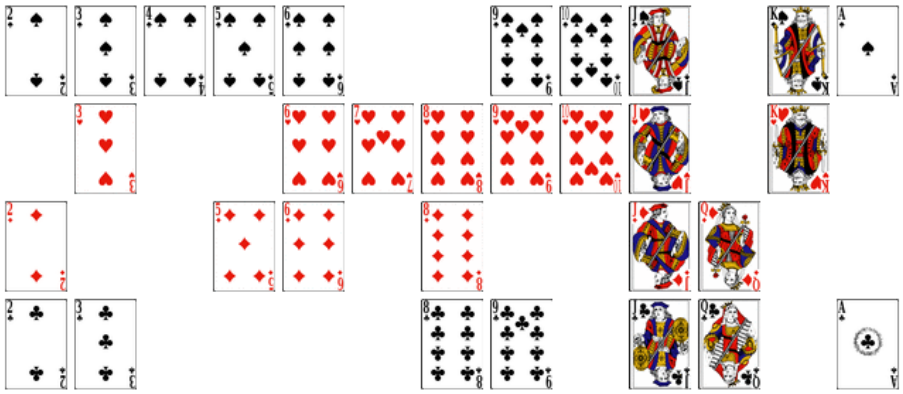
Pour aller plus loin : Etudiez en quoi cela améliore les statistiques de vos parties (nombre moyen, minimum et maximum de tas). Vous pouvez éventuellement générer de nouveaux graphiques d'estimations de probabilité quand on s'autorise un tel échange de deux cartes dans la pioche avant la partie.

5.2 Améliorer un mélange en trichant un peu plus

Sur le modèle de l'extension précédente, à vous d'imaginer d'autre façons de modifier un peu l'ordre de la pioche pour améliorer le score à la réussite. Etudiez si cela améliore beaucoup les résultats.

5.3 Interface graphique avec Turtle

Récupérez le fichier `affichage.py` ainsi que le répertoire `imgs` (qui contient les images des cartes). Exécutez et observez ce programme d'affichage.
En vous inspirant de ce programme, proposez un affichage graphique de la réussite en cours d'exécution.



5.4 Estimer le nombre moyen de tas

- Ecrire la fonction `res_multi_simulation` qui prend en argument un nombre entier `nb_sim` et en argument optionnel un entier `nb_cartes` correspondant au nombre de cartes dans le jeu (32 par défaut). La fonction génère `nb_sim` mélanges aléatoires d'une pioche d'un jeu de `nb_cartes` cartes, effectue les réussites en mode automatique, sans affichage, et retourne la liste des nombres de tas obtenus à la fin de chaque réussite. (par exemple, si `nb_sim` vaut 3, la fonction retourne [5, 3, 12] si la première réussite s'est finie avec 5 tas, la deuxième avec 3 tas et la troisième avec 12 tas).

- Ecrire la fonction `statistiques_nb_tas` qui prend en argument un nombre entier `nb_sim` et en argument optionnel un entier `nb_cartes` correspondant au nombre de cartes dans le jeu (32 par défaut). La fonction calcule et affiche avec des messages clairs la moyenne, le minimum et le maximum du nombre de tas obtenus sur `nb_sim` réussites automatiques correspondant à `nb_sim` mélanges aléatoires de la pioche. Vous pouvez également afficher d'autres indicateurs qui vous sembleront pertinents.

5.5 Estimer la probabilité de gagner et faire un graphique

Dans les règles officielles de cette réussite, on a gagné si on a seulement deux tas à la fin. Vous avez dû vous rendre compte que cela arrive assez rarement, surtout quand on a 52 cartes. On souhaiterait pouvoir quantifier ce "assez rarement" et voir comment augmentent nos chances de gagner si on autorise plus de tas (par exemple si on décide que finalement une réussite sera gagnante si elle se finit avec au maximum 4 tas). Ces probabilités semblent difficiles à obtenir par un calcul théorique, par contre, il est relativement aisé de les estimer grâce à des simulations.

En faisant un nombre important de simulations de mélange aléatoire, faites en sorte d'estimer la probabilité de gagner pour chaque nombre maximum de tas possiblement autorisé par la règle (entre 2 et 32 pour un jeu de 32 cartes).

Vous pouvez stocker ces probabilités dans une liste. Créez cette liste à partir d'une fonction.

Pour mieux visualiser ces données, vous pouvez faire un graphique en utilisant le module `pyplot` de la librairie `matplotlib`. Pour eviter d'avoir à écrire le nom `matplotlib.pyplot` devant chaque fonction de ce module que vous utiliserez, vous pouvez l'importer ainsi : `import matplotlib.pyplot as plt` et ainsi il suffira d'écrire `plt` devant chaque fonction utilisée. Pour visualiser la fenêtre graphique, après avoir donné à l'ordinateur toutes les instructions pour créer le graphique, il ne faudra pas oublier de faire `plt.show()`. Vous trouverez la documentation de `pyplot` ici :

`http://matplotlib.org/api/pyplot_api.html` *ou tapez pyplot tutorial sans un moteur de recherche*