```
/*              Jeu des tas de cailloux */

affiche([N1|N2]) :-   nl,   nl,   affiche_ligne(N1),   affiche(N2),   !.
affiche([]).

affiche_ligne(0) :- !.
affiche_ligne(Nombre1) :-   writes("  Û"),   Nombre2 is Nombre1 - 1,   affiche_ligne(Nombre2).

regles(R) :- R = ["                    Jeu des tas de cailloux"," ",   " Le jeu se compose d'un nombre N (pas trop grand : =< 5) de tas de cailloux"," disposés horizontalement et dont l'initialisation est faite par le joueur.",   " A chaque tour de jeu, on peut enlever 1, 2 ou 3 cailloux dans un même tas"," à condition que le tas soit assez gros.", " Pour prendre 1 caillou dans le tas 4, il faut entrer [4,1].",   " La prise du coup précédent est indifférente."," "," Le gagnant est celui qui ramasse le dernier caillou."," "].


/* le format des états est le suivant : [1,2,5,3,1] veut dire qu'il y a 5 tas contenant respectivement 1,2,5,3, et 1 cailloux. */

/* le format des actions est le suivant : [4,3] veut dire qu'on prend 3 cailloux dans le tas 4 */


possibilites(E,L) :- coups_possibles(E,L,1).

/* coups_possibles donne les coups possibles en parcourant tous les tas */
/* prédicat logique : la liste des coups possibles associée aux tas qui vont
   du Numero_tas-ième tas au dernier tas est la liste des coups possibles
   associée au Numero_tas-ième tas, concaténée avec la liste des coups possibles
   associée aux tas qui vont du (Numero_tas+1) au dernier. */

coups_possibles([],[],_):-!.
coups_possibles([X|E2],L,Numero_tas) :-   Num2 is Numero_tas + 1,   coups_possibles(E2,L1,Num2),   min(X,3,Min),
   ajoute_coups_un_tas(Numero_tas,Min,L1,L).

/* ajoute_coups_un_tas concatène la liste des coups correspondant … un tas avec la liste L */
/* le 3ème paramètre N du prédicat est le nombre maximal de cailloux qu'on a  le droit d'enlever dans le tas : on pourra donc jouer, si Numero_tas=1 et  N=3,[1,3],[1,2],[1,1].  */

/* la liste des coups correspondant à  N est le coup [Numero_tas,N] concatènée avec la liste des coups correspondant à N-1. */

ajoute_coups_un_tas(Numero_tas,0,L,L) :- !.
ajoute_coups_un_tas(Numero_tas,N,L,[[Numero_tas,N]|L2]) :-   N2 is N - 1,    ajoute_coups_un_tas(Numero_tas,N2,L,L2).

/* calcul du minimum de 2 nombres (sert … calculer le nombre maximum de cailloux que l'on peut enlever d'un tas, ,gal au minimum du nombre de cailloux du tas et de 3 */

min(N1,N2,N1) :-   N1 =< N2,   !.
min(N1,N2,N2).

/* prendre Prise cailloux dans le tas numero Numero_tas revient … prendre Prise caillou dans le tas numero Numero_tas-1 de l'ensemble des tas qui vont du numero Numero_tas+1 au dernier tas  */

modifie([X1|E1],[1,Prise],[X2|E1]) :-   X2 is X1 - Prise,   !.
modifie([X|E1],[Numero_tas,Prise],[X|E2]) :-   Num2 is Numero_tas - 1,   modifie(E1,[Num2,Prise],E2).

/* exemple : si on veut N=4 tas, l',tat initial est [1,3,5,7] */
/* Nb_it est le nombre d'it,rations restant … faire */

initialisation(1,N,[N]) :- !.
initialisation(Nb_it,N,[N|E]) :-   Nb_it2 is Nb_it - 1,   N2 is N + 2,   initialisation(Nb_it2,N2,E).

init(E) :-   repeat,   writes("Entrez le nombre de tas (entre 1 et 5):"),   read(S),   integer(S),   S > 0,   S =< 5,
          initialisation(S,1,E),   !.

/* le seul état terminal perdant est le cas où il ne reste aucun caillou,   c'est à dire la liste E qui ne contient que des 0 */

termine(E,[0,nil]) :- que_des_z,ros(E).

que_des_z,ros([0|E]) :-   que_des_z,ros(E),   !.
que_des_z,ros([]).
```