Brun Morgan
El Anziz Aounou

Compte-rendu

Explication des fonctions :

(connue / ajoutée)

Premier programme (sans croupier) :

-paquet() : La fonction contient aucune variable et crée un paquet de carte à partir de deux listes créées avant la fonction qui contiennent la couleur (pique,coeur,…) et le style (as, dame, 3,…).

-valeurCarte() : reçoit en argument le nom d'une carte et renvoie sa valeur. Pour cela, on utilise une autre fonction que nous avons ajouté qui se nomme liste() et qui reçoit en argument le nom d'une carte aussi et qui renvoie une liste contenant le découpage des mots (nous aurions pu utiliser *split* mais nous ne connaissions pas). Pour « as de coeur » → ['as','de','coeur']. Puis dans la fonction valeurCarte(), il suffisait de donner une valeur selon le premier élément de la liste renvoyé.

-initPioche() : reçoit en argument un entier *n* qui représente le nombre de joueurs durant la partie et on crée une pioche qui est une liste contenant autant de paquets que de joueurs que nous mélangeons ensuite grâce à *shuffle*.

-piocheCarte() : reçoit en argument la pioche et le nombre de cartes que l'on souhaite piocher. Cela renvoie une liste contenant les cartes piochées, la pioche a été modifiée par la même occasion.

-initJoueurs() : reçoit en argument le nombre de joueurs de la partie et renvoie une liste contenant le prénom de chaque joueurs.

-initScores() : reçoit en argument la liste des joueurs de la partie ainsi qu'une certaine valeur *v* qui représente le score qu'on initialise à chaque joueurs. Dans le cas où il n'y a pas de mises, on s'en servira pour v=0 tandis qu'on pourra aussi se servir de cette fonction dans le cas de mises qu'on initialisera à 100, ce qui représente les kopecs. Cette fonction renvoie un dictionnaire contenant le nom d'un joueur avec son score.

-premierTour() : reçoit en argument la liste des joueurs, le dictionnaire contenant le score, un dictionnaire contenant les joueurs encore en jeu, ainsi qu'un dictionnaire contenant les kopecs des joueurs, paramètre optionnel qui vaut *None* dans le cas où il n'y a pas de mises. Ainsi une première partie de la fonction traite du premier tour lorsqu'il n'y a aucune mise. Donc on initialise la pioche puis pour chaque joueur dans la liste des joueurs, le joueur pioche deux cartes, on ajoute leurs valeurs à son score, s'il dépasse 21, il n'est plus *en jeu* , s'il fait 21 alors c'est un BlackJack et de même il n'est plus *en jeu*, sinon, cela veut dire qu'il est bien en dessous de 21 et la fonction renvoie le dictionnaire contenant le score et la pioche.
Dans le cas des mises, le principe est le même or le joueur doit miser entre 0 et son nombre total de kopecs, ici, le joueur mise après avoir vu ses deux premières cartes. S'il réalise un BlackJack, alors il ne mise rien, cadeau de la maison, s'il dépasse 21, il y a 20 % de son total de kopecs qui est automatiquement mis en jeu, s'il n'est dans aucun de ces cas, alors il choisit sa mise. Chaque mises que le joueur met en jeu lui est retiré de son total de kopecs et toutes les mises sont ajoutées à une somme initialisée à zéro. La fonction renvoie alors le dictionnaire contenant les scores, la pioche, le dictionnaire contenant le total de kopecs des joueurs et la somme totale.

-gagnant() : reçoit en argument le dictionnaire des scores, celui du nombre de kopecs et la somme des mises renvoyées plus tôt. De même, la fonction travaille en deux partie, première partie où il n'y a pas de mises et seconde où il y en a.
Dans le premier cas, pour chaque joueurs, on vérifie si son score est le plus grand d'entre tous les joueurs et s'il est bien inférieur ou égal à 21. Si c'est le cas, alors c'est le gagnant et la fonction l'affiche. S'ils sont plusieurs à réaliser le même plus grand score, alors la fonction affiche les différents gagnants et renvoie rien.
Le deuxième cas se passe de la même sorte or chaque gagnant reçoit la somme des mises divisés par le nombre de gagnant. Si un joueur se retrouve sans kopecs, alors il est retiré du dictionnaire contenant les scores.

-continue() : est une fonction booléenne renvoyant *True* si le joueur désire continuer de jouer sinon *False*.

