TP 8 et 9 : projet

1. Représenter un jeu de cartes

Dans un jeu de cartes, une carte est caractérisée par sa couleur (carreau, cœur, pique, trèfle) et par sa hauteur (as, valeurs de 2 à 10, valet, dame, roi). Dans ce projet, une carte sera représentée par une liste de deux valeurs dont la première correspond à sa hauteur et la seconde à sa couleur. Ainsi, la carte « As de pique » correspondra à la liste `'(1 3)`.

2. Quelques fonctions à écrire avant de jouer

Passer d'une représentation pour la machine à une représentation pour l'utilisateur

1. Écrire la fonction `carte_affichable` qui affiche la carte correspondant à une liste de deux valeurs donnée en paramètre. Le premier élément de cette liste correspond à la hauteur de la carte (1 pour l'as, 2 à 10 pour les cartes correspondantes, 11 pour le valet, 12 pour la dame, 13 pour le roi) et le second à sa couleur (1 pour carreau, 2 pour coeur, 3 pour pique, et 4 pour trèfle).

```
(carte_affichable '(12 3)) → (Dame Pique)
(carte_affichable '(5 1)) → (5 Carreau)
```

2. Écrire la fonction `liste_cartes_affichables` qui affiche une liste de cartes à partir d'une liste de couples de valeurs.

```
(liste_cartes_affichables '((12 3) (5 1))) →
        ((Dame Pique) (5 Carreau))
```

N'hésitez pas ensuite à utiliser ces deux fonctions pour rendre plus lisibles les cartes affichées à l'écran.

Créer un jeu de 52 cartes

On souhaite créer un jeu de 52 cartes défini comme une liste de cartes. Pour cela, on va définir une fonction intermédiaire qui construit une liste de cartes suivant une première carte donnée, et l'on utilisera ensuite cette fonction pour créer le jeu de cartes à partir de l'as de carreau.

3. Écrire la fonction `cartes_suivantes` qui construit la liste des cartes suivantes d'une carte donnée en paramètre. Cette liste doit respecter l'ordre des couleurs (carreau, cœur, pique, trèfle) ; ainsi la dernière carte correspond au roi de trèfle.

```
(cartes_suivantes '(13 4)) → ()
(cartes_suivantes '(8 4)) → ((9 4) (10 4) (11 4) (12 4) (13 4))
(cartes_suivantes '(12 3)) → ((13 3) (1 4) (2 4) (3 4) (4 4) (5 4)
        (6 4) (7 4) (8 4) (9 4) (10 4) (11 4) (12 4) (13 4))
```

4. Écrire la fonction `creer_52_cartes` qui crée un jeu de 52 cartes défini comme une liste de couples de valeurs.

```
(creer_52_cartes) → ((1 1) (2 1) … (12 4) (13 4))
```

Mélanger un jeu de cartes

Pour écrire une fonction qui mélange un jeu de cartes, on va procéder en plusieurs étapes. Pour mélanger, il faut pouvoir tirer une carte au hasard pour en faire la première carte du nouveau jeu mélangé. Pour cela, on choisira au hasard l'indice *i* de la carte à tirer, et l'on extraira du jeu la carte d'indice *i*. Voici donc trois fonctions qui permettent de programmer cette méthode.

Université Lyon1 – Licence Sciences, Technologies, Santé UE LIF3

5. Écrire la fonction `extraire_ieme_carte` qui extrait la i^{ème} carte d'une liste de cartes donnée en paramètre. Cette fonction renvoie une liste constituée de la carte extraite et de la liste des cartes restantes.

```
(extraire_ieme_carte '((1 1) (12 3) (14 4)) 2) → ((12 3) ((1 1) (14 4)))
```

6. Écrire la fonction `tirer_une_carte` qui tire une carte au hasard parmi les cartes d'une liste de cartes donnée en paramètre. Cette fonction renvoie une liste constituée de la carte tirée, et de la liste des cartes restantes après le tirage.

```
(tirer_une_carte '((1 2) (13 4) (5 3) (1 1) (6 3))) →
        ((6 3) ((1 2) (13 4) (5 3) (1 1)))
```

Pour écrire cette fonction, utilisez la fonction `(random n)` qui renvoie un nombre au hasard compris entre 0 (inclus) et n (exclus), ainsi que la fonction précédente.

