

Projet individuel d'algorithmique-programmation AP1 :

groupe 1.2

octobre 2010

1 Informations générales

1.1 Travail à rendre

Le projet est à réaliser en OCaml **individuellement**. Il sera accompagné d'un **dossier** contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées).

Même si le sujet est décomposé en questions, il est possible qu'une question se résolve par l'écriture d'une ou plusieurs fonctions intermédiaires. Celles-ci doivent comporter une interface également.

Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés.

Sur le site du cours figure un petit document sur ce que l'on attend dans un rapport. Consultez-le!

1.2 Calendrier et procédure de remise

Le projet (dossier + copie du listing de code) est à rendre au secrétariat **le 22 novembre à 16h au plus tard**. Le numéro du groupe, ainsi que le nom du chargé de TD, devront figurer en gros, et en rouge sur la page de garde.

Le fichier .ml contenant votre code devra être déposé électroniquement au plus tard **le 22 novembre à minuit**.

Les soutenances seront organisées dès la semaine suivante. Il vous faudra consulter les panneaux d'affichage et votre courrier électronique pour obtenir l'ordre de passage.

Enfin n'attendez pas pour vous mettre au travail! Un projet se travaille dès la remise du sujet afin d'avoir le temps de laisser murir la solution et de poser des questions au client (dans votre cas, votre chargé de TP).

1.3 Procédure de dépôt

Pour déposer le code du projet, consultez les informations sur les machines de l'école, tout figure.

Ne vous inquiétez pas, l'interface fabrique un nom de projet déposé à partir de votre login. Votre projet ne sera pas confondu avec celui d'un autre.

Indiquez quand même en commentaire dans votre fichier de code Ocaml votre nom et votre groupe. Ce sera plus facile pour le correcteur.

Vous pouvez déposer successivement plusieurs versions, le dernier dépôt écrase le précédent, seul le dernier dépôt est pris en compte

Enfin tout cela peut se faire de l'extérieur.

Le code à déposer est un fichier.ml commenté avec les interfaces des fonctions. Le fichier doit pouvoir être compilé (sans aucune intervention humaine - lecture du buffer complet) sur les machines Yaka de l'école, n'oubliez pas de vérifier que tout fonctionne sur les machines de l'école avant de le dép oser - si vous l'avez développé sous un autre système d'exploitation.

2 Énoncé du projet : Comparaison d'IA de poker

3 Introduction

L'objectif de ce projet est de créer un comparateur d'IA de Poker pour des tables de type *argent réel*. Les règles utilisées seront une version simplifiée du *Texas hold'em*. Les règles, ainsi que les petits aménagements consentis pour que le projet ne soit pas trop difficile sont expliqués ici. Néanmoins, vous pouvez également consulter <http://fr.wikipedia.org/wiki/Poker> pour obtenir les règles du Poker et http://fr.wikipedia.org/wiki/Texas_hold'em pour obtenir plus d'informations sur la variante utilisée dans ce projet.

4 Règles du jeu

Le jeu se déroule autour d'une table. Chaque joueur dispose de 200 jetons qui constituent son tapis. Au cours du jeu, il peut miser ses jetons pour tenter de prendre ceux des autres. Ici, l'objectif pour chaque joueur sera de maximiser ses gains (on ne considère pas le format des tournois). Pour simplifier, le nombre de joueurs à la table sera fixé. Si un joueur n'a plus de jeton, il en rachète automatiquement en s'endettant. Inversement, si un joueur devient trop riche, une partie de son tapis sera mise de côté. Ainsi le nombre de jetons total sur la table ne variera pas trop.

Le jeu se joue en plusieurs tours. L'ordre des joueurs est important. Dans une partie réelle, le dernier joueur est repéré par un *bouton* et l'on tourne dans le sens des aiguilles d'une montre. Le *bouton* est passé au joueur de gauche au début de chaque tour. Pour le projet, on placera les joueurs dans une liste ordonnée et on mettra le premier de la liste à la fin de celle-ci au début de chaque tour. A chaque tour, un jeu de 52 cartes (jeu standard sans jocker) est mélangé puis deux cartes sont distribuées à chaque joueur. Ces cartes ne sont visibles que par les joueurs qui les possèdent. Cinq autres cartes seront ensuite révélées à tous les joueurs progressivement entre les phases de mise.

