Des jeux pour une introduction à l'algorithmique en seconde

Renée De Graeve

2 septembre 2016

Remerciements

Je remercie :
— Bernard Parisse pour ses précieux conseils et ses remarques sur ce texte,

Table des matières

Index

Introduction

Qu'est-ce qu'un algorithme ?
Un algorithme est la description d'une méthode pour effectuer une tâche précise : l'algorithme prend en compte des données de départ et donne les instructions à effectuer pour parvenir au résultat cherché.
La façon dont les instructions sont données importe peu, mais il est important que ces instructions ne soient pas ambigües et soient valables pour toutes les données de départ.
Quel langage choisir ?
Cela dépend à qui l'algorithme est destiné :

- Si c'est à Monsieur "Toutlemonde" on utilisera sa langue habituelle,
- Si c'est à un ordinateur on utilisera un langage de programmation particulier pour cette machine.

Ici, on suppose que l'on va traduire les algorithmes en des langages de même type qui seront compréhensibles par différentes machines.
On choisira le langage du logiciel `Xcas` pour traduire les algorithmes et ainsi de pouvoir les faire fonctionner.

Chapitre 1

Savoir minimum pour programmer avec Xcas

1.1 Installation de Xcas

Le programme Xcas est un logiciel libre écrit en C++, (disponible sous licence GPL). La version à jour se récupère sur :
http : //www − fourier.ujf − grenoble.fr/~parisse/giac_fr.html ou
ftp://fourier.ujf-grenoble.fr/xcas
où l'on trouve le code source (giac.tgz) ou des versions précompilées pour Linux (PC ou ARM), Windows, Mac OS.

1.2 Pour écrire une fonction ou un programme

Voir aussi le chapitre 6.
Il faut :

- choisir la syntaxe Xcas,
  — soit avec le menu Cfg->Mode(syntax)->xcas,
  — soit en ouvrant la fenêtre de configuration en appuyant sur la barre Config et choisir Xcas dans Prog style,
- ouvrir un niveau éditeur de programme soit en tapant Alt+p, soit avec le menu Prg->Nouveau programme. Il contient déjà le :; qui doit terminer le programme.
- taper la fonction en terminant chaque instruction par ;. Les noms de cette fonction et de ses arguments ne doivent pas déjà être utilisés par Xcas. On prendra donc l'habitude de commencer le nom des fonctions par une Majuscule pour avoir moins de risques de définir une fonction qui existe déjà dans Xcas : de toutes les façons, dans un niveau éditeur de programmes, les mots clés apparaissent en bleu et les nomms de commandes apparaissent en brun.
- appuyer sur OK ou sur F9, pour compiler le programme.
- pour exécuter le programme, il suffit de taper son nom en mettant entre parenthèses les valeurs des paramètres séparées par des virgules.

1.3 Le menu Ajouter d'un niveau éditeur de programme

Ce menu vous permet d'avoir facilement la syntaxe d'une fonction, d'un test et des différentes boucles. Une fonction s'écrit avec la syntaxe suivante :

```
f(x,y):={
  local z,a,b,...,val;
  instruction1;
  .....
  instructionk;
 retourne val;
}:;
```

On termine par :; pour que la réponse à une compilation reussie soit Done ou par ; pour que la réponse soit la traduction du programme après la compilation.

Par exemple :
L'algorithme d'Euclide pour calculer le pgcd de deux nombres a et b. On remarquera que != est un opérateur booléen qui teste la différence et que := est utilisé pour faire une affectation.

```
pgcd(a,b):={
 local r;
 tantque b!=0 faire
  r:=irem(a,b);
  a:=b;
  b:=r;
 ftantque
 retourne a;
}:;
```

On appuie sur OK ou sur F9 et on obtient // Parsing pgcd// Success compiling pgcd
On tape pgcd(138,51) et on obtient 3.

1.4 Un exemple : plusieurs versions d'une fonction

Un fabricant de chaussures vous demande de programmer une fonction qui donne la pointure p selon la longueur L en cm du pied.
On sait que si le pied a comme longueur L cm la pointure p est donnée par la formule $p = 3L/2$.
Voici votre premier programme :
On tape dans un niveau éditeur de programme :

```
pointure(L):={
  local p;
  p:=(L*3)/2;
  retourne evalf(p,2);
}:;
```

Vous remarquerez :

— la forme de la définition d'une fonction son nom, ses arguments mis entre parenthèses, puis := et le corps de la fonction délimité par { et } qui contient les instructions. Ici on définit la fonction pointure qui a comme paramètre L et qui est composée de 2 instructions : une affectation := et la valeur du résultat précédée de retourne. Cette fonction utilise une variable locale qui est p,
— le mot reservé local qui sert à définir les variables locales qui sont utilisées dans le programme,
— l'instruction retourne qui fait sortir du programme et qui renvoie la valeur de la fonction,
— le := qui affecte une valeur à une variable : ici 3*L/2 est stocké dans p,
— les ; qui terminent chaque instruction,
— la fonction evalf(a,d) de Xcas qui renvoie la valeur décimale de a avec d décimales.

On appuie sur OK ou sur F9 et on obtient écrit en bleu :
// Parsing pointure// Success compiling pointure
et en réponse Done car on a mis :; à la fin de la définition de la fonction. Si on met ; à la fin de la définition de la fonction on a alors en réponse :

```
(L)->
{ local p;
  p:=(L*3)/2;
  retourne(evalf(p,2));
}
```

On tape :

```
pointure(31)
```

On obtient :

```
46.5
```

On tape :

```
pointure(26.7)
```

On obtient :

```
40.5
```

On tape :

```
pointure(26.7)
```

On obtient :

```
40.05
```

On tape :

```
pointure(25.3)
```

On obtient :

```
37.95
```

Si la réponse ne vous satisfait pas parce que les pointures sont en général des nombres entiers ou des nombres entiers plus un demi, il faut améliorer ce programme en renvoyant l'entier ou l'entier plus un demi le plus proche.
Pour cela on va utiliser `round(b)` de `Xcas` qui arrondit `b` à l'entier le plus proche : `round(2p)/2` sera alors le résultat.
On tape dans un niveau éditeur de programme :

```
pointure0(L):={
  local p;
  p:=(L*3)/2;
  retourne round(2p)/2;
}:;
```