-tourJoueur() : reçoit en argument le nom d'un joueur, le dictionnaire contenant les scores, celui indiquant ceux qui sont encore en jeu, le tour indiquant à quel tour sommes-nous ainsi que la pioche. Fonctionne comme le

premier tour, or on demande au joueur s'il souhaite continuer ou non, si oui il pioche une seule carte et l'ajoute à son score. La résultante du résultat obtenu marche de la même sorte qu'au premier tour. De plus, lorsque le joueur réalise 21, il est automatiquement arrêté car ne pouvant faire mieux . La fonction ne renvoie rien et affiche ce qu'il faut afficher.

-tourComplet() : reçoit en argument le dico des scores, celui des joueurs en jeu, le tour actuel et la pioche. Pour chaque joueurs en jeu, le joueur réalise son tour et la fonction ne renvoie rien.

-partieFinie() : reçoit le dictionnaire des scores. Fonction booléenne qui renvoie *True* si le nombre de joueur en jeu est strictement inférieur à 1, sinon renvoie *False*.

-classement() : reçoit en argument le dico contenant les kopecs de chaque joueurs. Affiche les trois joueurs ayant le plus de Kopecs dans l'odre à la fin de d'une donne.

-partieComplete() : reçoit en argument la liste des joueurs, ainsi que la réponse indiquant si la partie se joue avec des mises ou non. La fonction initialise les scores (avec ou sans dico kopecs selon la réponse), les joueurs en jeu, réalise un premier tour puis tant que la partie n'est pas terminée (partieFinie renvoie *False)*, réalise un tour complet. Lorsque la partie est considérée comme terminée, on regarde qui est/sont le/les gagnant(s),puis s'il y avait des mises, on affiche le classement et la fonction renvoie le dico des kopecs (*None* si pas de mises).

-partieComplete_suite() : reçoit en argument la liste des joueurs et le dico kopecs. Cette fonction fonctionne de la même façon que la précédente, mais à la différence que celle-ci n'initialise pas le dico kopecs mais récupère l'ancien afin d'être à jour au niveau des kopecs de chaque joueurs. Elle renvoie la même chose.

-partir() : reçoit en argument le dico kopecs et la liste des joueurs. Cette fonction est seulement utilisée dans le cas de mises et demande si des joueurs veulent quitter la table, si c'est le cas, la fonction affiche avec combien de kopecs part le joueur et il est retiré du dico kopecs et de la liste des joueurs. La fonction renvoie dico kopecs et liste des joueurs.

Deuxième programme (avec croupier) :

(les fonctions similaires ne seront pas indiquées)

-initJoueur(): reçoit en argument le nombre de joueur, le style de jeu du croupier et le dico proba qui indique à quel point un joueur est joueur que nous indiquons avant, si son style de jeu est « plus ou moins joueur ». Cette fonction crée une liste des joueurs et un dictionnaire contenant le nom des joueurs joués par l'ordi avec leur style de jeu et ajoute leur proba dans le dico proba si le style est « plus ou moins joueur ». La fonction renvoie la liste des joueurs, le dico des joueurs joués par l'ordi et le dico proba.

-premierTour() : reçoit en argument la liste des joueurs, le dico des scores, celui des joueurs en jeu,celui des kopecs, la pioche, le dico des joueurs joués par des ordis et celui des proba. Il se passe de la même manière que le programme précédent pour les joueurs non joués par l'ordi avec des mises.
Dans le cas d'un joueur joué par l'ordi, le joueur (dont le croupier) décide s'il continue ou non grâce à d'autres fonctions que nous verrons après, puis s'il continue, il pioche deux cartes. Si le croupier tombe sur un « as », il y a un système d'assurance pour les autres joueurs car la deuxième carte du croupier reste cachée jusqu'au prochain tour, les joueurs décident s'ils prennent ou non l'assurance. Ensuite si le croupier dépasse 21 ou fait blackjack, la partie s'arrête. Système de mise est le même lorsque le score fait 21 ou le dépasse que dans l'autre programme. Dans l'autre cas, selon le style de jeu, le joueur a une mise prédéfinie ou passe par une fonction pour donner sa mise.
Détail : si le croupier tire une première carte qui laisse croire qu'il peut dépassé 21 avec sa deuxième carte qui est retournée, alors il la vérifie et indique s'il dépasse ou non.
Détail : si un joueur tombe sur « as », il fera le choix le plus judicieux entre la valeur de 1 et 11 grâce à une autre fonction.

-gagnant() : reçoit en argument dico des scores, celui des kopecs et la somme des mises. Ici le croupier n'est pas pris en compte comme pouvant être gagnant et les gagnants seront définis selon le score du croupier. Selon qu'il dépasse ou qu'il fasse un certain score. La fonction renvoie le dico kopecs.