A la fin du tour, le joueur qui a la meilleure main remporte le total des mises du tour. Si plusieurs joueurs ont des mains équivalentes, il se partagent le tout équitablement. Le premier joueur peut éventuellement recevoir un jeton de plus que les autres si le calcul ne tombe pas juste. Pour réaliser sa main, le joueur doit choisir 5 cartes parmi 7 : les deux qu'il est le seul à voir, et les 5 cartes visibles. La méthode pour comparer des mains est expliquée dans la section suivante.

Les deux premiers joueurs sont obligés de miser les *blinds*. Ce sont des mises obligatoires, pour qu'il y ait des jetons en jeu à chaque tour. Le premier joueur misera toujours 1 jeton (petite blind), et le second en misera 2 (grosse blind). Toutes les mises effectuées pendant un tour seront partagées par les gagnants du tour.

Un tour se compose de quatre phases de mise. Pendant une phase de mise les joueurs peuvent choisir, chacun leur tour, de passer/checker, de suivre ou bien de relancer. A la fin d'une phase de mise tous les joueurs qui sont encore dans le coup doivent avoir payé la même somme de jetons. Ceux qui n'ont pas payé cette somme ne jouent plus jusqu'au prochain tour. Ils ne peuvent plus prétendre récupérer les jetons qui ont été misés, même les leurs. Lorsqu'un joueur passe/check, il ne veut plus investir de jetons supplémentaires ce tour-ci. S'il n'y a pas eu de relance, il conserve ses cartes, sinon, il rend ses cartes et ne participe plus à ce tour. On dit qu'il est couché. Si un joueur suit, il complète sa mise actuelle pour égaler la dernière relance. Enfin, lorsqu'un joueur relance, il doit ajouter à sa mise actuelle le double de la dernière relance de

cette phase de mise (ou le montant de la grosse blind s'il y a pas encore eu de relance). Pour simplifier le jeu, la relance maximale est fixée au tapis (l'ensemble des jetons) du joueur qui a le moins de jetons. Si un joueur essaie de relancer plus haut que cette limite, sa relance sera réduite à cette limite. Quand un joueur a misé tous ses jetons, plus aucune relance n'est possible. Une fois que tous les joueurs ont suivi la dernière relance ou qu'ils se sont tous couchés, la phase de mise est terminée. Tous les jetons misés sont alors placés au milieu, la mise des joueurs est remise à zéro, et on peut passer à la phase suivante.



Attention : Pour la première phase de mise (Pré-flop) les mises commencent par le 3ème joueur et non le premier. Cela vient du fait que les deux premiers joueurs ont déjà misé les *blinds*.

Un tour comprend donc plusieurs phases de mises. La première a lieu pré-flop, c'est à dire avant qu'une seule carte ne soit révélée face visible aux joueurs. Après cette phase de mise, 3 cartes de la pioche sont révélées, puis une nouvelle phase de mise commence. Cette phase sera appelée Flop. Ensuite une quatrième carte est révélée, puis une phase de mise appelée Turn est effectuée. Enfin, la dernière des 5 cartes est dévoilée puis une ultime phase de mise a lieu, qu'on appellera River.



Attention : Avant de révéler des cartes, donc trois fois dans la partie, on brûle une carte, cela signifie que l'on enlève la première carte du paquet, et que l'on prend les suivantes. Les cartes brûlées ne sont pas révélées.

5 Comparer des mains

Visitez http://fr.wikipedia.org/wiki/Main_au_poker si vous ne savez pas comment comparer deux mains.

