Projet de Programmation Fonctionnelle 2017-2018 : Calculette du Texas Hold'em

Table des matières

1 Introduction

Le but de ce projet est de programmer une calculette pour la variante "*Texas Hold'em*" du Poker [1]. Le programme doit permettre à un joueur de calculer ses chances de victoire à chaque phase de la partie en fonction de ses cartes et des cartes communes visibles.

Le travail minimal que nous vous demandons est d'implémenter *toutes* les fonctionnalités décrites à la Section 3 et *au moins une* parmi toutes celles décrites à la Section 4.

2 Les règles du Texas Hold'em

Nous adopterons les règles comme décrites par Wikipedia dans le lien ci-dessous :

`https://fr.wikipedia.org/wiki/Texas_hold'em`

1. Attention ! Le Poker est un jeu d'argent pouvant induire une forte dépendance. Ce jeu, combinatoirement non trivial, n'est ici qu'un simple support pour un projet de programmation : nous n'incitons personne à y jouer, et encore moins pour de l'argent. Pour en savoir plus sur l'addiction au jeu : `http://www.evalujeu.fr/static/pages/`

Le Texas Hold'em est une variante de Poker qui se joue à au moins deux joueurs, jusqu'à dix joueurs. Nous nous occuperons ici principalement de la version à deux joueurs.

Le jeu nécessite un jeu ordinaire de 52 cartes. Chaque partie se déroule en cinq phases [2].

1. Chaque joueur reçoit d'abord deux cartes visibles par lui seul (*pre-flop*).
2. Trois cartes sont ensuite disposées faces visibles sur la table (*flop*).
3. Une quatrième carte est placée face visible sur la table (*turn*).
4. Une cinquième est placée face visible sur la table (*river*).
5. Enfin, les joueurs étalent leurs cartes (*showdown*).

On détermine alors pour chaque joueur la plus forte *combinaison à cinq cartes* composable à partir de ses deux cartes et des cinq cartes communes. Un joueur est déclaré gagnant si cette combinaison est strictement plus forte que toutes celles des autres joueurs. En cas d'égalité des combinaisons les plus fortes, la partie est déclarée nulle.

Nous précisons à la section suivante le détail des combinaisons ainsi que leur ordre, *ie.* les règles qui déterminent dans quel cas une combinaison est supérieure à une autre. Le strict respect de ces règles est nécessaire pour implémenter correctement les fonctions demandées dans ce projet.

2.1 Comparaison des combinaisons

Chaque carte a une *couleur* et un *rang* :

— la couleur peut être
Pique, *Cœur*, *Carreau* ou *Trèfle* ($\spadesuit, \heartsuit, \diamond, \clubsuit$),

— le *rang* peut être, par ordre de valeur décroissante,
As, *Roi*, *Dame*, *Valet*, *Dix*, *Neuf*, *Huit*, *Sept*, *Six*, *Cinq*, *Quatre*, *Trois*, *Deux* ($A, R, D, V, 10, \ldots, 2$).

Noter qu'il n'y a pas au poker d'ordre sur les couleurs, seulement sur les rangs : quelles que soient les couleurs de cartes, un As est plus fort qu'un Roi, qui est plus fort qu'une Dame, etc.

Une *combinaison* est formée d'exactement cinq cartes. Les combinaisons sont ordonnées par niveaux (Section 2.1.1), toute combinaison étant battue par une combinaison de niveau supérieur. Lorsque deux joueurs ont des combinaisons de même niveau, des règles supplémentaires permettent de déterminer si l'un des deux est gagnant (Section 2.1.2).

2.1.1 Niveaux des combinaisons

Voici la liste des combinaisons, par ordre de niveau décroissant.

Quinte flush (ou *suite colorée*). Cette combinaison est formée de cinq cartes de rangs consécutifs et de même couleur. Pour la décrire, on indique le rang de sa plus haute carte. Exemple, une quinte flush au Roi :

$$R\diamond, D\diamond, V\diamond, 10\diamond, 9\diamond$$

Noter que $3\diamond, 2\diamond, A\diamond, R\diamond, D\diamond$ n'est pas une quinte flush.

