

Projet de programmation fonctionnelle: Les dames chinoises

Benoît Barbot, Nicolas Basset

Licence 2, Université Paris Est Créteil

Résumé

Les **dames chinoises** est, comme son nom ne l'indique pas, un jeu dérivé du jeu Halma inventé par l'Américain George Howard Monks en 1883.

La version classique se joue avec 10 pions par joueurs que l'on déplace sur un plateau comportant 121 emplacements disposés en étoile à 6 branches. Voir figure 1. Le but de ce projet est de réaliser une application pour jouer à ce jeu.

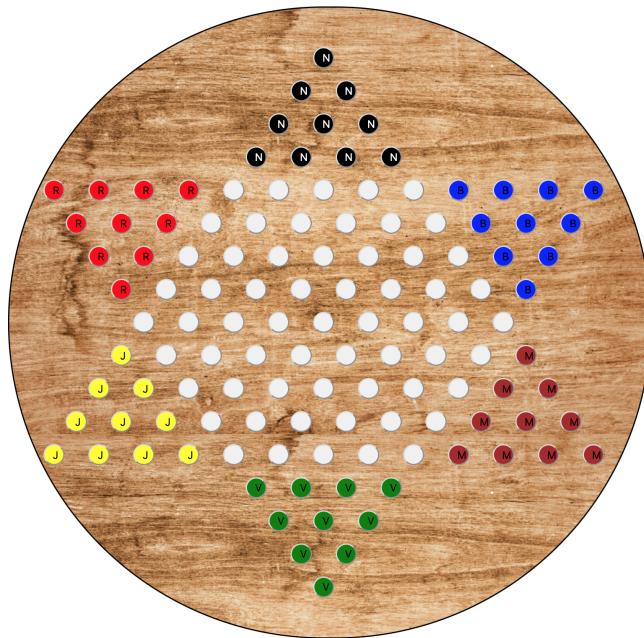


FIGURE 1 – Plateau de jeux dans sa configuration initiale pour une partie à six joueurs.

1 Consignes

toutes les informations pour le projet se trouve sur www.caseine.org. Vous y trouverez ce document, une archive `projet.zip` contenant les fichiers du projet,

une activité de rendu et un lien vers une page démontrant le projet. Le but de la première partie de ce projet est de reproduire le programme sur cette page : <https://www.lacl.fr/~barbot/file/projetfun2021/display.html>. Par rapport aux TD les questions sont moins guidées et elles le sont de moins en moins au fur et à mesure de l'avancée du projet. La section 2 sert à vous familiariser avec le système de coordonnées, elle permet de définir un certain nombre de fonctions auxiliaires utiles pour la suite. La section 3 est la première partie du projet. Vous pouvez commencer cette partie avant d'avoir fini les préliminaires et revenir au début quand il vous manque certaines fonctions.

1.1 Compilation

Dans l'archive vous trouverez les fichiers suivants :

- Le fichier `rendu_etudiant.ml` est le fichier principal, c'est le seul que vous devez modifier, il contient pour le moment des définitions de types et des squelettes de fonction. Ce code est détaillé dans la section suivante.
- Les fichiers `display.ml`, `html.ml` contiennent du code OCaml permettant de construire une interface utilisateur web à partir de votre fichier. Vous n'avez pas à les modifier
- Le fichier `display_ascii.ml` contient du code OCaml permettant d'afficher la configuration initiale dans un terminal.
- Le fichier `Makefile` contient un script pour compiler le projet
- Les fichiers restant `display.html`, `fond-bois2.jpg` servent à faire l'interface graphique.

Pour compiler le projet vous aurez besoin d'installer <https://opam.ocaml.org/> un gestionnaire de paquet pour OCaml. Sous Windows, vous pouvez suivre les instruction disponible ici <https://moodle.caseine.org/mod/book/view.php?id=43384>. Vous devez installer les paquets `ocamlfind`, `ocamlbuild`, `js_of_ocaml`, `js_of_ocaml-ppx`. Une fois opam installé vous pouvez le faire avec la commande `opam install ocamlfind ocamlbuild js_of_ocaml js_of_ocaml-ppx`.

Le projet peut être exécuté de deux manières différentes :

- En mode web application, taper `make` et une fois la compilation finie vous pouvez ouvrir le fichier `display.html` avec un navigateur pour charger l'interface du projet. La liste des cases que vous cliquées est stockée comme un coup et affichées dans l'interface, en cliquant sur jouer le coup est envoyé à votre fonction `mis_a_jour_configuration`.
- En mode interprété, vous pouvez exécuter `ocaml display_ascii.ml` qui fera un affichage ASCII dans le terminal de la configuration. Pour effectuer un coup, il faut donner une liste de case et taper sur entrée (*Vous pouvez copier-coller la liste de case générée par l'interface web.*)

Vous devez avancer dans le projet pour que le plateau ressemble à celui de la figure 1. Initialement le projet va afficher un quadrillage de case,

1.2 Date de rendu

le projet est à réaliser par binôme le choix des binômes ce fait sur caséine : <https://moodle.caseine.org/mod/choicegroup/view.php?id=43373>. La première partie du projet doit être rendu mardi 16 novembre 23 h 59.