6 Types

Les types permettant de construire une partie vous sont donnés. Vous devrez utiliser ces derniers tout au long de votre projet.

```
(* Couleurs des cartes. *)
type t_couleur = Pique | Coeur | Trefle | Carreau;;

(* Hauteur d'une carte, comprise entre 2 et 14
  11 : Valet
  12 : Dame
  13 : Roi
  14 : As *)
type t_hauteur = int;;

(* Chaque carte possède une couleur et une hauteur.
  Il n'y a pas de joker. *)
type t_carte = t_hauteur * t_couleur;;

(* Valeur d'une main. *)
type t_valeur =
  | Quinteflush of t_hauteur
```

```

| Carre of t_hauteur * t_hauteur
| Full of t_hauteur * t_hauteur
| Couleur of t_hauteur * t_hauteur * t_hauteur * t_hauteur * t_hauteur
| Suite of t_hauteur
| Breelan of t_hauteur * t_hauteur * t_hauteur
| Doublepaire of t_hauteur * t_hauteur * t_hauteur
| Paire of t_hauteur * t_hauteur * t_hauteur * t_hauteur
| Hauteur of t_hauteur * t_hauteur * t_hauteur * t_hauteur * t_hauteur;;

```

Le type `t_valeur` permet de donner la valeur d'une main. Deux mains peuvent être comparée à partir de leurs seules valeurs. Pour chacun des constructeurs, on donne toutes les hauteurs nécessaires pour comparer deux mains entre elles, par ordre d'importance. Ainsi la main :

```
[(14,Pique);(14,Trefle);(5,Carreau);(14,Pique);(4,Coeur)]
```

aura la valeur Breelan (14, 5, 4) et la main [(14,Pique);(12,Pique);(5,Pique);(2,Pique);(4,Pique)] aura la valeur Couleur (14, 12, 4, 4, 2) car la hauteur des 5 cartes peut être nécessaire pour comparer deux mains qui forment des couleurs.

```

(* Etat d'un joueur. *)
type t_joueur = {
  joueur_nom : string;
  joueur_ia : t_joueur_ia;
  joueur_argent : int; (* argent du joueur en dehors de la table *)
  joueur_tapis : int; (* tapis du joueur sur la table *)
  joueur_mise : int; (* jetons mises pour le coup en cours *)
  joueur_cartes : t_carte list (* 2 cartes du joueur pour le coup en cours *)
};;

```

```

(* Etat d'une partie de poker avant de jouer une main. *)
type t_partie = {
  partie_joueurs : t_joueur list;
  partie_cartes_visibles : t_carte list; (* cartes visibles sur la table *)
  partie_cartes_restantes : t_carte list; (* cartes restantes dans le paquet *)
  partie_pot : int; (* jetons au milieu de la table *)
};;

```

Un joueur possède un tapis initial de 200 jetons. Son argent peut être positif ou négatif, il en perd lorsqu'il doit racheter des jetons s'il n'en a plus assez ; il en gagne quand son tapis est trop important et qu'il met des jetons de côté. A la fin de la simulation, la richesse d'un joueur sera évaluée par la somme de son argent et de son tapis. Lorsqu'un joueur est couché, on lui retire ses cartes.

7 Intelligences artificielles

Vous allez comparer plusieurs IA de Poker. Chaque IA sera constituée d'une liste de règles. Chaque règle associera une condition et une action. Pour faire prendre une décision à une IA, il suffit de parcourir les règles dans l'ordre. Dès que la condition d'une règle est vérifiée, on applique l'action associée.

```

(* Condition pour une regle *)
type t_condition_ia =
| CondEt of t_condition_ia * t_condition_ia
| CondOu of t_condition_ia * t_condition_ia
| CondSup of t_cond_nombre * t_cond_nombre
| CondEq of t_cond_nombre * t_cond_nombre
| Preflop (* Regle valable uniquement avant le flop *)