Carré (ou *poker*). Cette combinaison est formé de quatre cartes de même rang et d'une carte quelconque. Pour la décrire, on indique le rang commun des quatre cartes. Exemple, un carré de Rois :

$$R\spadesuit, R\heartsuit, R\diamond, R\clubsuit, 10\clubsuit$$

2. La description du jeu que nous donnons ici ne tient pas compte du détail du mécanisme d'enchères, avant la phase initiale (*blind*) et entre les phases, ni de l'abandon potentiel d'un joueur. Les détails omis sont sans importance pour la partie commune de ce projet. Vous trouverez une description plus précise de cet aspect du jeu sur l'article de Wikipédia.

Full (ou *main pleine*). Cette combinaison est formée de trois cartes de même rang, et de deux autres cartes de même rang. Pour la décrire, on indique d'abord le rang des trois premières, puis celui de deux autres. Example, voici un full aux Rois par les Trois :

$$R\heartsuit, R\diamond, R\clubsuit, 3\spadesuit, 3\clubsuit$$

Couleur (ou *flush*) Cette combinaison est formée de cinq cartes de couleur identique mais ne formant pas une quinte flush (donc, de rangs non consécutifs). Pour la décrire, on indique le rang de sa plus haute carte. Exemple, une couleur au Roi :

$$R\diamond, 7\diamond, 4\diamond, 3\diamond, 2\diamond$$

Suite (ou *quinte*) Cette combinaison est formée de cinq cartes de rangs consécutifs mais ne formant pas une quinte flush (elle contient donc au moins deux couleurs distinctes). Pour la décrire, on indique le rang de sa plus haute carte. Example, voici une suite au Huit :

$$8\diamond, 7\spadesuit, 6\clubsuit, 5\clubsuit, 4\heartsuit$$

Noter que $3\diamond, 2\spadesuit, A\clubsuit, R\clubsuit, D\heartsuit$ n'est pas une suite.

Brelan (ou *three*). Cette combinaison est formée de trois cartes de même rang, et de deux autres cartes ne formant avec les trois premières ni un carré ni un full (ces deux cartes sont donc de rangs distincts, et distincts du rang des trois premières). Pour la décrire, on indique le rang commun des trois premières cartes. Exemple, un brelan de Rois :

$$R\diamond, R\spadesuit, R\clubsuit, 5\clubsuit, 4\heartsuit$$

Double paire Cette combinaison est formée de deux cartes de même rang, deux autres cartes de même rang, ainsi qu'une cinquième carte, l'ensemble ne formant ni un carré ni un full. Pour la décrire, on indique parmi les deux rangs communs le rang le plus élevé, puis le moins élevé. Exemple, une double paire de Rois par les Huit :

$$R\diamond, R\spadesuit, 8\clubsuit, 8\heartsuit, 4\clubsuit$$

Paire Cette combinaison est formée de deux cartes de même rang et de trois autres cartes, l'ensemble ne formant ni une double paire, ni un brelan, ni un full, ni un carré. Pour la décrire, on donne le rang commun des deux cartes. Exemple, une paire de Huit :

$$8\clubsuit, 8\heartsuit, R\diamond, D\spadesuit, 4\clubsuit$$

Carte haute Cette combinaison celle formée par tous les ensembles de cinq cartes ne formant aucune des combinaisons précédentes. Pour la décrire, on indique le rang de la carte de plus haut rang. Exemple, une carte haute au Roi.

$$R\diamond, D\spadesuit, 10\clubsuit, 8\heartsuit, 4\clubsuit$$

2.1.2 Comparaison des combinaisons de même niveau

Lorsque deux joueurs ont des combinaisons de même niveau, les règles suivantes permettent de déterminer si l'un des deux est gagnant. Les couleurs n'intervenant pas dans cette comparaison, il peut y avoir égalité. Noter que le Texas Hold'em se jouant avec des cartes communes, rien n'empêche une même carte d'apparaître dans les combinaisons de deux joueurs distincts.