1.3 Les définitions qui vous sont données

les définitions suivantes vous sont données dans le fichier `rendu_etudiant.ml`.

```
let dim=4

(* Une case est un triplet d'entier l'encodage est décris
   plus loin dans ce document*)
type case=int*int*int

type couleur=
  (* Les six couleurs des joueurs *)
  | Vert
  | Jaune
  | Rouge
  | Noir
  | Bleu
  | Marron

(*case libre du plateau, utile pour l'affichage*)
| Libre
(*case en dehors du plateau, utile pour l'affichage*)
| Dehors
(*pour mettre des nombre dans une configuration*)
| Nombre of int
(*pour mettre des petits noms*)
| Nom of string

(* Chaque case coloré correspond à un pions il
   possède une couleur et une case *)
type case_coloree = (case*couleur)

(* La liste des joueurs est une liste de couleurs qui
   indique la liste des joueurs ainsi que leur ordre
   pour jouer. Le joueur dont c'est le tour sera
   toujours en tête de liste *)
type liste_joueur = couleur list

(* Une configuration est un couple comprenant une
   liste de case_coloree et la liste des joueurs. *)
type configuration = case_coloree list * liste_joueur
type coup = Du of case*case | Sm of case list
```

De plus le type `('a,'b) result = Ok of 'a | Error of 'b` est un type déjà défini dans OCaml.

Le fichier `rendu_etudiant.ml` contient également des squelettes de fonctions vous aurez à les modifier pour avancer dans le projet mais vous devez toujours garder les mêmes types sans quoi le projet ne compilera plus.

```
(* La configuration initial pour le début de partie,
   pour le moment le plateau est vide. *)
let configuration_initial =
  ([] , [ Vert; Jaune; Rouge; Noir; Bleu; Marron ])

(* Renvoi la liste des joueurs vous n'avez pas à
   toucher à cette fonction*)
let liste_joueurs (_, 1) = 1

(* Renvoi la couleur des pions dans les cases *)
let quelle_couleur _ _ = Libre

(* Fonction principale à chaque coup cette fonction
   vérifie que le coup est valide et met l'a jour
   la configuration*)
```

```

let mis_a_jour_configuration _ _ = Error "To do"

(* Indique si un joueur a gagner la partie, Libre indique
pas de gagnant*)
let gagnant _ = Libre

```

2 Préliminaires

2.1 Le plateau et les coordonées des cases

Au début de la partie, les pions d'un joueur se situent dans son camp qui est le triangle devant lui qui comprend 4 lignes de pions comportant en allant de l'extérieur vers le centre 1,2,3 et 4 pions.

On dira que le joueur le plus au Sud est le protagoniste, son but est de déplacer ses pions du camp le plus au Sud vers le camp, en face, des cases les plus au Nord. Les joueurs jouant à tour de rôle, on tournera le plateau de telle sorte que le joueur qui doit jouer soit le protagoniste, son camp étant donc au Sud avec pour but le camp Nord.

Dans ce projet de programmation, nous pouvons choisir une taille de plateau différente. On définit ainsi au début du code OCaml une constante `dim` de type `int` qui correspond au nombre de lignes de pions qu'un joueur a. Le plateau étoilé a ainsi `4*dim+1` lignes horizontales que nous numérotions de bas en haut de `-2*dim` à `2*dim`.

Une case est définie par trois coordonnées (i, j, k) , la case au centre du plateau de jeu a pour coordonnée $(0, 0, 0)$. La figure (2a) illustre ce système de coordonnées, Les coordonné représentent