```

```

| Flop (* Regle valable uniquement juste apres le flop *)
| Turn (* Regle valable uniquement juste apres le turn *)
| River (* Regle valable uniquement apres la river *)
| CartesAssorties
| Cartes of t_hauteur * t_hauteur
  (* Cartes de certaines hauteurs (la carte la plus haute en premier) *)
| CartesMin of t_hauteur * t_hauteur
  (* Cartes de hauteurs superieures ou egales a celles precisees *)
;;

(* Action d'une joueur : Check / Coucher, Suivre ou Relancer. *)
type t_action_ia =
  | CheckCoucher
  | Suivre
  | Relancer;;

(* Type d'une IA *)
type t_joueur_ia = (t_condition_ia * t_action_ia) list;;

```

Par exemple l'IA qui suit, suivra tout le temps avant le flop tant que la mise reste raisonnable par rapport à la taille du flop. Après le flop elle relancera si elle a plus de 75% de chances de gagner face à une main quelconque et suivra si elle a au moins 25% de chances.

```

let ia_basique = [
  (CondEt(Preflop, CondSup (OpMult (Entier 4, PotSize), MontantSuivi)), Suivre);
  (CondSup (Chances, Entier 75), Relancer);
  (CondSup (Chances, Entier 25), Suivre);
];;

```

Voici quelques exemples d'IA que vous pouvez utiliser pour votre projet :

```

let ia_basique = [
  (CondEt(Preflop, CondSup (OpMult (Entier 4, PotSize), MontantSuivi)), Suivre);
  (CondSup (Chances, Entier 75), Relancer);
  (CondSup (Chances, Entier 25), Suivre);
];;
let ia_fou_des_as = [
  (CartesMin(14, 2), Relancer);
]@ia_basique;;
let ia_selection = [
  (CondEt (Preflop, CartesMin(11, 11)), Relancer);
  (CondEt (Preflop, CartesMin(11, 2)), Suivre);
  (Preflop, CheckCoucher);
]@ia_basique;;
let ia_fou_du_velo = [
  (CondEt (Preflop, Cartes(14, 14)), Relancer);
  (CondEt (Preflop, Cartes(13, 13)), Relancer);
  (CondEt (Preflop, Cartes(12, 12)), Relancer);
  (CondEt (Preflop, Cartes(11, 11)), Relancer);
  (CondEt (Preflop, Cartes(10, 10)), Relancer);
  (CondEt (Preflop, Cartes(9, 9)), Relancer);
  (CondEt (Preflop, Cartes(8, 8)), Relancer);
  (CondEt (Preflop, Cartes(7, 7)), Relancer);
  (Preflop, CheckCoucher);
]@ia_basique;;

```

8 Questions

Question 1 Ecrivez la fonction `valeur_main liste_cartes` de type `(t_hauteur * t_couleur)list -> t_valeur` qui renvoie la valeur de la meilleure main que l'on peut former avec les 5 à 7 cartes que contient `liste_cartes`. Pour cela, vous pouvez écrire des fonctions auxiliaires qui recherche un type de figure particulier. Par exemple, `cherche_carre liste_cartes` qui renvoie la valeur d'un carré si elle en trouve un, et qui lève l'exception `Not_found` si elle ne trouve aucun carré. Vous pourrez alors essayer chacune des fonctions auxiliaires dans `valeur_main` et rattraper les exceptions si nécessaire (`try ___ with ___ -> ___`)



Astuce : Pour écrire les fonctions auxiliaires, vous pouvez utiliser la fonction `List.sort` pour trier les cartes par hauteur puis par couleur. Vous aurez besoin d'une fonction de comparaison des cartes pour l'utiliser.

Question 2 Ecrivez la fonction `compare_mains cartes_visibles m1 m2` de type `(t_hauteur * t_couleur)list -> (t_hauteur * t_couleur)list -> (t_hauteur * t_couleur)list -> int` qui compare les mains de deux joueurs. `cartes_visibles` contient 3, 4 ou 5 cartes et `m1` et `m2` en contiennent 2. Cette fonction doit comparer la meilleure main que l'on peut former avec `m1` et la meilleure main que l'on peut former avec `m2`. `compare_mains` renvoie 1 si `m1` est mieux que `m2`, 0 si elles sont de valeurs équivalentes et -1 sinon.