Si la réponse ne vous satisfait encore pas parce que `pointure(25.5)` renvoie `77/2` et non `38.5`, il faut améliorer ce programme en renvoyant soit un nombre décimal : le résultat sera alors `round(2p)/2.0`. En effet si dans un calcul il y a un nombre décimal, `Xcas` renvoie un nombre décimal. Ainsi, `76/2` est le nombre entier `38` alors que `76/2.0` est le nombre décimal `38.0`, de même `77/2` est le nombre rationnel `77/2` alors que `77/2.0` est le nombre décimal `38.5`.
On tape dans un niveau éditeur de programme :

```
pointure1(L):={
  local p;
  p:=(L*3)/2;
  retourne round(2p)/2.0;
}:;
```

Si la réponse ne vous satisfait encore pas parce que sur les chaussures on met par exemple 39 ou 39 1/2 et non 39.0 et 39.5, il faut décomposer `round(2p)/2` :
Si `round(2p)` est pair et vaut `2n`, on renvoie `n` et
si `round(2p)` est impair et vaut `2n+1` on renvoie `n+0.5`,
On tape dans un niveau éditeur de programme :

```
pointure2(L):={
  local p,a;
  p:=(L*3)/2;
  a:=round(2p);
  si est_pair(a) alors retourne iquo(a,2);fsi;
  retourne iquo(a,2),1/2;
}:;
```

ou

```
pointure2bis(L):={
  local p,a;
  p:=(L*3)/2;
  a:=round(2p);
  si est_pair(a)alors retourne iquo(a,2);
  sinon
```

```
  retourne iquo(a,2),1/2;
  fsi;
}:;
```

On remarquera :

- l'instruction `est_pair` qui teste si un nombre entier est pair : elle renvoie un booléen,
- l'instruction conditionnelle `si ...alors... fsi` `pointure2` et `si ...alors ...sinon...fsi` de `pointure2bis`. `pointure2` et `pointure2bis` renvoie le même résultat car l'instruction `retourne` fait sortir du programme. Le programme `pointure2` semble plus lisible,
- l'instruction `iquo(a,b)` qui renvoie le quotient euclidien de `a` par `b`,
- la nature de la réponse qui est soit un entier (par exemple `39`), soit une séquence de nombres exacts (par exemple `39,1/2`).

Si la réponse ne vous satisfait encore pas parce que sur les chaussures on met par exemple 39 ou 39 1/2 et non 39 ou 39,1/2, il faut renvoyer une chaîne de caractères.

```
pointure3(L):={
  local p,a;
  p:=(L*3)/2;
  a:=round(2p);
  si est_pair(a)alors retourne string(iquo(a,2));
  sinon
  retourne string(iquo(a,2))+" 1/2";
  fsi;
}:;
```

On remarquera :

- l'instruction `string(iquo(a,2))` qui transforme le nombre renvoyé par `iquo(a,2)` en une chaîne de caractères,
- `" 1/2"` qui représente la chaîne de caractères formées de 4 caractères : l'espace, "1", "/"et "2",
- l'instruction + qui est infixé et qui concaténe deux chaînes de caractères (on peut aussi utiliser :
  `concat(string(iquo(a,2)),"  1/2")`

Si la réponse ne vous satisfait encore pas, parce que sur les chaussures pour les pointures plus grandes que 33 on met par exemple 6 ou 6 1/2 et non 39 ou 39 1/2, il faut encore renvoyer une chaîne de caractères.

```
pointure4(L):={
  local p,a;
  p:=(L*3)/2;
  a:=round(2p);
  si a>=66 alors
  si est_pair(a) alors retourne string(iquo(a,2)-33);fsi;
  retourne string(iquo(a,2)-33)+" 1/2";
  fsi;
  si est_pair(a)alors retourne string(iquo(a,2));fsi;
  retourne string(iquo(a,2))+" 1/2";
 }:;
```

On remarquera que grâce à l'instruction `retourne`, on n'a pas écrit de `si...sinon ...fsi` emboités.
On remarquera aussi le test d'inégalité avec le signe >=.
Si la réponse ne vous satisfait encore pas parce que `pointure4(25.2)` renvoie `0 1/2` et non `1/2` il faut encore faire un test !

```
pointure5(L):={
  local p,a;
  p:=(L*3)/2;
  a:=round(2p);
  si a>=66 alors
  si est_pair(a) alors retourne string(iquo(a,2)-33);fsi;
  si iquo(a,2)==33 alors retourne " 1/2"; fsi;
  retourne string(iquo(a,2)-33)+" 1/2";
  fsi;
  si est_pair(a)alors retourne string(iquo(a,2));fsi;
  retourne string(iquo(a,2))+" 1/2";
 }:;
```

On remarquera que le test d'égalité est fait avec le signe == et on remarquera aussi que `string` renvoie l'expression évaluée sous la forme d'une chaîne de caractères.
Ouf ! j'espère que cela vous satisfait !

En résumé :
vous avez utilisé les instructions :
`local`
`si...alors...fsi`
`si...alors...sinon ...fsi`
`retourne`
`:=`
vous avez utilisé les tests :
`>=`
`==`
vous avez utilisé les fonctions de `Xcas` :
`evalf`
`round`
`iquo`
`est_pair`
`string`
`+`

1.5 Les instructions en fançais

Instructions en fançais	
affectation	`a:=2;`
entrée expression	`saisir("a=",a);`
entrée chaine	`saisir_chaine("a=",a);`
sortie	`afficher("a=",a);`
valeur retournée	`retourne(a);`
arrêt	`break;`
alternative	`si <condition> alors <inst> fsi;`
	`si <condition> alors <inst1> sinon <inst2> fsi;`
boucle pour	`pour j de a jusque b faire <inst> fpour;`
	`pour j de a jusque b pas p faire <inst> fpour;`
boucle répéter	`repeter <inst> jusqua <condition>;`
boucle tantque	`tantque <condition> faire <inst> ftantque;`
boucle faire	`faire <inst1> si <condition> break;<inst2> ffaire;`