Paire, Brelan, Carré. On compare les rangs respectifs des 2, 3 ou 4 cartes de même rang dans chaque combinaison. S'ils sont identiques, on compare les cartes *restantes* par *ordre lexicographique inverse* : d'abord les deux cartes les plus fortes puis, si elles sont identiques, les deux cartes suivantes les plus fortes, etc. Par exemple :

$R\diamond, R\spadesuit, 8\heartsuit, 5\clubsuit, 3\clubsuit$	bat	$8\diamond, 8\spadesuit, 10\clubsuit, 6\heartsuit, 4\clubsuit$	(le Roi bat le 8)
$R\diamond, R\spadesuit, 9\heartsuit, 5\clubsuit, 2\clubsuit$	bat	$R\clubsuit, R\heartsuit, 9\clubsuit, 4\heartsuit, 3\clubsuit$	(le 5 bat le 4)
$R\diamond, R\spadesuit, 9\heartsuit, 5\clubsuit, 2\clubsuit$	est à égalité avec	$R\clubsuit, R\heartsuit, 9\clubsuit, 5\heartsuit, 2\clubsuit$	

Suite, Quinte flush. On compare les rangs des cartes de rang maximal dans chaque combinaison. Par exemple,

$R\spadesuit, D\spadesuit, V\spadesuit, 10\spadesuit, 9\spadesuit$ bat $V\spadesuit, 10\spadesuit, 9\spadesuit, 8\spadesuit, 7\spadesuit$ (le Roi bat le Valet)

Carte haute, Couleur. On compare les cartes de chaque combinaison par ordre lexicographique inverse.

$R\spadesuit, 10\spadesuit, 8\spadesuit, 5\spadesuit, 2\spadesuit$ bat $R\spadesuit, 10\spadesuit, 8\spadesuit, 4\spadesuit, 3\spadesuit$ (le 5 bat le 4)

Full. On compare les rangs respectifs des 3 cartes de même rang dans chaque combinaison. En cas d'égalité, on compare les rangs respectifs des 2 autres cartes de même rang dans chaque combinaison.

$R\diamond, R\spadesuit, R\clubsuit, 2\heartsuit, 2\clubsuit$ bat $D\diamond, D\spadesuit, D\clubsuit, V\heartsuit, V\clubsuit$ (le Roi bat la Dame)

Double paire. On compare d'abord les rangs des couples de cartes de rang commun maximal dans chaque combinaison. En cas d'égalité, on compare les rangs des couples suivants. En cas d'égalité encore, on compare les rangs des cartes restantes dans chaque combinaison.

$A\diamond, A\spadesuit, 2\clubsuit, 2\heartsuit, 8\clubsuit$	bat	$R\diamond, R\spadesuit, D\clubsuit, D\heartsuit, 1\clubsuit$	(l'As bat le Roi)
$A\spadesuit, A\diamond, D\clubsuit, D\heartsuit, 2\clubsuit$	bat	$A\diamond, A\spadesuit, 2\clubsuit, 2\heartsuit, D\clubsuit$	(la Dame bat le 2)
$A\diamond, A\spadesuit, D\clubsuit, D\heartsuit, 3\clubsuit$	bat	$A\diamond, A\spadesuit, D\clubsuit, D\diamond, 2\clubsuit$	(le 3 bat le 2)

3 Implémentation de la partie commune

3.1 Les types de bases

Définir les types de données suivants :

— `carte` : l'association d'un rang et d'une couleur.

— `donne` : représentant les deux cartes données à un joueur.

— `table` : représentant les 3, 4 ou 5 cartes présentes sur la table.

— `comb` : représentant les combinaisons.

Les valeurs du type `comb` doivent encapsuler toutes, et seulement les informations permettant de comparer efficacement deux combinaisons suivant les règles détaillées en Section 2.1.

Par exemple, la seule information à mémoriser pour connaître la valeur d'une "Quinte flush", outre le fait qu'il s'agit d'une quinte flush, est le rang de sa plus haute carte (et pas sa couleur). En revanche, la force d'une "Carte haute" ne peut être déterminée sans connaître la suite décroissante de rangs de chacune de ses cartes.