-tourJoueur() : reçoit le nom d'un joueur, le dico scores, en jeu, le tour actuel, la pioche, dico joueur joué par l'ordi, la carte cachée du croupier et dico proba. Même principe que premier tour pour le fonctionnement avec le croupier qui indique la carte qui était face cachée. La fonction ne renvoie rien.

-clas() : reçoit dico kopecs et affiche les joueurs encore en jeu avec leurs kopecs actuels, ne renvoie rien.

Fonctions styles (booléens:True=joue, False=arrête):
-aleatoire() : choisis un nombre au hasard entre 0 et 1, si vaut 0, renvoie *True* sinon *False*.
-plus_ou_moins_joueur() : reçoit en argument a quel point le joueur est joueur et renvoie *True* ou *False* selon la proba. (proba vaut 1, 0.5 ou 0).
-strategie1() : reçoit le score d'un joueur et selon le score du joueur, il a plus ou moins de chance de renvoyer *True* ou *False*. (peu de chance qu'il continue s'il est entre 17 et 19). Il y a encore une part de hasard.
-strategie2() : reçoit la pioche, le score d'un joueur et un certain pourcentage qui vaut 60 par défaut. Si le pourcentage de carte pouvant dépassé 21 en étant associé au score du joueur dépasse le pourcentage défini, alors la fonction renvoie *False*, sinon *True*.
-strategie3() : reçoit dico scores, pioche, le score d'un joueur et le nom d'un joueur. Si plus d'un joueur dépasse le score du joueur ou qu'un joueur se retrouve avec un score entre 14 et 17 et qu'il est à 3 points ou moins de moins que le joueur, alors la fonction renvoie True sinon False.
-complet() : reçoit dico scores, pioche, score d'un joueur. C'est un mélange des fonctions précédentes, la fonction joue plutôt la sureté.

La fonction style_de_jeu() contient toutes les fonctions de styles de jeu et choisis la bonne fonction selon le style de jeu du joueur reçu en argument, renvoie *True* ou *False*.

Fonctions mises :
-premiere_mise() : reçoit dico kopecs et le nom d'un joueur. La fonction prend la moitié des kopecs du joueur, et selon le chiffre obtenue obtenu au hasard, la moitié des kopecs sont de nouveau divisés et sont renvoyés par la fonction, ça représente alors la mise du joueur.
-deuxieme_mise() : reçoit en argument dico kopecs, des scores et le nom d'un joueur. Selon le score du joueur, on divise ses kopecs par un certain nombre puis si le nombre de joueurs ayant un score entre 11 et 15 dépasse 1, alors on divise de nouveau par 2 et la fonction renvoie le résultat.
troisieme_mise() : reçoit dico kopecs, en jeu et le nom d'un joueur. On multiplie par 0.2 les kopecs du joueur, puis on utilise la suite de Syracuse avec u0 vaut le résultat calculé plus tôt et le nombre d'itération dépend du nombre de joueur en jeu. De plus, si le résultat après la suite est plus élevé que le résultat calculé au début, on divise par 2 et on renvoie le résultat, sinon on renvoie le résultat après la suite.

La fonction hasard() choisis un style de mise au hasard entre les trois façons de miser et renvoie le résultat, ainsi le joueur peut miser de trois façons différentes mais sans choisir laquelle prendre. De plus, s'il reste 10 kopecs ou moins au joueur, alors il doit tous les parier.

Déroulement des programmes :

premier programme :

Le programme demande dans un premier temps si la partie se joue avec des mises ou non. Suite à cela, il faut indiquer combien de joueurs participent. On peut alors initialiser la pioche et la liste des joueurs. Puis nous appelons la fonction partieComplete pour jouer une partie complète selon la réponse donnée au début. La partie complète se déroule comme indiquée dans les fonctions précédemment.
Ensuite, si la réponse était oui, on reconstruit la liste des joueurs selon les joueurs qui ont encore plus de 0 kopec et si la longueur du dico kopecs est strictement supérieur à 1, alors le programme demande si on souhaite refaire une donne, puis demande si des joueurs veulent quitter la table.
A la suite de ça, tant que le nombre de joueurs est strictement supérieur à 1 et que l'on souhaite continuer de jouer, nous rejouons une partie complète, différente si mise ou non.
Lorsque une des deux conditions n'est plus remplie, si il y avait des mises, le programme affiche le vainqueur en admettant qu'il ne restait plus que ce joueur, et affiche un message de fin dans tous les cas.