1.6 Les instructions en comme en C++

Instructions comme en C++	
affectation	`a:=2;`
entrée expression	`input("a=",a);`
entrée chaine	`textinput("a=",a);`
sortie	`print("a=",a);`
valeur retournée	`retourne(a);`
arrêt	`break;`
alternative	`if (<condition>) {<inst>};`
	`if (<condition>) {<inst1>} else {<inst2>};`
boucle pour	`for (j:= a;j<=b;j++) {<inst>};`
	`for (j:= a;j<=b;j:=j+p) {<inst>};`
boucle répéter	`repeat <inst> until <condition>;`
boucle tantque	`while (<condition>) {<inst>};`
boucle faire	`do <inst1> if (<condition>) break;<inst2> od;`

1.7 Les instructions comme en Maple

Instructions comme en Maple	
alternative	`if <condition> then <inst> end;`
	`if <condition> then <inst1> else <inst2> end;`
boucle pour	`for j from a to b do <inst> end;`
	`for j from a to b by p do <inst> end;`
boucle tantque	`while <condition> do  <inst> end;`

1.8 Signification des signes de ponctuation

Signification des signes de ponctuation	
`.`	sépare la partie entière de la partie décimale
`,`	sépare les éléments d'une liste ou d'une séquence
`;`	termine chaque instruction d'un programme
`:;`	termine les instructions dont on ne veut pas l'affichage de la réponse
!	factorielle

1.9 Les opérateurs

Opérateurs	
`+`	addition
`-`	soustraction
`*`	mutiplication
`/`	division
`^`	puissance
`==`	teste l'égalité
`!=`	teste la différence
`<`	teste la stricte infériorité
`<=`	teste l'infériorité ou l'égalité
`>`	teste la stricte supériorité
`>=`	teste la supériorité ou l'égalité
`ou`	opérateur booléen infixé
`et`	opérateur booléen infixé
`non`	renvoie l'inverse logique de l'argument
`vrai`	est le booléen vrai ou true ou 1
`faux`	est le booléen faux ou false ou 0

1.10 La notion de séquences et de chaînes de caractères

Séquences	
`S:=a,b,c`	S est une séquence de 3 élements
`S:=NULL`	S est une séquence de 0 élement
`dim(S)`	renvoie le nombre d'éléments de S
`S[0]`	renvoie le premier élément de S
`S[n]`	renvoie le $n+1$ ième élément de S
`S[dim(S)-1]`	renvoie le dernier élément de S
`S,d`	ajoute l'élément `d` à la fin de la séquence S

Chaînes de caractères	
`S:="abc"`	S est une chaîne de 3 caractères
`S:=""`	S est une chaîne de 0 caractère
`dim(S)`	renvoie le nombre de caractères de S
`S[0]`	renvoie le premier caractère de S
`S[n]`	renvoie le $n+1$ unième caractère de S
`S[dim(S)-1]`	renvoie le dernier caractère de S
`S+d`	ajoute le caractère d à la fin de la chaîne S

1.11 Les fonctions utilisées

Fonctions d'arithmétique	
`irem(a,b)`	renvoie le reste de la division de a par b
`iquo(a,b)`	renvoie le quotient de la division de a par b

Fonctions mathématiques	
`evalf(t,n)`	renvoie l'évaluation de t avec n décimales
`floor(t)`	renvoie la partie entière t
`round(t)`	renvoie l'entier le plus proche de t
`abs(t)`	renvoie la valeur absolue de t
`sqr(t)`	renvoie la racine carrée de t

Fonctions de géométrie	
`A:=point(a,b)`	renvoie et dessine le point A de coordonnées (a,b)
`affichage=epaisseur_point_5`	3ième argument de `point` pour l'afficher avec une croix d'épaisseur 5
`droite(A,B)`	renvoie et dessine la droite AB
`triangle(A,B,C)`	renvoie et dessine le triangle ABC
`bissectrice(A,B,C)`	renvoie et dessine la bissectrice de l'angle A du triangle ABC
`angle(A,B,C)`	renvoie la valeur de la mesure (radians ou degrés) de l'angle A du triangle ABC
`mediane(A,B,C)`	dessine la médiane issue de A du triangle ABC
`mediatrice(A,B)`	dessine la médiatrice de AB
`carre(A,B)`	renvoie et dessine le carré direct de côté AB
`cercle(A,r)`	renvoie et dessine le cercle de centre A et rayon r
`cercle(A,B)`	renvoie et dessine le cercle de diamètre AB
`rayon(c)`	renvoie la longueur du rayon du cercle c
`centre(c)`	renvoie et dessine le centre du cercle c
`distance(A,B)`	renvoie la longueur de AB
`distance(A,d)`	renvoie la distance de A à la droite d
`inter(G1,G2)`	renvoie et dessine la liste des points de $G1 \cap G2$
`inter_unique(G1,G2)`	renvoie et dessine l'un des points de $G1 \cap G2$
`rotation(A,t,B)`	renvoie et dessine le point transformé de B par la rotation de centre A et d'angle t

Chapitre 2

Jeu sur la notion d'algorithme

2.1 Description

On forme des petits groupes d'élèves de taille p. Chaque groupe doit mettre au point une stratégie pour gagner au jeu suivant :

- Chaque groupe passe a tour de rôle devant le meneur de jeu.
- Le meneur de jeu distribue à chaque élève 10 cartons de numéro 0,1,....9.
- Le meneur de jeu annonce un nombre $n \leq 9p$ et les élèves du groupe ne doivent plus se concerter mais doivent appliquer leur stratégie en montrant un carton de façon à ce que le total des cartons fasse n.
- On répéte cela 10 fois de suite pour chaque groupe. Pendant le jeu, les autres élèves observent et notent : le nombre annoncé et les réponses. Ils peuvent ainsi attribuer une note entre 0 et 10 au groupe.

2.2 Le but

Lorsque tous les groupes ont joué, on demande à chaque groupe de décrire leur stratégie : on peut voir alors, si cette stratégie répond au problème, si elle a ète appliquée correctement par tous les membres du groupe, si elle est compréhensible, si il y des ambigüités, si elle est valable dans tous les cas...
Le but de cette mise en commun est d'aboutir à l'écriture de plusieurs agorithmes décrivant les bonnes stratégies (celles valables pour tous les nombres n et p) et d'avoir un langage compréhensible par tous et par `Xcas` !

