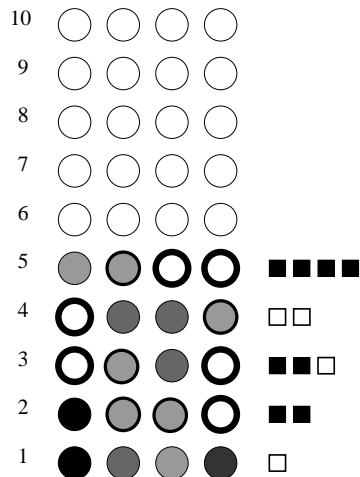


Mastermind

Implémentation et Résolution



Vous avez sûrement joué au Mastermind quand vous étiez petit. C'est un jeu de réflexion qui se joue à deux joueurs.

Un des participants choisit secrètement 4 boules de couleur parmi un ensemble de boules à 6 couleurs qu'il place dans un ordre précis. Ainsi il a $6^4 = 1296$ choix possibles.

Le but de l'autre joueur est de deviner la combinaison qu'à inventé son adversaire. Pour cela il dispose de 10 questions. A chaque tour, il montre une combinaison de 4 boules et l'adversaire lui répond combien de boules sont bien placées et combien ont la bonne couleur mais sont mal placées, en représentant par des pions noirs le nombre de boules bien placées et par des pions blancs le nombre de mal placées (on peut remarquer que la somme des deux ne fait pas toujours 4).

Un exemple de partie est représentée ci contre. La personne qui devait deviner a trouvé la réponse en 5 coups.

Nous allons dans ce TD essayer tout d'abord de programmer un adversaire qui répond aux question qu'on lui pose puis dans un deuxième temps de programmer un adversaire qui pose les bonnes questions et arrive à trouver la solution.

Dans la suite du TD, une combinaison sera représentée par une suite d'entier à quatre éléments a, b, c, d où $1 \leq a, b, c, d \leq 6$. Le nombre de jetons bons/mauvais sera représenté par une suite à deux éléments noirs, blancs compris entre 0 et 4.

Exercice 1. Jouer contre la machine

a. Écrire une fonction `aleatoire := proc()` retournant une suite a, b, c, d correspondant à une combinaison valide. On pourra se servir de la fonction `rand(6)()` qui renvoie un entier entre 0 et 5.

b. Écrire une fonction `noir_blanc := proc(a,b,c,d, A,B,C,D)` qui prend en argument huit nombres et rend un couple `noir, blanc` correspondant aux pions bien placés (noir) et mal placés (blanc) dans le cas où la réponse serait A, B, C, D et la question posée par l'adversaire a, b, c, d .

Tester cette fonction sur quelques valeurs. Par exemple $1, 2, 3, 4, 4, 2, 2, 1$ doit rendre $1, 2; 1, 1, 2, 3, 2, 1, 2, 6$ doit rendre $2, 0$; ou encore $5, 4, 3, 2, 1, 3, 5, 3$ doit rendre $0, 2$

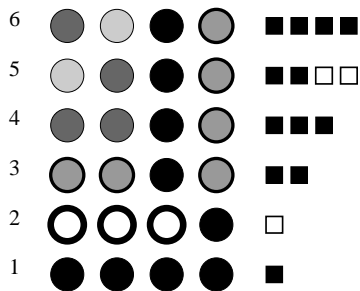
c. Écrire un programme permettant de jouer contre la machine au Mastermind. On pourra s'inspirer fortement du programme ci dessous :

```
master_mind := proc()
  local A,B,C,D,a,b,c,d, noir, blanc, gagne, i;
  A,B,C,D := aleatoire();
  gagne := false; i := 0;
  while i < 10 and not gagne do
    ligne = readline(-1); a,b,c,d := op(sscanf(ligne, "%d_%d_%d_%d"));
    noir, blanc := noir_blanc(a,b,c,d, A,B,C,D);
    if noir = 4
      then printf("Gagne!\n"); gagne := true;
      else printf("%d_%d_%d_%d donne %d bons et %d mals placés\n",
                  %a,b,c,d, noir, blanc);
    fi;
  od;
  if not gagne then printf("perdu\n"); fi;
end;
```

