## Coupure

```
/* Les voyelles                                                          */
voyelle(a).
voyelle(e).
voyelle(i).
voyelle(o).
voyelle(u).
voyelle(y).
/* Les consonnes                                                         */
consonne(C):- \+ voyelle(C).
/* Les groupes ch, ph, gn, th sont inseparables */
non_separable(c,h).
non_separable(p,h).
non_separable(g,n).
non_separable(t,h).
/* Le groupe [C,l] est inseparable si C est une consonne differente de l.   */
non_separable(C,l):-
       C \== l,
       consonne(C).
/* Le groupe [C,r] est inseparable si C est une consonne differente de r.   */
non_separable(C,r):-
       C \== r,
       consonne(C).
/* Une consonne placee entre deux voyelles introduit une nouvelle syllabe.  */
coupure([V1,C1,V2|Suite],[V1,-,C1|Reste]):-
       voyelle(V1),
       consonne(C1),
       voyelle(V2),
       coupure([V2|Suite],Reste).
/* De deux consonnes placees entre deux voyelles, la premiere appartient a  */
/* la syllabe precedente, la seconde a la suivante.                         */
/* Exception: le cas des consonnes non separables.                          */
coupure([V1,C1,C2,V2|Suite],[V1,C1,-,C2|Reste]):-
       voyelle(V1),
       consonne(C1),
       consonne(C2),
       \+ non_separable(C1,C2),
       voyelle(V2),
       coupure([V2|Suite],Reste).
/* L'exception sus-citee (on coupe avant les deux consonnes).               */
coupure([V1,C1,C2|Suite],[V1,-,C1|Reste]):-
       voyelle(V1),
       consonne(C1),
       consonne(C2),
       non_separable(C1,C2),
       coupure([C2|Suite],Reste).
/* Quand il y a trois consonnes consecutives a l'interieur d'un mot,        */
/* ordinairement les deux premieres terminent une syllabe, l'autre commence */
/* la suivante sauf si les deux dernieres consonnes sont inseparables.      */
coupure([C1,C2,C3|Suite],[C1,C2,-|Reste]):-
       consonne(C1),
       consonne(C2),
       consonne(C3),
       \+ non_separable(C2,C3),
       coupure([C3|Suite],Reste).
coupure([C1,C2,C3|Suite],[C1,-,C2|Reste]):-
       consonne(C1),
       consonne(C2),
       consonne(C3),
       non_separable(C2,C3),
       coupure([C3|Suite],Reste).
/* Cas terminal                                                          */
coupure([],[]).
/* Si il n'y a pas de separation. (ramasse miettes)                      */
/* Cette solution ou le cas complementaire est a la fin est efficace mais   */
/* introduit un determinisme genant pour l'evolution du programme        */
coupure([L1|Suite],[L1|Reste]):-
       coupure(Suite,Reste).
/*  Solution  2   */
coupure1([V1,C1,V2|Suite],[V1,-,C1|Reste]):-
       voyelle(V1),
       consonne(C1),
       voyelle(V2),
       coupure1([V2|Suite],Reste).
coupure1([V1,C1,C2,V2|Suite],[V1,C1,-,C2|Reste]):-
       voyelle(V1),
```

```
       consonne(C1),
       consonne(C2),
       \+ non_separable(C1,C2),
       voyelle(V2),
       coupure1([V2|Suite],Reste).
coupure1([V1,C1,C2|Suite],[V1,-,C1|Reste]):-
       voyelle(V1),
       consonne(C1),
       consonne(C2),
       non_separable(C1,C2),
       coupure1([C2|Suite],Reste).
coupure1([C1,C2,C3|Suite],[C1,C2,-|Reste]):-
       consonne(C1),
       consonne(C2),
       consonne(C3),
       \+ non_separable(C2,C3),
       coupure1([C3|Suite],Reste).
coupure1([C1,C2,C3|Suite],[C1,-,C2|Reste]):-
       consonne(C1),
       consonne(C2),
       consonne(C3),
       non_separable(C2,C3),
       coupure1([C3|Suite],Reste).
coupure1([L1|Suite],[L1|Reste]):-
       \+ coupable([L1|Suite]),
       coupure1(Suite,Reste).
coupable([L1,L2,L3|_]):-
       voyelle(L1),
       consonne(L2),
       voyelle(L3).
coupable([L1,L2,L3,L4|_]):-
       voyelle(L1),
       consonne(L2),
       consonne(L3),
       (voyelle(L4);non_separable(L2,L3)) .
coupable([L1,L2,L3|_]):-
       consonne(L1),
       consonne(L2),
       consonne(L3).
coupure1([],[]).
/*
| ?- coupure1([s,t,r,u,c,t,u,r,e],L).
L = [s,-,t,r,u,c,-,t,u,-,r,e]
| ?- coupure1([c,h,a,l,e,u,r],H).
H = [c,h,a,-,l,e,u,r]
| ?-  coupure1([e,l,e,l,l,e],X).
X = [e,-,l,e,l,-,l,e]
*/
```

## canmiss

```
canmiss(Liste_Etats) :-
       etat_initial(E),
       solve([E],Liste_Etats).
solve([Ec|E],[Ec|E]):-
       etat_succes(Ec).
solve([[Mc,Cc,Rc]|S],L):-
       genere_etat_valide([Mc,Cc,Rc],[Mn,Cn,Rn]),
       \+ member([Mn,Cn,Rn],S),
       solve([[Mn,Cn,Rn],[Mc,Cc,Rc]|S],L).
genere_etat_valide([Mc,Cc,Rc],[Mn,Cn,Rn]):-
       select(M,C,Mc,Cc),
       Mo is Mc-M,
       Co is Cc -C,
       ok(Mo,Co),
       Mn is 3 - Mo,
       Cn is 3 - Co,
       ok(Mn,Cn),
       succ(Rc,Rn).
select(M,C,Mc,Cc):-
       enum(M,0,Mc),
       enum(C,0,Cc),
       MC is M+C,
       MC >= 1,
       MC =< 2.
etat_succes([3,3,droite]).
etat_initial([3,3,gauche]).
succ(gauche,droite).
succ(droite,gauche).
ok(0,C) :- C > 0.
```

```
ok(M,C) :- M >= C.
/* Utilitaires */
enum(X,_,X).
enum(X,Y,Z):- Z > Y,Z1 is Z-1, enum(X,Y,Z1).
```

## tracassin

```
tr(P) :-
        etat_i(E),
        transfo([E],P),
       write(P).
transfo([Ef|L],[Ef|L]):-
        etat_f(Ef).
transfo([Ec|La],P):-
        ope(Ec,Es,Op),
       transfo([Es,Ec|La],P).
ope([n,l|L],[l,n|L],r1).
ope([l,b|L],[b,l|L],r2).
ope([n,b,l|L],[l,b,n|L],r3).
ope([l,n,b|L],[b,n,l|L],r4).
ope([C|L1],[C|L2],R) :-
       ope(L1,L2,R).
/* Jeu d'essai */
etat_i([n,n,n,n,n,l,b,b,b,b,b]).
etat_f([b,b,b,b,b,l,n,n,n,n,n]).
```