2.3 Une correction possible avec `Xcas`

Avec `Xcas`, on aura besoin de :

- la notion d'entrée et de sortie `saisir` et `afficher` ou de la notion de fonction,
- la notion d'affectation `:=`
- de l'instruction conditionnelle `si ...alors ...sinon ...fsi`
- la commande `iquorem(a,b)` qui renvoie le quotient et le reste de la division euclidienne de `a` par `b` :
  `iquorem(46,7)` renvoie la liste `[6,4]`

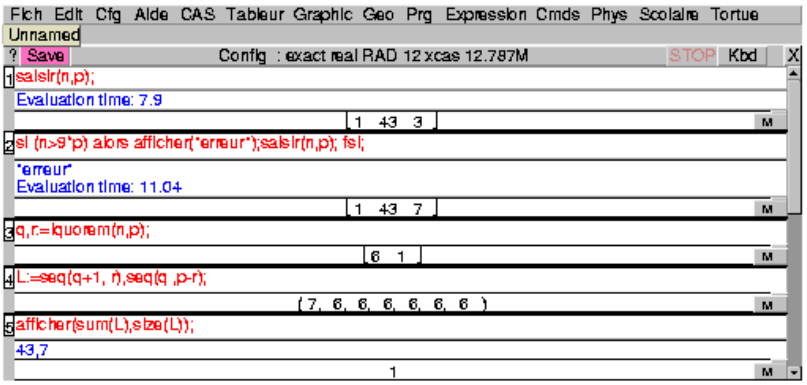
— la notion de séquence que l'on peut créer avec la commande `seq` ou l'opérateur infixé $:
`seq(3,5)` ou `3$5` renvoie `3,3,3,3,3`.

Il y a plusieurs stratégies possibles : on va en décrire 2 qui sont assez simples.

— Première stratégie,
On peut imaginer que n représente un nombre de cartes que l'on distribue à p joueurs jusqu'à épuisement des cartes.
Si $n = pq + r$ avec $0 \leq r < p$ alors les r premiers joueurs auront $q + 1$ cartes et les $p-r$ derniers joueurs auront q cartes ($n = (q+1)r+q(p-r)$).
Avec `Xcas`, on tape dans des lignes de commandes :

```
saisir(n,p);
si (n>9*p) alors
 afficher("erreur");
 saisir(n,p); fsi;
q,r:=iquorem(n,p);
L:=seq(q+1, r),seq(q ,p-r);
afficher(sum(L),size(L));
```

On entre par exemple 47 pour n, et 3 pour p, puis 47 pour n, et 7 pour p.
On obtient :

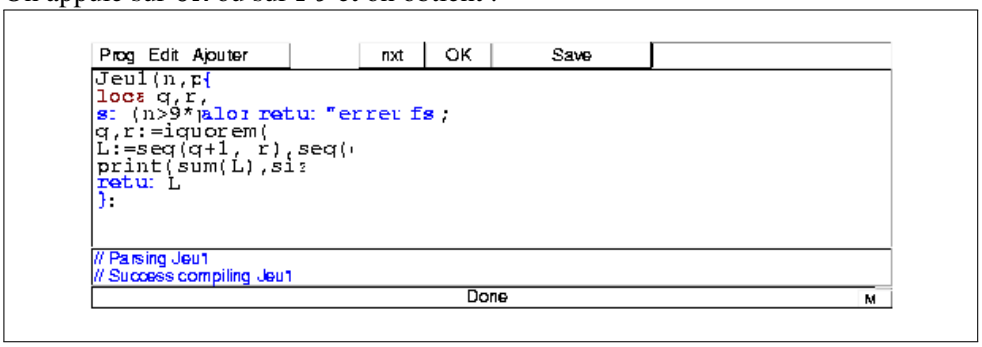


Ou encore, on écrit la fonction `Jeu1`. On prendra l'habitude de commencer le nom des fonctions par une Majuscule pour ne pas définir une fonction qui existe peut être déjà dans `Xcas`.
On tape :

```
Jeu1(n,p):={
 local q,r,L;
 si (n>9*p) alors retourne "erreur" fsi;
 q,r:=iquorem(n,p);
 L:=(q+1) $ r,q $ (p-r);
 afficher(sum(L),size(L));
 retourne L;
}:;
```

On renvoie la séquence L et on fait imprimer sa somme et sa taille pour vérifier qu'il n'y a pas d'erreurs.

On appuie sur OK ou sur F9 et on obtient :



On tape :

```
Jeu1(43,7)
```

On obtient :

```
7,6,6,6,6,6,6
```

On tape :

```
Jeu1(57,7)
```

On obtient :

```
9,8,8,8,8,8,8
```

On tape :

```
Jeu1(23,7)
```

On obtient :

```
4,4,3,3,3,3,3
```

— Deuxième stratégie,

On peut imaginer que n représente des billes que l'on doit mettre dans des boites. Chaque boite doit contenir 9 billes. Si $n = 9q + r$ avec $0 \leq r < 9$, on pourra, remplir les q premières boites, la boite $q + 1$ contiendra r billes et les $p - q - 1$ dernières boites contiendront 0 billes.

On écrit la fonction Jeu2 qui renvoie la séquence L et on fait imprimer la somme et la taille de L pour vérifier qu'il n'y a pas d'erreurs :

```
Jeu2(n,p):={
 local q,r,L;
 q,r:=iquorem(n,9);
 si (n>9*p) alors retourne "erreur" fsi;
 L:=seq(9,q),r,seq(0,p-q-1);
 afficher(sum(L),size(L));
 retourne L;
};
```

On tape :

```
Jeu2(43,7)
```

On obtient :

```
9,9,9,9,7,0,0
```

On tape :

```
Jeu2(57,7)
```

On obtient :

```
9,9,9,9,9,9,3
```

On tape :

```
Jeu2(23,7)
```

On obtient :