Exercice 2. A l'ordinateur de jouer

On munit l'ensemble des combinaisons de l'ordre lexicographique (*ie* l'ordre du dictionnaire)

$$(a, b, c, d) < (e, f, g, h) \text{ ssi } a < e \text{ ou } a = e \text{ et } (b < f \text{ ou } b = f \text{ et } (...))$$



L'idée de l'algorithme de résolution du problème est d'énumérer toutes les configurations possibles dans l'ordre. On commence donc par essayer la configuration (1,1,1,1). L'autre joueur nous donne alors une réponse (noir,blanc) ce qui élimine un certain nombre de solutions possibles. On énumère ensuite toutes les configurations dans l'ordre en s'arrêtant à première configuration dont le couple (noir,blanc) corresponde aux valeurs données par l'adversaire.

Un exemple est fournis sur le dessin ci contre. La première réponse montre qu'il y a un 1 de bien placé. La configuration suivante ayant un unique 1 est 1,2,2,2. L'adversaire répondant qu'il y a une boule de mal placée, la configuration suivante devient 3,1,3,3, etc... Au bout de 6 coups, l'adversaire trouve la solution.

a. Écrire une fonction `coup_suivant := proc(a,b,c,d)` qui prend en argument 4 nombres compris entre 1 et 6 et qui rend la suite de quatre entiers correspondants à l'élément suivant selon l'ordre lexicographique (on considèrera que le successeur de 6,6,6,6 est 7,1,1,1).

Une partie est constituée de la liste des coups joués. Par la suite on stockera la partie dans une liste `[a,b,c,d, noir,blanc]`, `[[a',b',c',d', noir', blanc']`, `[[a'', b'', c'', d'', noir'', blanc'']...]]`. Par exemple, la partie ci dessus est avant la ligne 5 : `[[3,1,4,4, 3,0],[3,1,3,3, 2,0], [[1,2,2,2, 0,1],[1,1,1,1, 1,0],[]]]`

b. Écrire une fonction `correcte := proc(a,b,c,d, partie)` testant la validité d'une nouvelle combinaison en fonction des coups joués précédemment. On utilisera une fonction définie par récurrence.

c. Implémenter l'algorithme de résolution du mastermind. On pourra soit partir de zéro pour les plus motivés, soit s'inspirer de l'algorithme suivant :

```
local a,b,c,d,noir , blanc , partie ,gagne , ligne ;
partie := [] ; gagne := false ; ligne := 0 ;
a,b,c,d := 1,1,1,1 ;

while ligne < 10 and not gagne do
  printf("je joue %d %d %d %d\n", a,b,c,d) ;
  ligne := readline(-1) ; noir , blanc := op(scanf(ligne , "%d %d")) ;
  if noir = 4 then gagne := true ;
  else
    partie := [[a,b,c,d, noir , blanc] , partie] ;
    while a <> 7 and not correcte (a,b,c,d, partie)
      do a,b,c,d := coup_suivant(a,b,c,d) ; od ;
    if a=7 then printf("il n'y a pas de solution\n") ; gagne:=true ; fi ;
  fi ;
  ligne := ligne + 1 ;
od ;
```

d. Tester votre algorithme sur des exemples choisis par vous.

Exercice 3. Pour aller plus loin

Sur des exemples précis, cet algorithme semble bien se comporter (il trouve généralement la solution en 6 coups). Néanmoins, il est bon de l'étudier d'avantage. Comme le nombre de configuration n'est que de 6^4 , nous allons toutes les étudier.

a. Modifier l'algorithme ci dessus en un algorithme `master_seul := proc(a,b,c,d)` pour le faire jouer tout seul.

b. Utiliser la question précédente pour faire des statistiques sur le nombre de coups que met cet algorithme à trouver la solution (étudier la moyenne, le pire cas, ...)

c. Si le nombre de couleurs n'est plus 6 mais n , combien de questions pensez vous que cet algorithme doit poser pour trouver la réponse à chaque fois (essayer par exemple avec 7 ou 8) ?