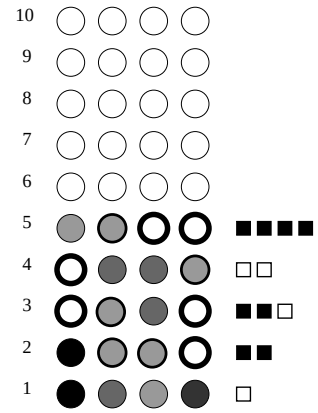
# *Mastermind*

## Implémentation et Résolution



Le Mastermind est un jeu de réflexion qui se joue à deux joueurs.

Un des participants choisit secrètement 4 boules dans un ensemble de boules à 6 couleurs qu'il place dans un ordre précis (remarque : il a donc $6^4 = 1296$ choix possibles).
Le but de l'autre joueur est de deviner la combinaison choisie par son adversaire. Pour cela il dispose de 10 essais. A chaque tour, il montre à son adversaire une combinaison de 4 couleurs et celui-ci lui répond combien de boules sont bien placées et combien ont la bonne couleur mais sont mal placées. Sur le dessin ci-contre, j'ai représenté les bien placées sont représentées par un carré noir et les mal placées par un carré blanc.

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

Ce TD est divisé en plusieurs parties. Dans une première partie nous programmerons un adversaire qui choisit une combinaison et vous la fait deviner un nombre puis nous chercherons une stratégie permettant à l'ordinateur de trouver la solution. La dernière partie est consacrée à l'étude et à l'amélioration de cette stratégie.

En maple, une combinaison de 4 boules sera représentée par une suite de quatre entiers `a,b,c,d` où $0 \leq \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d} \leq 5$.

### Exercice 1. Jouer contre la machine

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

**b.** Écrire une fonction `bon_mal := proc( a,b,c,d, A,B,C,D)` qui prend en argument huit nombres et rend un couple `bien,mal` correspondant respectivement au nombre de boules bien placées et au nombre de boules de la bonne couleur mais mal placées dans le cas où la combinaison à trouver serait A,B,C,D et la question posée par l'adversaire a,b,c,d.

On fera attention à ne pas compter deux fois chaque boules : une boule ne peut être bien et mal placée à la fois.

**c.** Programmer un programme qui vous fait jouer au mastermind en vous inspirant de l'algorithme ci-dessous :

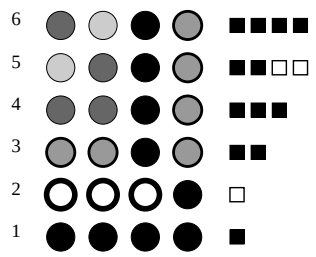
```
Tirer une combinaison aléatoire
l_adversaire_n_a_pas_gagne <- vrai;
nombre_de_coup <- 0;

tantque (nombre_de_coup < 10 et l_adversaire_n_a_pas_gagne) faire
   Demander une nouvelle combinaison.
   Calculer le couple (bien,mal) correspondant à sa combinaison.
   Si le nombre de bien placées est 4
     alors il a gagné.
     sinon lui afficher les nombres (bien,mal).
finfaire

Si nombre_de_coup >= 10 et l_adversaire_n_a_pas_gagne
   alors il a perdu
```

**Remarque :**
- pour demander une nouvelle combinaison, on pourra se servir de la commande :
  `ligne := readline(-1) ; a,b,c,d := op(sscanf(ligne,"%d %d %d %d")) ;`
- pour afficher le résultat, on se servira de `print` ou `printf`

## Exercice 2. A l'ordinateur de jouer

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

$(a,b,c,d) < (e,f,g,h)$ *ssi* $a < e$ ou $\Big(a = e$ et $b < f$ ou $\Big(b = f$ et $[\ldots]\Big)\Big)$



L'idée de l'algorithme de résolution du problème est d'énumérer toutes les combinaisons possibles dans l'ordre. On commence donc par essayer la combinaison (0,0,0,0). Suivant la réponse de l'autre joueur, un certain nombre de combinaisons deviennent impossible.
On va donc proposer à notre adversaire la prochaine configuration possible. On recommence jusqu'à trouver la bonne réponse.

Un exemple est fournis sur le dessin ci contre. La première réponse montre qu'il y a un 0 de bien placé. La combinaison suivante ayant un unique 0 est 0,1,1,1. L'adversaire répondant qu'il y a une boule de mal placée, la combinaison possible suivante est 2,0,2,2, 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 0 et 5 et qui rend la suite de quatre entiers correspondants à l'élément suivant selon l'ordre lexicographique.

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

**b.** Écrire une fonction `correcte := proc(a,b,c,d, partie)` testant la validité d'une combinaison en fonction des coups joués précédemment.

**c.** En s'inspirant des lignes ci-dessous, implémenter l'algorithme en maple

```
mon_coup <- 0,0,0,0;
nombre_de_coup_joues <- 0;        partie <- []

tantque nombre_de_coup_joues < 10 faire
   dire à l'humain: je joue la combinaison mon_coup
   ligne := readline(-1); bon,mal := op(scanf(ligne,"\%d \%d"));
   si bon = 4 alors
      j'ai gagné!
   sinon
      ajouter le coup courant à la partie.
      tantque mon_coup ne pourrait pas être la solution faire
         si coup_suivant existe
             alors  mon_coup = coup_suivant(mon_coup);
             sinon écrire ``il n'y a pas de solution!''.
      finfaire
   nombre_de_coup_joues = nombre_de_coup_joues + 1
finfaire
```

## Exercice 3. Pour aller plus loin

Sur des exemples précis, cet algorithme semble trouver la bonne réponse mais on peut se poser différentes questions sur la viabilité (nombre de coups en moyenne, dans le pire cas,etc...). Le nombre de combinaison n'étant que de $6^4$, il est possible de toutes les étudier.

**a.** Modifier votre programme pour le faire jouer tout seul sur une combinaison fixés `a,b,c,d` et qu'il vous rende le nombre de coups qu'il a mis pour trouver

**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.** Refaire les même statistiques mais en remplaçant la combinaison de départ par une nouvelle combinaison (par exemple 4,4,3,1) et étudier les performances. Est-ce mieux ?