Deuxième programme :

Le programme demande directement avec quel type de croupier on souhaite jouer, si l'on choisit « plus ou moins joueur », il nous faudra ajouter sa probabilité de rejouer ou non. Cette probabilité sera ajoutée dans un dictionnaire avec le nom de 'croupier'. Suite à ça, on indique le nombre de joueurs qui vont participer et de même que pour l'autre programme, on initialise la pioche, la liste des joueurs avec le dictionnaire contenant les joueurs joués par des ordis contenant aussi le croupier et le dictionnaire contenant les probabilités de chaque joueurs si son style de jeu est « plus ou moins joueur ». Puis on réalise une partie complète qui se déroule comme décrit dans les fonctions précédentes.

Suite à cela, le programme demande si des joueurs souhaitent quitter la partie entre les donnes. Puis, tant que le croupier a au moins 1 kopec et que la liste des joueurs comporte plus d'un joueur alors une nouvelle partie complète est jouée avec cette fois la fonction partiecomplete_suite.

Lorsque l'on sort de la boucle, si cela est parce que le croupier n'a plus de kopec, alors le programme l'indique, si c'est alors parce que le nombre de joueurs n'est plus suffisant, le programme l'indique aussi en affichant une phrase. Il peut afficher les deux si nous sommes dans les deux cas.

Pour finir, le programme affiche un message de fin.

Spécificités :

Ce programme de BlackJack contient des spécificités qui ont été des choix de notre part. On en retrouve notamment autour du croupier, effectivement le croupier est le coeur même du jeu et le déroulement de la partie dépend beaucoup de lui. S'il décide de ne pas piocher de carte dès le début, alors la donne se termine, de même s'il fait un BlackJack. S'il décide de ne pas piocher de règle général, alors la donne se termine, ainsi il peut empêcher des joueurs d'atteindre un score suffisamment grand pour gagner comme il peut donner facilement des kopecs.

Au niveau des mises, on remarque que ceux qui font un BlackJack ne misent pas, pourtant ils peuvent gagner la somme des mises des autres joueurs, ainsi il est sur de remporter des kopecs si au moins un joueur dont le croupier n'a pas fait de BlackJack. De plus, lorsque l'on dépasse 21 dès le début, 20 % des kopecs sont automatiquement misés, ce qui pénalise le joueur car il est presque sur de les perdre.

Encore dans les mises, on peut observer qu'un joueur qui mise une petite somme sortira vainqueur si un autre joueur à misé plus que lui et qu'ils sont tous les deux gagnants car le total divisé par deux sera plus important que sa mise de départ, d'où le fait que les mises peuvent être assez radine du côté des ordi.

En ce qui concerne les fonctions de style de jeu et de mises, il y a bon nombre de hasard qui rajoute une incertitude durant la partie, surtout au niveau de la mise, cependant, dans les styles de jeu plus poussés, il n'y a pas de hasard, mais la fonction est faite de façon à ce que le joueur joue de façon sécure quitte à s'arrêter tôt. Une suite a été ajouté dans un style de mise pour ajouter un aspect maths au programme.

Une autre spécificité concerne la fonction qui demande au joueur de partir, on laisse ici la personne derrière l'ordi choisir si l'ordi continue de jouer ou non, ça rajoute de l'interaction puis comme cela, nous pouvons faire durer une partie très longtemps.

Pour poursuivre sur l'interaction, plusieurs fois dans le programme il suffit seulement d'appuyer sur 'entrer', cela permet à la personne de suivre les textes, de ne pas être perdu dans les tours et d'interagir avec la partie qui se déroule, se sentir concerné.

Encore d'autres spécificités à découvrir...

En ce qui concerne les défauts, il faudrait continuer de tester le programme, notamment le cas de l'assurance qui est compliqué à tester. Au niveau des « as » il y avait quelques soucis de décisions et il se pouvait que le croupier dépasse 21 alors que sa première carte était inférieur à 8 mais nous avons retravaillé ces soucis. Puis il est difficile d'explorer toutes les possibilités.

Organisation :

Nous nous sommes occupés tous les deux du premier programme afin de comprendre son fonctionnement, ses fonctions avec ses variables et ainsi pouvoir se partager le travail pour la suite. Donc pour la suite, l'un s'est plus occupé du programme avec intelligence artificielle et l'autre s'est plus occupé de l'interface graphique.
PS : si on s'y était pris à temps, il aurait été incroyable