3.2 Fonctions de comparaisons

Vous devrez écrire une fonction

```
compare_hands : donne -> donne -> table -> int
```

qui, étant données les donnes de deux joueurs et *cinq* cartes sur la table, renvoie 1 si le premier joueur est gagnant, -1 si le deuxième joueur est gagnant, et 0 s'il y a égalité. À cette fin, nous vous suggérons de définir les fonctions auxiliaires suivantes :

— `compare_comb : comb -> comb -> int`
telle que `compare_comb c1 c2` s'évalue en `1` si la combinaison `c1` est strictement plus forte de `c2`, `-1` si `c2` est strictement plus forte de `c1`, et `0` si les deux combinaisons sont à égalité.

— `compute_comb : donne -> table -> comb list`
telle que `compute_comb d t` s'évalue en la liste de toutes les combinaisons possibles pouvant être construites avec une donne `d` et cinq cartes sur la table `t` [3].

La fonction `compare_hands` peut alors être implémentée comme suit : (i) on génère (en faisant appel à la fonction `compute_comb`) les listes des combinaisons possibles associées à chaque donne, pour une table donnée ; (ii) en utilisant `compare_comb`, on extrait de chaque liste une combinaison maximale ; (iii) enfin, on rend en sortie le résultat de la comparaison de ces deux combinaisons par `compare_comb`.

3.3 Calcul de la probabilité de victoire

Vous devrez écrire les deux fonctions suivantes :

```
proba_double : donne -> donne -> table -> float * float
proba_simple : donne -> table -> float
```

telle que :

— `proba_double d1 d2 t` renvoie un couple de nombres réels compris entre 0 et 1 indiquant les probabilités de victoire de deux joueurs en fonction de leurs donnes `d1` et `d2` et des cartes présentes sur la table `t` (0 signifiant qu'un joueur n'a aucune chance de gagner, 1 qu'il a la certitude de gagner) [4].

— `proba_simple d1 t` renvoie un nombre réel compris entre 0 et 1 donnant la probabilité de victoire d'un joueur sans que soit connue la donne de son adversaire, en fonction de sa donne `d1` et des cartes présentes sur la table `t`.

On supposera que la table contient déjà au moins trois cartes [5], et bien sûr, qu'aucune carte n'apparaît à la fois sur la table et dans une donne, ni dans deux donnes distinctes.

Une implémentation possible de la première fonction consiste à générer, étant donné deux donnes `d1` et `d2` et un ensemble de cartes présentes sur la table `t`, une liste de toutes les fins de parties compatibles avec ces donnes et les cartes présentes, *ie.* une liste de triplets de la forme `(d1, d2, tt)`, où `tt` est l'une des complétions à cinq cartes possibles de `t`. La probabilité de victoire d'un joueur sera obtenue en divisant le nombre de triplets gagnants pour ce joueur par la longueur globale de la liste.

La seconde fonction peut être implémentée avec la même méthode, en calculant cette fois tous les triplets de la forme `(d1, dd, tt)` où `dd` est l'une des donnes possibles pour l'adversaire, et `tt` l'une des complétions à cinq cartes possibles de `t`. Là encore, la probabilité de victoire du joueur de donne `d1` sera obtenue en divisant le nombre de triplets gagnants pour ce joueur par la longueur globale de la liste.

3. Vous pouvez aussi tenter d'extraire *directement* d'une donne et de cinq cartes la plus grande combinaison possible, c'est-à-dire d'écrire une fonction `compute_comb_max : donne -> table -> comb` sans construire une telle liste.

4. La somme de ces probabilités peut ne pas être égale à 1, à cause des possibilités d'égalité.

5. On peut calculer cette probabilité avant le flop, mais cela suppose des calculs assez lourds. Cette information peut aussi être pre-calculée – mais ceci n'est pas le problème que nous vous demandons de résoudre.

Pour vous permettre de tester la correction de votre implémentation, nous vous fournirons sur `Moodle` une batterie de tests avec des probabilités de victoire calculées par nos soins. Vous pouvez aussi comparer vos résultats avec des calculettes disponibles sur internet (pourvu qu'elles soient correctes !), comme par example : `https://fr.pokernews.com/poker-tools/poker-odds-calculator.htm`.