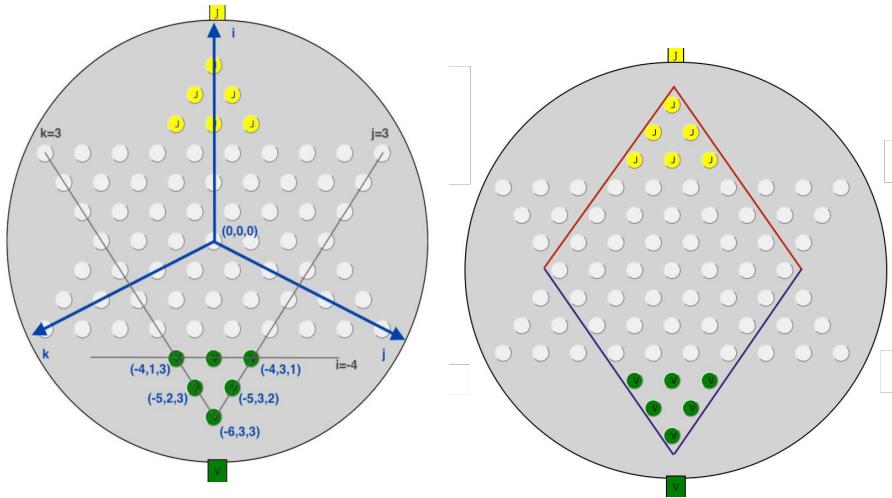
- le numéro de la ligne horizontale
- le numéro de la ligne horizontale lorsqu'on a tourné le plateau d'un tiers de tour dans le sens antihoraire (dans la Figure 1, les rouges prennent la place des verts au sud).
- le numéro de la ligne horizontale lorsqu'on a tourné le plateau d'un tiers de tour dans le sens horaire (dans la Figure 1, les bleus prennent la place des verts au sud).

Il est à noté que l'équation $i + j + k = 0$ est satisfaite pour toutes les cases de la grille. Il sera utile de tester si un triplet tel que $i + j + k = 0$ est bien dans une certaine partie du plateau, ou même tout simplement dans l'étoile.

Le but des questions suivantes est de vous familiariser avec les coordonnées des cases et d'écrire les premières fonctions de tests.

question 1 Pour chaque condition ci-après décrire les cases qui satisfont cette condition :

1. $i < -\text{dim}$
2. $i > \text{dim}$
3. $j < -\text{dim}$
4. $(i, j, k) = (2\text{dim}, -\text{dim}, -\text{dim})$
5. $i \geq -\text{dim} \wedge j \geq -\text{dim} \wedge k \geq -\text{dim}$



(a) Système de coordonnées : les axes sont marqués en bleu foncé, les coordonnées de l'origine, et des pions verts sont indiquées également en bleu. Les lignes gris-clair représentent les droites $i = -4$, $j = 3$ et $k = 3$

(b) Losange NordSud, le losange contient toutes les cases dans lesquelles un coup peut se terminer.

FIGURE 2 – Détail des coordonnées.

question 2 Donner une formule booléenne qui est vraie si et seulement si une case est dans le losange nord-sud (voir figure (2b)).

En déduire une implémentation de la fonction `est_dans_losange: case -> bool` telle que `(est_dans_losange c)` est vraie ssi `c` est une case du losange nord-sud.

question 3 Donner une formule booléenne qui est vraie si et seulement si une case est dans l'étoile. Remarquez que plusieurs définitions de l'étoile sont possibles : union de trois grands losanges, union de deux triangles, union de l'hexagone centrale et des camps de chaque joueur. Vous choisirez et expliquerez une définition correcte.

En déduire une implémentation de la fonction `est_dans_etoile: case -> bool` telle que `(est_dans_etoile c)` est vraie ssi `c` est une case de l'étoile et qui de plus vérifie que la somme des coordonnées de `c` vaut 0.

question 4 La fonction `quelle_couleur : case -> couleur` indique à l'interface où sont les cases et si elles ne sont pas libres la couleur du pion. Dans le code qui vous est donné, cette fonction renvoie toujours `Libre`. Modifier la fonction `quelle_couleur` pour qu'elle renvoie `Libre` pour les cases dans l'étoile et `Dehors` pour les autres. Les interfaces utilisateur devraient maintenant afficher le plateau correctement.

question 5 Implémenter une fonction `tourne_case: int -> case -> case` telle que `tourne_case m c` est la case `c` après avoir fait tourner le plateau de `m` sixième de tour dans le sens antihoraire (dans la Figure 1, une case jaune se retrouve dans le camp vert après un sixième de tour.)

question 6 Implémenter une fonction `tourne_config: configuration -> configuration` telle que `tourne_config conf` est la configuration `conf` après pas-

sage au joueur suivant. On tourne donc la liste des couleurs ainsi que les cases en utilisant la fonction `tourne_case` avec les bons paramètres. Dans la Figure 1, le camp jaune se retrouve à la place du camp vert après l'application de `tourne_config`.