`9,9,5,0,0,0,0`

— **Exercice**
Programmer la stratégie suivante :
On considère que l'on distribue n cartes que l'on donne à p joueurs (avec $n \leq 9p$ de la façon suivante :
on fait un premier tour en donnant les cartes 5 par 5, si il reste des cartes, on fait un deuxième tour en les donnant 4 par 4. Dans le cas où $n = 50$ et $p = 6$, au premier tour chaque joueur à 5 cartes et on a distribué 5*6=30 cartes. Au deuxième tour, on distribue les 20 cartes restantes et les 5 premiers joueurs ont 9 cartes (car 4*5=20) et le dernier en a 5.
Dans le cas où $n = 47$ et $p = 7$, au premier tour chaque joueur à 5 cartes et on a distribué 5*7=35 cartes. Au deuxième tour, on distribue les 12 cartes restantes et les 3 premiers joueurs ont 9 cartes (car 3*4=12) et les 4 derniers joueurs en ont 5.
Corrigé avec `Xcas`
On tape on distribue toutes les cartes en au plus 2 tours :

```
Jeu3(n,p):={
 local q,r,L;
 si (n>9*p) alors retourne "erreur" fsi;
 si n< 5*p alors
   q,r:=iquorem(n,5);
   L:=seq(5,q),r,seq(0,p-q-1);
 sinon
   n:=n-5*p;
   q,r:=iquorem(n,4);
   L:=seq(9,q),5+r,seq(5,p-q-1);
 fsi;
 afficher(sum(L),size(L));
 retourne L;
}
:;
```

2.4 Un algorithme plus compliqué

On cherche tout d'anord une stratégie lorsque le nombre p délèves est une puissance de 2 et $n \leq 9p$.
On peut tout d'abord simplifier lorsque p vaut 1 ou 2 puis lorsque p est une puissance de 2 par exemple $p = 2^a$. Lorsque $p = 1$, la réponse est n, et lorsque $p = 2$, il suffit de partager n en deux parties `iquo(n,2)+irem(n,2)` et `iquo(n,2)` pour avoir la réponse.
On peut alors se ramener à ce cas en partageant le groupe en 2 sous-groupes de $p2 = p/2$ personnes et n en `iquo(n,2)+irem(n,2)` et `iquo(n,2)` etc...
On peut donc écrire une fonction récursive dans le cas où p est une puissance de 2, et renvoyer une erreur dans le cas contraire :

```
Jeurec(n,p):={
local a,b,ap2;
```

```
si (n>9*p or n<0) alors return "erreur" fsi;
si (n==0) alors return  seq(0,p) fsi;
si (p==1) alors return n fsi;
a:=0;
ap2:=2;
tantque (ap2<=p) faire
ap2:=ap2*2;
a:=a+1;
ftantque
ap2:=ap2/2 ;
b:=p-ap2;
si (b==0) alors
n1:=iquo(n,2);
p1:=iquo(p,2);
return Jeurec(n1,p1),Jeurec(n-n1,p1);
sinon  return "erreur" fsi ;
}
:;
```

Dans le cas où p n'est pas une puissance de 2, on peut écrire $p = 2^a + b$ avec $b < 2^a$, on peut donc partager le groupe en 2 sous-groupes l'un de taille 2^a et l'autre de taille b. Mais comment partager n ? Si $n \leq 9 * 2^a$, on peut choisir de partager n en n, 0, et si $n > 9 * 2^a$, on peut choisir de partager n en $n - 9b$, $9b$.
On peut aussi choisir de partager n en $n-9b$, $9b$ si $n > 9b$ et sinon lorsque $n \leq 9b$, on partage n en 0, n.
Dans `Jeu3`, on choisit la première solution.
Dans `Jeu4`, on choisit la deuxième solution.
On tape :

```
Jeu3(n,p):={
local a,b,ap2,n1,p1;
if (n>9*p or n<0) return "erreur";
print(n,p);
si (n==0) alors return seq(0,p) fsi;
si (p==1) alors return n fsi;
a:=0;
ap2:=2;
tantque (ap2<=p) faire
ap2:=ap2*2;
a:=a+1;
ftantque
ap2:=ap2/2 ;
b:=p-ap2;
si (b==0) alors
n1:=iquo(n,2);
p1:=iquo(p,2);
return Jeu3(n1,p1),Jeu3(n-n1,p1);
fsi
si (n>9*ap2) alors
```

```
return Jeu3(n-9b,ap2),Jeu3(9b,b);
sinon return Jeu3(n,ap2),Jeu3(0,b);
fsi
}
:;
Jeu4(n,p):={
local a,b,ap2,n1,p1;
if (n>9*p or n<0) return "erreur";
print(n,p);
if (n==0) return seq(0,p);
if (p==1) return n;
a:=0;
ap2:=2;
while(ap2<=p) {
ap2:=ap2*2;
a:=a+1;
}
ap2:=ap2/2 ;
b:=p-ap2;
if (b==0){
n1:=iquo(n,2);
p1:=iquo(p,2);
return Jeu4(n1,p1),Jeu4(n-n1,p1);}
if (n>9b){
return Jeu4(n-9b,ap2),Jeu4(9b,b);}
else return Jeu4(0,ap2),Jeu4(n,b);
}
:;
```

On peut aussi partager le groupe en 2 sous-groupes l'un de taille `iquo(p,2)` et `iquo(p,2)+irem(p,2)`. Comment partager `n` ?
Si `p` est pair :
`p=2p1` alors
si `n=2n1` on a `n<=9p` entraine `n1<=9p1` et
si `n=2n1+1` on a `n<=9p` entraine `n1<=9p1-1/2<9p1` et `n1+1<=9p1+1/2`
mais comme `n1` est un entier on a `n1+1<=9p1` donc si `p` est pair on partage `n` en `n1` et `n-n1`.
Si `p` vaut 1 on renvoie `n`.
Si `p` est impair différent de 1 :
`p=2p1+1` avec `p1` non nul alors
si `n=2n1` on a `n=2n1<=9p=18p1+9` entraine `n1<=9p1+9/2` et comme `n1` est un entier on a `n1<=9p1+4`
donc `n1-4<=9p1` et `n1-4<=9p1+9/2+4<9(p1+1)`
si `n=2n1+1` on a `n=2n1+1<=9p=18p1+9` entraine `n1<=9p1+4` soit `n1-4<=9p1` et `n1+5<=9p1+9<9(p1+1)`.
Mais il faut avoir `n1-4>=0` : mais lorsque `n1<=4` on a `n1<9<9p1` puisque `1<p1`.
On a donc montrer que si `p` est impair :
`n1<=4` entraine `n1<9p1` ou encore