Nous apprécierons particulièrement tout effort pour améliorer l'efficacité de la méthode ci-dessus. N'hésitez donc pas à mettre en valeur, pendant la soutenance, vos choix d'implémentation.

3.4 Lecture d'un fichier

Afin de nous permettre de tester vos fonctions, vous devrez écrire et compiler un exécutable `compute` lançable depuis le terminal, et prenant en argument le nom d'un fichier décrivant une configuration de Texas Hold'em. Le fichier pourra contenir une configuration complète ou incomplète – une ou deux donnes, et 3, 4 ou 5 cartes sur la table.

— Si la configuration est complète, le programme devra préciser si l'un des deux est gagnant (voir Exemple 1).

— Si la configuration contient deux donnes mais une table incomplète, le programme devra afficher à l'écran la probabilité de victoire de chaque joueur (voir Exemple 2).

— Enfin, si la configuration ne contient qu'une seule donne, le programme devra afficher la probabilité de victoire du joueur recevant cette donne (voir Exemple 3).

Plus précisément, le fichier texte sera dans le format standard suivant (à respecter strictement pour nous permettre d'effectuer les tests).

— Chaque carte sera représentée par deux ou trois caractères contigus indiquant son rang $(A, R, D, V, 10, \ldots, 2)$ et sa couleur (p, co, ca, t). Deux représentations de cartes sur une même ligne seront toujours séparées par un unique espace.

— La premiere ligne du fichier contiendra les représentations des deux cartes du premier joueur.

— La seconde ligne du fichier contiendra un `?` si la donne du second joueur est inconnue et sinon, les représentations de ses deux cartes.

— La troisième ligne contiendra les représentation des cartes présentes sur la table.

Exemple 1. Le fichier `ex1.txt` :

```
Aca 10p
Dt Aco
Rco 10t 10ca 3t Vco
```

représente une configuration complète dans laquelle les cartes du premier joueur sont $A\diamondsuit, 10\spadesuit$, les cartes du deuxième joueur sont $D\clubsuit, A\heartsuit$ et les cinq cartes présentes sur la table sont $R\heartsuit, 10\clubsuit, 10\diamondsuit, 3\clubsuit, V\heartsuit$. La commande `compute ex1.txt` sur le terminal devrait donc afficher à l'écran

```
Le joueur 2 est gagnant.
```

Exemple 2. Par exemple, le fichier `ex2.txt` :

```
Aca 10p
Dt Aco
Rco 10t 10ca 3t
```

représente une configuration à deux donnes incomplète, dans laquelle les cartes du premier joueur sont $A\diamondsuit, 10\spadesuit$, les cartes du deuxième joueur $D\clubsuit, A\heartsuit$ et les quatre cartes présentes sur la table sont $R\heartsuit, 10\clubsuit, 10\diamondsuit, 3\clubsuit$. La commande `compute ex2.txt` sur le terminal devrait donc afficher à l'écran

```
Joueur 1: 0,9091
Joueur 2: 0.0909
```

Exemple 3. Le fichier `ex3.txt` :

```
Aca 10p
?
Rco 10t 10ca 3t
```

représente la configuration précédente sauf pour la donne du deuxième joueur qui est inconnue. La commande `compute ex3.txt` sur le terminal devrait donc afficher à l'écran

```
Joueur 1: 0,9560
```

4 Extensions (dont au moins une obligatoire)

Toutes les extensions seront les bienvenues, et leur présence sera prise en compte lors de l'évaluation. Néanmoins, parmi les extensions détaillées ci-dessous, une doit être obligatoirement mise en œuvre dans votre projet.

Interface graphique. Une extension plutôt simple à mettre en œuvre, et procurant une certaine satisfaction, est bien sûr l'implémentation d'une interface graphique. Cette interface doit permettre au moins de choisir une donne et les cartes présentes sur la table. Vous pouvez vous inspirer des calculettes disponibles sur internet pour d'autres fonctionnalités (sans copier de code source, bien entendu !).