7. Écrire la fonction `melanger_jeu_cartes` qui permet de mélanger les cartes d'une liste de cartes donnée en paramètre. Cette fonction renvoie la liste des cartes mélangées. À noter que cette fonction utilise la fonction `tirer_une_carte`.

```
(melanger_jeu_cartes '((10 1) (4 4) (5 2)) → ((10 1) (5 2) (4 4))
```

Piocher des cartes et se défausser

Piocher des cartes est différent de tirer des cartes au hasard : la pioche est un paquet de cartes dont on prend les cartes de dessus lorsqu'on pioche. Lorsque qu'on se défausse, on remet les cartes en dessous de la pioche.

8. Écrire la fonction `piocher_n_cartes` qui permet de piocher les n premières cartes d'une liste de cartes donnée en paramètre. Cette fonction renvoie une liste constituée de la liste des cartes piochées, et de la liste des cartes restantes dans la pioche.

```
(piocher_n_cartes '((13 4) (5 3) (1 1) (6 3)) 2) →
        (((13 4) (5 3)) ((1 1) (6 3)))
```

9. Écrire la fonction `defausser_cartes` qui prend deux listes de cartes en paramètres (une liste de cartes correspondant à la pioche et une liste des cartes à défausser) et qui retourne la liste de ces cartes dont les cartes à défausser se trouvent à la fin.

```
(defausser_cartes '((1 1) (1 2)) '((3 4) (4 3))) → ((1 1) (1 2) (3 4) (4 3))
```

3. Le Black Jack

Une partie de blackjack va opposer le joueur à la banque. La banque commence par tirer une carte et va ensuite en donner deux au joueur. Le but de ce jeu consiste à approcher ou faire le chiffre 21 sans le dépasser. La valeur des cartes est établie comme suit : les cartes de 2 à 10 conservent leurs valeurs, les figures valent 10 et l'as vaut 1.

Le joueur peut demander autant de cartes qu'il le souhaite. S'il dépasse 21, il perd. S'il s'arrête avant, la banque tire des cartes jusqu'à dépasser 17 pour ne pas prendre de risque. Si la banque fait moins que le joueur, ou qu'elle dépasse 21, le joueur gagne. S'il fait moins que la banque, le joueur perd. Sinon, il y a égalité.

Dans ce projet, un jeu de blackjack sera considéré comme une liste de 3 sous-listes : la liste des cartes de la banque, la liste des cartes du joueur et la liste des cartes restantes dans la pioche.

10. La fonction `creer_blackjack` prépare le jeu du blackjack en initialisant la liste correspondante au jeu. Cette fonction retourne ainsi une liste constituée de la liste des cartes du

joueur (vide au départ), de la liste des cartes de la banque (vide au départ), et de la pioche (constituée au départ des 52 cartes mélangées du jeu).

```
(define creer_blackjack
   (lambda ()(list '() '() melanger_jeu_cartes (creer_52_cartes)))))
```

Les fonctions `cartes_joueur`, `cartes_banque` et `cartes_pioche` permettent d'accéder respectivement à la liste des cartes du joueur, de la banque et de la pioche à partir d'une liste définissant le jeu du blackjack. Exemple d'appel : `(cartes_joueur (creer_blackjack))`.

```
(define cartes_joueur      (define cartes_banque      (define cartes_pioche
   (lambda (b)                (lambda (b)                (lambda (b)
     (car b)))                  (cadr b)))                 (caddr b))
```

11. Écrire la fonction `banque_pioche_n` qui met à jour le jeu du blackjack après une pioche de n cartes de la part de la banque.

12. Écrire la fonction `joueur_pioche_n` qui met à jour le jeu du blackjack après une pioche de n carte de la part du joueur.

13. Écrire la fonction `total_cartes` qui retourne les points correspondant à une liste de cartes donnée en paramètre. Pour rappel, un as compte pour un point ; les figures comptent pour dix points ; les cartes de 2 à 10 conservent leurs valeurs.

```
(total_cartes '((1 3) (13 4) (5 4))) → 16
```

14. Écrire la fonction `faire_jouer_banque` qui fait jouer la banque jusqu'à qu'elle atteigne au moins un total de 17 points. Cette fonction renvoie la liste du blackjack mise à jour.