`n1>=9p1` entraine `n1>4`,
et on a aussi `n1-4<=9p1`
Donc si `p` est impair :
On peut choisir :

— Premier choix
si `n1-4>=0` on partage `n` en `n1-4` et `n-n1+4` et
si `n1-4< 0` on partage `n` en `n1` `n-n1` car `n1<n1+1<=4<9p1<9(p1+1)`
puisque `p1` est non nul
On tape :

```
Jeu5(n,p):={
local a,b,n1,p1;
if (n>9*p or n<0) return "erreur";
print(n,p);
if (n==0) return  seq(0,p);
if (p==1) return n;
n1:=iquo(n,2);
p1:=iquo(p,2);
if (odd(p) and n1-4>=0){
return Jeu5(n1-4,p1),Jeu5(n-n1+4,p-p1);}
return Jeu5(n1,p1),Jeu5(n-n1,p-p1);
}
:;
```

— Deuxiéme choix
On peut aussi choisir de partager `n` en `n1` et `n-n1` lorsque `n1<=9p1` (on a alors aussi `n1<n1+1=9(p1+1)`) et lorsque `n1>9p1>4` de partager `n` en `n1-4` et `n-n1+4`. On tape :

```
Jeu6(n,p):={
local a,b,n1,p1;
if (n>9*p or n<0) return "erreur";
print(n,p);
if (n==0) return  seq(0,p);
if (p==1) return n;
n1:=iquo(n,2);
p1:=iquo(p,2);
if (odd(p) and n1>9*p1){
return Jeu5(n1-4,p1),Jeu5(n-n1+4,p-p1);}
return Jeu5(n1,p1),Jeu5(n-n1,p-p1);
}
:;
```

En résumé :
On tape : `Jeu1(57,7)`
On obtient : `9,8,8,8,8,8,8`
On tape : `Jeu2(57,7)`
On obtient : `9,9,9,9,9,9,3`
On tape : `Jeu3(57,7)`
On obtient : `7,8,7,8,9,9,9`
On tape : `Jeu4(57,7)`
On obtient : `7,8,7,8,9,9,9`

On tape : `Jeu5(57,7)`
On obtient : `8,8,8,8,8,8,9`
On tape : `Jeu6(57,7)`
On obtient : `8,8,8,8,8,8,9`
On tape : `Jeu1(23,7)`
On obtient : `4,4,3,3,3,3,3`
On tape : `Jeu2(23,7)`
On obtient : `9,9,5,0,0,0,0`
On tape : `Jeu3(23,7)`
On obtient : `5,6,6,6,0,0,0`
On tape : `Jeu4(23,7)`
On obtient : `0,0,0,0,7,7,9`
On tape : `Jeu5(23,7)`
On obtient : `3,2,2,4,4,4,4`
On tape : `Jeu6(23,7)`
On obtient : `5,3,3,3,3,3,3` Mais il y a certainement encore d'autres stratégies ! ! ! !

Chapitre 3

Jeu sur la notion de boucle

3.1 Description

Le meneur de jeux choisit une séquence $(a_0, a_1, ...a_9)$ de 10 nombres et il marque chaque nombre sur un carton. Il montre ces nombres les uns après les autres à la classe.

Chaque élève doit trouver une stratégie pour trouver la somme $a_0+a_1+a_2...+a_8+a_9$ de cette séquence : il n'a pas le droit de noter ces nombres il a juste le droit de noter le résultat.
Chaque élève doit trouver une nouvelle stratégie pour dire si la somme $a_0 + a_1 + a_2... + a_8 + a_9$ est paire ou impaire.
Chaque élève doit trouver une stratégie pour trouver la somme alternée $a_0 - a_1 + a_2... + a_8 - a_9$ de cette séquence.

3.2 La stratégie

On doit faire la somme au fur et à mesure : on garde en tête a_0, puis $a_0 + a_1$....
Pour dire si la somme $a_0 + a_1 + a_2... + a_8 + a_9$ est paire ou impaire, il est inutile de calculer cette somme, il suffit de retenir à chaque étape la parité de la somme par exemple en representant un nombre pair par 0 et un nombre impair par 1....
Pour avoir la somme alternée c'est un peu plus difficile car il faut savoir si on doit ajouter ou retrancher...

3.3 Une correction possible avec `Xcas`

```
Jeusomme(L):={
 local S,j;
 S:=0;
 pour j de 0 jusque 9 faire
  S:=S+L[j];
 fpour;
 retourne S;
}:;
```

```
Jeupari(L):={
 local S,j;
 S:=0;
 pour j de 0 jusque 9 faire
  S:=irem(S+L[j],2) ;
 fpour;
 retourne S;
}:;
Jeusommealt(L):={
 local S,j,s;
 S:=0;
s:=1;
 pour j de 0 jusque 9 faire
  S:=S+s*L[j];
  s:=-s;
 fpour;
 retourne S;
}:;
```

On tape :
`Jeusomme(seq(2^n, (n=0..9)))`
On obtient :
`1023`
On tape :
`L:=seq(2^n, (n=0..9))`
On obtient :
`1,2,4,8,16,32,64,128,256,512`
On tape :
`Jeupari(seq(2^n, (n=0..9)))`
On obtient :
`1 % 2`
Donc la somme est impaire.
On tape :
`Jeusommealt(seq((2)^n, (n=0..9)))`
On obtient :
`-341`
On tape :
`Jeusomme(seq((-2)^n, (n=0..9)))`
On obtient :
`-341`
On tape :
`LL:=seq((-2)^n, (n=0..9))`
On obtient :
`1,-2,4,-8,16,-32,64,-128,256,-512` On tape :
`sum(LL)`
On obtient :
`-341`

Chapitre 4

Jeu sur la notion de condition

4.1 Description

Le meneur de jeux choisit une séquence de 10 nombres $(a_0, a_1, ...a_9)$ et il marque chaque nombre sur un carton. Il montre ces nombres les uns après les autres à la classe.
Chaque élève doit trouver une stratégie pour trouver le maximum de cette séquence.
Chaque élève doit trouver une stratégie pour trouver le minimum de cette séquence.
Chaque élève doit trouver une stratégie pour trouver le maximum et le minimum de cette séquence.

