Mastermind
Correction

```
# Cette fonction rend une configuration aléatoire (soit 4 entiers entre 1 et 6)
aleatoire := proc()
   rand(6)()+1, rand(6)()+1,rand(6)()+1,rand(6)()+1;
end:

# Cette fonction calcul le nombre de boules bien placées (noir) et mal (blanc)
noir_blanc := proc(a,b,c,d, A,B,C,D)
   local e,f,g,h, E,F,G,H, noir, blanc;
   e,f,g,h := a,b,c,d;
   E,F,G,H := A,B,C,D;

   noir,blanc := 0,0;

 # les e := -1, E := -1, ... sont des marqueurs mis pour éviter qu'un
  # pion bien placé soit recompté ensuite comme un pion mal placé.
  # Dans la suite de la fonction, on teste toujours e<>-1 pour vérifier que
  # le pion n'a pas déjà été compté.

  # On commence par regarder quels boules sont bien placées.

   if (e=E) then e:=-1; E := -1; noir := 1; fi;
   if (f=F) then f:=-1; F := -1; noir := noir+1; fi;
   if (g=G) then g:=-1; G := -1; noir := noir+1; fi;
   if (h=H) then h:=-1; H := -1; noir := noir+1; fi;

  # Puis on regarde les boules mal placées

   if(e=F and e <>-1 and F <>-1) then e:=-1; F := -1; blanc := 1; fi;
   if(e=G and e <>-1 and G <>-1) then e:=-1; G := -1; blanc := blanc + 1; fi;
   if(e=H and e <>-1 and H <>-1) then e:=-1; H := -1; blanc := blanc + 1; fi;
   if(f=E and f <>-1 and E <>-1) then f:=-1; E := -1; blanc := blanc + 1; fi;
   if(f=G and f <>-1 and G <>-1) then f:=-1; G := -1; blanc := blanc + 1; fi;
   if(f=H and f <>-1 and H <>-1) then f:=-1; H := -1; blanc := blanc + 1; fi;
   if(g=E and g <>-1 and E <>-1) then g:=-1; E := -1; blanc := blanc + 1; fi;
   if(g=F and g <>-1 and F <>-1) then g:=-1; F := -1; blanc := blanc + 1; fi;
   if(g=H and g <>-1 and H <>-1) then g:=-1; H := -1; blanc := blanc + 1; fi;
   if(h=E and h <>-1 and E <>-1) then h:=-1; E := -1; blanc := blanc + 1; fi;
   if(h=F and h <>-1 and F <>-1) then h:=-1; F := -1; blanc := blanc + 1; fi;
   if(h=G and h <>-1 and G <>-1) then h:=-1; G := -1; blanc := blanc + 1; fi;

   noir,blanc;
end:
```

La fonction `mastermind` est la même que celle écrite dans l'énoncé du TD.

```
# Cette fonction calcule le quadruplet suivant
coup_suivant := proc(a,b,c,d)
   local A,B,C,D;
   D := d+1; C := c; B := b; A := a;
   if D = 7 then  D := 1; C := c+1;
      if C = 7 then C := 1; B := b+1;
         if B=7 then B := 1; A := a+1;
   fi;fi;fi;
   A,B,C,D;
end:
```

```
# Pour regarder si un coup est valide, on procède par récurence en
# supposant que partie est de la forme [ [A,B,C,D, noir, blanc], partie2] où
# partie2 est une partie valide
correcte := proc (a,b,c,d, partie)
   local e,f,A,B,C,D, noir,blanc, no,bl;
   if nops(partie) = 0 then true;
   else
     e,f := op(partie);
     A,B,C,D, noir,blanc := op(e);
     no,bl := noir_blanc(a,b,c,d,A,B,C,D);
     no = noir and bl = blanc and correcte (a,b,c,d, f);
   fi;
end:

# Cette fonction implémente l'algorithme décrit dans l'énoncé.
trouve := proc()
  local a,b,c,d,noir,blanc, partie,gagne, ligne;

  #introduire ici l'algorithme décrit dans l'énoncé.

  if gagne then printf("gagne"); fi;
end:

simulation := proc(A,B,C,D)
  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
# La seule différence avec l'algorithme précédant est ici : la machine ne
# demande plus son avis à l'être humain mais calcule elle-même ses coups
      noir,blanc := noir_blanc(a,b,c,d,A,B,C,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; # la valeur retournée est le nombre de coups
  od;
end:

# Cette algorithme appelle l'algorithme précédant sur toutes les valeurs
# que peuvent prendre les configurations.
stat := proc()
   local a,b,c,d, n; global s;
   s[1],s[2],s[3],s[4],s[5],s[6],s[7],s[8],s[9],s[10] := 0,0,0,0,0,0,0,0,0,0;
   a,b,c,d := 1,1,1,1;
   while( a <> 7) do
      n := simulation(a,b,c,d); s[n] := s[n] +1;
      a,b,c,d := coup_suivant(a,b,c,d); od;
   seq(s[i],i=1..10);
end:

stats();
                                  1,4,25,108,305,602,196,49,6,0;
```

Note : on peut remarquer qu'en moyenne, le nombre de coups nécessaire à l'algorithme pour trouver la solution est 4.77, dans le pire cas cet algorithme trouve une solution en 9 coups (c'est à dire qu'il gagne tout le temps).

Si on met n couleurs et non plus 6 comme ici, l'algorithme met n+3 coups à trouver une solution dans le pire des cas.