Question 3 Créez une variable `pioche` qui contient l'ensemble des 52 cartes du jeu de poker. Vous utiliserez pour cela au moins une fonction auxiliaire.

Question 4 Ecrivez la fonction `melange_pioche liste_cartes` de type `'a list -> 'a list` qui prend une liste de carte en argument et qui renvoie la même liste dans le désordre. Vous pouvez pour cela écrire une fonction auxiliaire qui insère un élément dans une liste à un rang donné.



Astuce : Pour obtenir un nombre aléatoire entre 0 compris et `n` non compris, utilisez la fonction `Random.rand n`. Note : Cette fonction donne toujours les mêmes nombres aléatoires. Pour changer ces derniers, utilisez la fonction `Random.init`.

Question 5 Ecrivez la fonction `calcule_entier_ia cond_nombre pot suivi relance chances` de type `t_cond_nombre -> int -> int -> int -> int -> int` qui calcule la valeur entière d'un élément de type `t_cond_ia` à partir des autres paramètres, la fonction `verif_cond_ia cond_ia etat cartes pot suivi relance chances` de type `t_condition_ia -> t_condition_ia -> (t_hauteur * 'a)list -> int -> int -> int -> int -> bool` qui vérifie si une `cond_ia` est vraie et la fonction `prendre_decision_ia ia etat cartes pot suivi relance chances` de type `(t_condition_ia * t_action_ia)list -> t_condition_ia -> (t_hauteur * 'a)list -> int -> int -> int -> int -> t_action_ia` qui renvoie l'action choisie par l'IA en fonction des autres paramètres. L'action par défaut quand aucune des règles n'est applicable est `CheckCoucher`.

Question 6 Ecrivez les fonctions suivantes :

- `init_partie liste_noms_ia` de type `(string * t_joueur_ia)list` qui crée une nouvelle partie prête à être utilisée à partir de la liste des joueurs (dont on donne le nom et l'IA). Vous pouvez créer une fonction `init_joueur`. Chaque joueur commence sans argent, avec un tapis de 200 jetons, et ne possède pas encore de carte.

- `tourner_joueurs` partie de type `t_partie` -> `t_partie` qui renvoie partie dans laquelle le premier joueur a été placé en dernier, et `inv_tourner_joueurs` partie qui fait l'effet inverse.
- `mise_joueur` joueur mise de type `t_joueur` -> `int` -> `t_joueur` qui renvoie un joueur pour lequel la nouvelle `joueur_mise` est égale à `mise`. Si le joueur avait déjà une mise, on retire moins de jetons de son tapis.
- `ajuste_tapis` partie de type `t_partie` -> `t_partie` qui ajuste le tapis des joueurs de la partie avant de commencer un nouveau tour. Les joueurs ayant un tapis inférieur au quart du tapis initial et ceux qui ont un tapis supérieur à 4 fois le tapis initial repartent avec un tapis de 200. Le champ `joueur_argent` doit bien entendu être adapter pour que la somme de l'argent et du tapis d'un joueur ne varie pas.
- `distribue_et_pose_blinds` partie de type `t_partie` -> `t_partie` qui mélange la pioche pour former une les nouvelles *cartes restantes* de la partie, puis distribue deux cartes à chaque joueur. Enfin elle fait miser les deux premiers joueurs 1 et 2 jetons (petite et grosse blind).



Astuce : Pour toutes ces fonctions, vous pouvez utiliser la syntaxe `{partie with partie_joueurs = nouv_joueurs}` qui permet de créer un nouvel enregistrement similaire à `partie`, en dehors du champ `partie_joueurs` qui sera égal à `nouv_joueurs`.