4.2 La stratégie

Pour trouver le maximum, on doit comparer le nouveau nombre avec le maximum des nombres précedents : on garde donc en tête un nombre M qui est le maximum provisoire et qui vaut au départ a_0, puis, qui sera maximum de a_0 et a_1 etc...
Pour trouver le minimum, on doit comparer le nouveau nombre avec le minimum des nombres précedents : on garde donc en tête un nombre m qui est le minimum provisoire et qui vaut au départ a_0, puis, qui sera minimum de a_0 et a_1 etc...
Pour trouver le maximum et le minimum , on doit garder en tête un nombre M et un nombre m comparer le nouveau nombre avec M et m et éventuellement mettre soit M, soit m, à jour.

4.3 Une correction possible avec `Xcas`

```
Jeumax(L):={
 local M,j,n;
 M:=L[0];
 pour j de 1 jusque 9 faire
  n:=L[j];
  si (M<n) alors M:=n fsi;
 fpour;
 retourne M;
```

```
}:;
```

On tape :
```
Jeumax(seq((-2)^n+3n, (n=0..9)))
```
On obtient :
```
280
```
On tape :
```
L3:=seq((-2)^n+3n, (n=0..9))
```
On obtient :
```
1,1,10,1,28,-17,82,-107,280,-485
```

```
Jeumaxmin(L):={
 local M,m,j,n;
 M:=L[0];
 m:=L[0];
 pour j de 1 jusque 9 faire
  n:=L[j];
  si (M<n) alors
    M:=n;
  sinon
    si (m>n) alors m:=n; fsi;
  fsi;
 fpour;
 retourne M,m;
}:;
```

On tape :
```
Jeumaxmin(seq((-2)^n+3n, (n=0..9)))
```
On obtient :
```
280,-485
```
On tape :
```
L3:=seq((-2)^n+3n, (n=0..9))
```
On obtient :
```
1,1,10,1,28,-17,82,-107,280,-485
```

Chapitre 5

Un jeu sur la notion de diviseurs

5.1 Description

Soit n un entier et D l'ensemble de tous ses diviseurs. Le jeu se fait avec 2 joueurs.
Le premier joueur choisit un élément p de D.
Cela a pour effet d'oter de D tous les diviseurs de p.
Le deuxième joueur choisit un élément p du nouveau D etc...
Le perdant est est le joueur qui choisit n.

5.2 Le programme

```
jeudivis(n):={
local D, p;
D:=idivis(n);
afficher(D);
tantque D!= [n]  et D!=[] faire
 repeat
   saisir("joueur1",p);
 until member(p,D);
 afficher(p);
 si p==n alors return "le joueur1 a perdu" fsi;
 D:=[op(D minus idivis(p))];
 afficher(D);
 si D==[n]  alors return "le joueur2 a perdu" fsi;
repeat
   saisir("joueur2",p);
 until member(p,D);
 afficher(p);
si p==n alors return "le joueur2 a perdu" fsi;
 D:=[op(D minus idivis(p))];
 afficher(D);
si D==[n]  alors return "le joueur1 a perdu" fsi;
ftantque;
}:;
```

5.3 La stratégie

??????????????????????????????

5.4 Le jour de la semaine

5.4.1 L'énoncé

Étant donné une date entre l'an 1583 et 9999, trouver à quel jour de la semaine elle correspond.

5.4.2 Le calendrier grégorien

Le calendrier grégorien date du 15 octobre 1582 (le lendemain du jeudi 4 octobre 1582 fut le vendredi 15 octobre 1582 car avant les années bissextiles étaient tous les 4 ans ce qui donne comme durée moyenne de l'année civile 365.25 jours alors que la révolution de la terre autour du soleil est plus courte (365.242 jours) d'où un décalage qui était de 10 jours en 1582.
Avec la nouvelle règle (la dernière année de chaque siécle est bissextile si son millésime est divisble par 400) l'écart n'est plus que de l'ordre de 1 jour tous les 3000 ans.

5.4.3 La correction

Quand une année a est-elle bissextile ?

À partir du 15 octobre 1582, une année a est bissextile si :
a est divisible par 4 et a n'est pas divisible par 100 ou si a est divisible par 400.
Par exemple :
1900 est divisible par 4, par 100 mais pas par 400 donc 1900 n'est pas bissextile.
2000 est divisible par 4, par 100 et par 400 donc 2000 est bissextile.
Le test sur a s'écrit donc :

```
bissextile(a):=(irem(a,4)==0 and irem(a,100)!=0) or irem(a,400)==0.
```

On tape :

```
bissextile(1900)
```

On obtient :

```
faux
```

On tape :

```
bissextile(2000)
```

On obtient :

```
vrai
```

Nombre d'année bissextile depuis l'année a

En supposant que le calendrier débute en l'an 0, le nombre d'année bissextile depuis l'année a est :

```
nbre_de_bissextile(a):=iquo(a,4)- iquo(a,100) + iquo(a,400)
```

On tape :

```
nbre_de_bissextile(1800)
```

On obtient :
`436`
On tape :
`nbre_de_bissextile(1900)`
On obtient :
`460`
On tape :
`nbre_de_bissextile(2000)`
On obtient :
`485`
En effet dans l'intervalle $]1800, 1900]$ il y a 24=(460-436) années bissextiles et dans l'intervalle $]1900, 2000]$ il y a 25=(485-460) années bissextiles.
Étant donné une année a non bissextile et un mois m ($m = 1..12$) on constitue la liste `SL` qui permettra de trouver le numèro d'ordre du jour j du mois m dans l'année a.
Si l'année a est non bissextile on tape :
`L:=[0,31,28,31,30,31,30,31,31,30,31,30]`
`SL:=cumSum(L)`
`SL` est alors la liste :
`[0,31,59,90,120,151,181,212,243,273,304,334]`
Si l'année a est bissextile on tape :
`L:=[0,31,29,31,30,31,30,31,31,30,31,30]`
`SL:=cumSum(L)`
`SL` est alors la liste :
`[0,31,60,91,121,152,182,213,244,274,305,335]`
cela veut dire que :
le 1 janvier est le 0+1=1ier jour de l'année a,
le 4 janvier est le 0+4=4ième jour de l'année a,
le 2 février est le 2+31=33ième jour de l'année a,
le 21 septembre d'une année non bissextile est le 21+243=264 ième jour de l'année a
mais le 21 septembre d'une année bissextile est le 21+244=265 ième jour de l'année a.
Donc le jour j du mois m d'une année a bissextile (ou non) est le `j+SL[m-1]` ième jour de l'année a (il faut mettre comme indice $m-1$ car les indices des listes commencent à 0).
On tape le programme qui renvoie le nombre 0 pour dimanche, 1 pour lundi,.., et 6 pour samedi.
La variable `b` est le numèro d'ordre du jour j du mois m dans l'année a compté depuis l'an 0 :
`(a-1)*365+iquo(a1,4)+iquo(a1,400)-iquo(a1,100)` est le nombre de jours écoulés entre le début de l'an 0 et la fin de l'année $a-1$ et `SL[m-1]+j` est le le numèro d'ordre du jour j du mois m dans l'année a.
Remarque
On devra ensuite :