Boucle d'interaction. Cette extension consiste à implémenter une interface textuelle basique pour jouer au "Texas Hold'em" à deux joueurs (chacun devra regarder l'ecran à tour de rôle pour connaître sa donne). Vous trouverez une description détaillée des six phases de jeu (*blinds*, *pre-flop*, *flop*, *turn*, *river*, *showdown*) dans Wikipedia : `https://fr.wikipedia.org/wiki/Texas_hold'em`. À chaque tour d'enchères, le programme devra afficher la probabilité de victoire du joueur devant faire sa mise. Nous apprécierons vos efforts pour éviter de faire des calculs inutiles pour réactualiser les probabilités de victoire de chaque joueur d'une phase à l'autre.

Calculette à plusieurs joueurs. Cette extension consiste à étendre la calculette au jeu à plus de deux joueurs, en définissant une *nouvelle* fonction prenant en argument un nombre de joueurs `n` (entre 1 et 9), une ou plusieurs donnes (au plus `n`) et un ensemble de cartes sur la table, et calculant la probabilité de victoire de chaque joueur dont la donne est connue. Le défi sera d'écrire du code optimal en temps d'exécution.

Intelligence artificielle (IA). Cette extension est plus complexe, et nécessite aussi d'écrire une boucle d'interaction. L'utilisateur devra pouvoir jouer contre l'ordinateur (IA). L'IA doit suivre une stratégie à détailler pendant la soutenance, prenant en compte sa probabilité de victoire. Vous pouvez aussi implémenter des IA différentes, suivant les différents caractères d'un joueur de poker (joueur serré, flambeur, vulnérable, sous-marin, bluffeur...).

5 Modalités de rendu

Le projet est à réaliser par groupes de 2 personnes au plus :

les projets réalisés à 3 personnes ou plus ne seront pas acceptés.

Votre rendu consistera en :

— un code-source écrit en OCaml, compilable et utilisable tel quel sous Linux et/ou sur les ordinateurs de l'UFR ; tout commentaire utile dans le code est le bienvenu ;

— un fichier texte nommé README contenant vos noms et indiquant brièvement comment compiler votre code-source et, si nécessaire, comment utiliser l'exécutable produit ;

— tout autre fichier nécessaire à la compilation et à l'exécution : fichiers d'exemples, `Makefile`, script permettant de compiler votre code-source, etc.

Tous ces éléments seront placés dans une unique archive compressée en `.tar.gz`. L'archive devra *obligatoirement* s'appeler *`nom1-nom2`*`.tar.gz`, et s'extraire dans un répertoire *`nom1-nom2/`*, où *`nom1`* et *`nom2`* sont les noms des deux personnes constituant le groupe. Par exemple, si vous vous appelez Pierre Corneille et Jean Racine, votre archive devra s'appeler `corneille-racine.tar.gz` et s'extraire dans un répertoire `corneille-racine/`.

La date limite de soumission est le vendredi 5 janvier à 23h59.

La soumission se fera via la page Moodle du cours.

Les projets rendus au delà de ce délai ne seront pas acceptés.

Attention, il s'agit d'un projet, pas d'un TP — vous devez donc nous fournir un programme non seulement bien écrit, mais faisant au moins quelque chose. Il vaut mieux un projet incomplet mais bien écrit et dans lequel une partie des choses fonctionnent, plutôt que d'essayer de tout faire, et de nous rendre un programme mal rédigé et/ou dans lequel rien ne fonctionne.

Enfin, de manière évidente,

votre code doit être strictement personnel.

Il ne doit s'inspirer ni de code trouvé sur le web, ni de celui d'un autre groupe, ni d'une quelconque "aide" trouvée sur un forum [6]. Inversement, il relève de votre responsabilité de faire en sorte que votre code reste inaccessible aux autres groupes. Par exemple, si vous vous servez d'un dépot `git`, ce dépot doit être privé.

6. Un plagiat avéré en projet constitue une fraude aux examens, passible de sanctions discplinaires assez lourdes. Une telle issue s'est déjà produite dans le cadre de ce cours.