Question 7 Ecrivez la fonction `joue_joueur` joueur etat pot mise cartes_visibles mise_max de type `t_joueur` -> `t_condition_ia` -> `int` -> `int` -> `t_carte` list -> `int` -> `t_joueur * int * int` qui fait jouer un joueur. `mise` est le montant de la dernière relance, `mise_max` le montant de la mise maximale. La décision doit être prise en utilisant l'IA, puis la fonction doit renvoyer un triplet (`joueur`, `mise`, `pot`) avec le joueur mis à jour, la dernière relance (l'ancienne valeur s'il n'y a pas de relance, le montant de la relance s'il y en a une) et la nouvelle valeur du pot si le joueur y a mis des jetons.



Astuce : Pour écrire la fonction `joue_joueur`, il faut savoir calculer les chances de victoire d'une main par rapport à toutes les main possibles (toute les mains sauf celle qui contiennent les cartes que le joueur a vu). Cette opération étant complexe, vous pouvez la laisser pour plus tard et toujours donner la valeur 50 à `chances`. N'essayez pas de calculer cette valeur pré-flop. On peut poser que `chances` vaut toujours 0 avant le flop.

Question 8 Ecrivez la fonction `jouer_phase` partie etat de type `t_partie` -> `t_condition_ia` -> `t_partie` qui renvoie une partie après une phase de mise complète.



Astuce : Chaque joueur doit prendre au moins une décision. Il faut commencer par un tour complet de la liste des joueurs. Ensuite, il faut faire jouer les joueurs un par un jusqu'à ce que tous les joueurs soient couchés ou bien qu'ils aient payé le montant de la plus grosse mise (dernière relance).



Attention : Le cas `etat = Preflop` est un cas particulier, car les mises commencent à partir du troisième joueur. Pensez à utiliser les fonctions `tourner_joueurs` et `inv_tourner_joueurs`.

Question 9 Ecrivez la fonction `jouer_tour` partie de type `t_partie -> t_partie` qui effectue un tour de jeu complet. Cette fonction doit réaliser des 4 phases de mise puis partager le pot entre les vainqueurs (celui ou ceux qui ont la meilleure main).

Question 10 Ecrivez la fonction `stats_liste_noms_ia` n de type `(string * t_joueur_ia)list -> int -> (string * int)list` qui fait jouer une partie de n tours aux joueurs contenus dans la liste. Elle renvoie une liste de couples avec le nom de chaque joueur et le montant de jetons qu'il possède (tapis + argent).



Attention : Plus y il a de joueurs (il en faut au moins 3) et plus n est grand, plus la fonction sera longue a exécuter.

9 Pour aller plus loin

Plusieurs améliorations peuvent être apportées au projet. Tout d'abord, vous pouvez essayer de concevoir des IA plus performantes que celles fournies. Il est également possible d'enlever la limite de mise au niveau du tapis du joueur le moins riche. Il faut alors gérer plusieurs pots pendant les mises. Enfin, vous pouvez étendre les types `t_condition_ia` et `t_cond_nombre` afin qu'ils acceptent plus de paramètres (nombre de relances, position du joueur à la table).

10 Conseils

Le découpage des fonctions qui est suggéré dans certaines questions n'est pas obligatoire. Les seules fonctions qui vous sont imposées sont les fonctions principales de chaque question. Il est tout de même fortement recommandé de créer au moins autant de fonctions auxiliaires que ce qui vous est proposé. Rien ne vous empêche, par contre, d'en ajouter autant que vous le désirez.

Testez toutes les fonctions que vous écrivez. Il vaut mieux passer une heure à écrire une fonction juste, plutôt que dix minutes à en écrire une fausse qui posera par la suite des problèmes difficiles à identifier.

Le projet est assez long. Visez la qualité plutôt que la quantité. Vous serez mieux noté si vous parvenez à terminer correctement les premières questions que si vous atteignez la dernière question sans avoir tester les premières.

Pour ne pas avoir à copier le code présent dans cet énoncé, vous pouvez copier le fichier : `/home/prof/pierrenicolas.tollitte/poker.ml`.