15. La fonction `afficher_jeu` permet d'afficher l'état du jeu.

```
(define afficher_jeu
  (lambda (b) ; jeu de blackjack (liste de 3 sous-listes)
    (begin (display "------ Jeu de la banque -------")
           (newline) (display "Cartes :")
           (display  (liste_cartes_affichables (cartes_banque b)))
           (newline) (display "Score : ")
           (display  (total_cartes (cartes_banque b)))
           (newline) (display "------ Jeu du joueur ---------")
           (newline) (display "Cartes :")
           (display  (liste_cartes_affichables (cartes_joueur b)))
           (newline) (display "Score : ")
           (display  (total_cartes (cartes_joueur b)))
           (newline) (display "----------------------------") (newline))))

(afficher_jeu (creer_blackjack)) ; Premier appel
```

16. Écrire la fonction `faire_jouer_joueur` qui retourne le jeu du blackjack mis à jour après le jeu du joueur (après 0, une ou plusieurs pioches). Si le joueur atteint 21 points, il est inutile de continuer à jouer. Sinon cette fonction doit donner le choix au joueur de rejouer ou non (grâce à la fonction `read`).

17. La fonction `tout_defausser` permet de défausser toutes les cartes afin de recommencer à jouer. Cette fonction renvoie ainsi le jeu du blackjack dont les cartes de la banque et celles du joueur ont été mises dans la pioche.

```
(define tout_defausser
      (lambda (b)
            (list '() '()
                  (defausser_cartes
                        (defausser_cartes (cartes_pioche b) (cartes_banque b))
                        (cartes_joueur b)))))
```


18. La fonction `tour_de_jeu` simule un tour de jeu : la banque tire une carte, puis le joueur tire 2 cartes, et continue à jouer s'il le souhaite. Quand le joueur s'arrête, ses points sont comptabilisés. Si le joueur a dépassé les 21 points, la fonction affiche le fait qu'il a perdu. Sinon, la banque continue à jouer. À la fin du jeu de la banque, son nombre de points est comptabilisé. Puis la fonction affiche le jeu et un message précisant qui est le gagnant.

19. Les fonctions suivantes permettent de jouer plusieurs tours au blackjack.

```
(define commencer_blackjack
  (lambda ()
    (partie_blackjack (creer_blackjack))))

(define partie_blackjack
 (lambda (b)
   (let* ((b (tour_de_jeu b)))
     (begin
           (display "Voulez-vous faire un autre tour? (o/n)")
           (newline)
          (if (eq? 'o (read))
              (partie_blackjack (tout_defausser b))
              (display "Au revoir... "))))))
```

20. Question bonus (avant de faire cette question, faites la partie 4 et assurez-vous d'avoir un programme correctement commenté) : Dans la règle complète du blackjack, chaque As peut prendre au choix la valeur 1 ou 11. Dans la pratique, au maximum un As peut prendre la valeur 11 (sinon on a un score d'au moins 22), et la valeur d'un As est choisie pour maximiser le score sans dépasser 21. Exemple: Si un joueur a un As et un 4, il a (au choix) 5 points ou 15: on privilégiera la valeur 11 pour l'As pour profiter d'un meilleur score. Si le joueur décide de tirer une autre carte et qu'il obtient un 9, l'As vaut alors 1 pour atteindre un total de 14 (au lieu de 24 s'il valait 11). Proposez une fonction `total_cartes2` qui prend en compte cette nouvelle règle dans le comptage des points d'un ensemble de cartes.

4. La bataille

Une partie de bataille va opposer le joueur à la machine. Un jeu de 52 cartes est mélangé et partagé en deux paquets de 26 cartes : l'un pour la machine, l'autre pour le joueur. À chaque tour, le joueur et la machine piochent une carte dans leur paquet. Celui qui pioche la carte de hauteur la plus élevée emporte le pli, c'est-à-dire met les deux cartes en dessous de son paquet. Pour simplifier le problème, on décide que l'as vaut 1 et qu'en cas d'égalité, chacun remet sa carte sous son paquet. Le jeu se poursuit jusqu'à ce qu'un joueur n'ait plus de cartes. L'autre joueur est alors le gagnant.

Dans cette partie du projet, l'on ne vous demande pas de programmer. Vous devez juste décrire comment vous envisagez de décomposer le problème, quelles sont les fonctions des parties 2 et 3 que vous pourriez réutiliser et comment, quelles sont celles que vous devriez adapter, et quelles nouvelles fonctions il faudrait écrire (avec leur spécification).

5. Évaluation du projet

Lors de la dernière séance de TP (mardi 3 mai 2016), vous devrez faire une démonstration de votre programme à votre enseignant. Il vous demandera notamment d'expliquer les fonctions que vous avez implantées. Vous déposerez sur TOMUSS le **code source qui doit être commenté**, et qui doit comporter des **appels pertinents aux différentes fonctions** afin de les tester.