Les triplets d'entiers (i, j, k) tels que $i + j + k = 0$ servent aussi bien comme case de la grille que comme vecteur permettant des translations. Ainsi dans l'exemple de la Figure 2a le pion vert le plus à gauche est en $(-4, 1, 3)$. Si on lui ajoute le vecteur $(0, 2, -2)$ on arrive dans la case $(-4, 3, 1)$ c'est à dire la case du pion vert le plus à droite. Si on ajoute encore le vecteur $(1, 0, -1)$ on arrive dans la case libre $(-3, 3, 0)$.

On dit de deux cases qu'elles sont voisines si pour aller de l'une à l'autre on change deux coordonnées de 1. Ainsi les cases voisines de $(1, 0, -1)$ sont $(1, 1, -2), (2, 0, -2), (2, -1, -1), (0, 0, 0), (0, 1, -1)$.

question 7 Implémenter une fonction `sont_cases_voisines: case -> case -> bool` qui teste si deux cases sont voisines.

Les triangles situés en périphérie du jeu (autrement dit en dehors de l'hexagone central) sont les camps des joueurs. Un joueur a au départ tous ses pions dans son camp devant lui et a pour but de les déplacer dans le camp diamétralement opposé. Les autres camps seront appelés camps de côté.

Les joueurs jouent à tour de rôle dans le sens des aiguilles d'une montre. À son tour chaque joueur déplace un pion de son choix sur le plateau en respectant les règles suivantes :

- le pion joué doit être de la couleur du joueur à qui c'est le tour
- le pion une fois joué doit être sur une case préalablement libre du plateau, qui n'est pas dans un camp de côté.
- le déplacement est d'un des deux types suivants
 - déplacement unitaire : la case d'arrivée est une des six cases voisines de la case de départ
 - saut multiple : la case d'arrivée est obtenue après un ou plusieurs sauts au-dessus de pions pivots que nous décrirons dans la seconde partie du projet.

3 Première partie

Dans cette première partie de prise en main nous verrons comment implémenter la configuration initiale, les déplacements unitaires et les changements de tours. Nous verrons dans une seconde partie comment traiter des sauts multiples et du déroulement d'une partie.

question 8 Implémenter la fonction `case_dans_config : case -> configuration -> bool` qui renvoie vrai s'il existe un pion de n'importe quelle couleur dans la configuration à la case passée en paramètre.

question 9 Donner une implémentation de `quelle_couleur: case -> configuration -> couleur` qui étant donné une case et une configuration renvoie : Une couleur **Vert**, **Jaune**, **Rouge**, **Noir**, **Bleu**, **Marron** s'il existe un pion de couleur sur cette case, **Libre** si la case est libre et **Dehors** si la case ne correspond pas à une case du plateau.

question 10 Écrire la fonction `remplir_triangle configuration -> couleur -> case -> configuration` telle que `remplir_triangle conf col c` renvoie la configuration `conf` à laquelle aura été ajoutée un triangle de côté `dim` avec des jetons de couleur `col` avec pointe vers le bas en case `c`.

question 11 Écrire la fonction `remplir_init : liste_joueur -> configuration` qui à partir d'une liste de joueurs construit une configuration initiale. Vous pouvez utiliser `remplir_triangle` successivement en faisant tourner la configuration avec la fonction `tourne_config`.

Définir

```
let configuration_initial =
  remplir_init [Vert; Jaune; Rouge; Noir; Bleu; Marron]
```

Le projet compilé doit maintenant afficher la même chose que la figure 1.

question 12 Écrire la fonction `est_dep_unit: configuration -> case -> case -> bool` tel que `est_dep_unit conf c1 c2` est vraissi le déplacement du pion en `c1` vers `c2` est un déplacement unitaire valide, c'est à dire que les cases sont voisines, que `c1` contient un pion du joueur auquel c'est le tour de jouer, que `c2` est libre et que `c2` est dans le losange.

question 13 Écrire la fonction `fait_dep_unit: configuration -> case -> case -> configuration` qui applique un déplacement unitaire en supposant qu'il est valide.

question 14 Modifier la fonction `mis_a_jour_configuration: configuration -> coup -> (configuration, string) result` pour quelle mette à jour le plateau pour les coups unitaires. C'est à dire que l'appel à `mis_a_jour_configuration: conf cp` fait :

1. la vérification que `cp` est un coup unitaire et qu'il est valide ;
2. déplace le pion pour faire le déplacement unitaire ;
3. effectue la rotation du plateau pour le prochain joueur à qui c'est le tour soit vers le bas ;
4. met à jour la liste de joueurs pour que le prochain joueur à qui c'est le tour soit en tête de liste.

La fonction renvoie la nouvelle configuration grâce au type `result` (ex `Ok (nouvelle_configuration)`). Si une de ces étapes échoue elle renvoie une erreur avec une chaîne de caractère expliquant le problème (ex `Error "ce n'est pas un déplacement unitaire"`).