— soit ajuster le programme en vérifiant la réponse pour une date comme le 15 octobre 1583 qui doit renvoyer 5 ou pour la date du jour (ici le 21 septembre

2014 qui doit renvoyer 0).

— soit remarquer que 1582 n'est pas bissextile et que 1582=7*226 c'est à dire que 1582 est l'année 0 modulo 7, que le 15 octobre d'une année non bissextile est le 288 ième jour de l'année. Comme le résultat est le reste de la division par 7, le 15 octobre d'une année a non bissextile est donc le même jour de la semaine que le 1ier janvier et le 31 decembre de cette année a car on a 1=0*7+1, 288=7*41+1 et 365=7*52+1.
En supposant que le calendrier grégorien débute en l'an 1 que l'an 1 n'est pas bissextile, le nombre d'années bissextiles entre le 1,1,1 et le 31,12 1581 est :
`iquo(1581,4)+iquo(1581,400)-iquo(1581,100)` qui renvoie $383 = 5 \bmod 7$.
Comme `irem(383,7)` vaut 5 cela veut dire que si le 1,1,1 est un lundi alors le 15,10,1582 sera un vendredi et donc on n'aura rien à ajuster.
On vérifie :
Si le 1,1,1 est le 1ier jour, le 1,1,2 est le le 366 ième jour et on a :
1+365*1+0=2 $\bmod 7$ Si le 1,1,1 est le 1ier jour, le 1,1,5 est le 1+365*4+1 ième jour et on a :
1+365*4+1=6 $\bmod 7$ Si le 1,1,0 est le 1ier jour, le 1,1,1582 est le :
1+365*1581+383=5 $\bmod 7$ Si le 1,1,0 est le 1ier jour, le 15,10,1582 est le 1+287+365*1581+383 ième jour et on a :
1+287+365*1581+383=5 $\bmod 7$ On peut dire que si le 1,1,1 est le 1ier jour i.e. un lundi alors le 15,10,1582 sera un vendredi.

On tape :

```
jour_semaine0(j,m,a):={
  local L,SL,a1,b;
  si j<=0 or m<=0 or a<=0 alors return "erreur : j>,m>0,a>0"; fsi;
  L:=[0,31,28,31,30,31,30,31,31,30,31,30];
  si (irem(a,4)==0 and irem(a,100)!=0) or irem(a,400)==0 then
    L[2]:=29;
  fsi;
  SL:=cumSum(L);
   a1:=a-1;
  b:=a1*365+iquo(a1,4)+iquo(a1,400)-iquo(a1,100)+SL[m-1]+j;
  return irem(b,7);
}:;
```

On remarque que si `(j,m,a)=(1,1,1)` alors `b=1` donc il n'y à rien à ajuster ;
On vérifie qu'il n'y à rien à ajuster, on tape :
`jour_semaine0(15,10,1582)` et on obtient bien `5`.
Ou en mettant une date de référence par exemple on sait que le 21 septembre 2014 est un dimanche.
On tape :
`jour_semaine0(21,9,2014)`
On obtient :
`0`
Comme le résultat est le reste de la division par 7 on peut taper puisque 365=7*52+1 on peut aussi écrire :

```
jour_semaine1(j,m,a):={
  local L,SL,a1,b;
  si j<=0 or m<=0 or a<=0 alors return "erreur : j>,m>0,a>0"; fsi;
  L:=irem([0,31,28,31,30,31,30,31,31,30,31,30],7);
  si (irem(a,4)==0 and irem(a,100)!=0) or irem(a,400)==0 then
    L[2]:=L[2]+1;
  fsi;
  SL:=irem(cumSum(L),7);
  a1:=a-1;
  b:=a1+iquo(a1,4)+iquo(a1,400)-iquo(a1,100)+SL[m-1]+j;
  return irem(b,7);
}:;
```

Remarque
On tape pour a non bissextile :
`SL7:=irem([0,31,59,90,120,151,181,212,243,273,304,334],7)`
On obtient :
`[0,3,3,6,1,4,6,2,5,0,3,5]`
On tape :
`LL:=[(iquo(23*m,9)+5)$(m=1..2),(iquo(23*m,9)+3)$(m=3..12)]`
On obtient :
`[7,10,10,13,15,18,20,23,26,28,31,33]`
On tape :
`LL7:=irem(LL,7)`
On obtient :
`[0,3,3,6,1,4,6,2,5,0,3,5]`
et pour a bissextile, on tape :
`SL7:=irem([0,31,60,91,121,152,182,213,244,274,305,335],7)`
On obtient :
`[0,3,4,0,2,5,0,3,6,1,4,6]`
On tape :
`LL:=[(iquo(23*m,9)+5)$(m=1..2),(iquo(23*m,9)+4)$(m=3..12)]`
On obtient :
`[7,10,11,14,16,19,21,24,27,29,32,34]`
On tape :
`LL7:=irem(LL,7)`
On obtient :
`[0,3,4,0,2,5,0,3,6,1,4,6]`
Donc `LL7` et `SL7` sont identiques : on peut donc modifier le programme.
La variable `bi` vaut `0` si l'année n'est pas bissextile et vaut `1` si l'année est bissextile.
On utilse alors les valeurs :
`bi+(iquo(23*m,9)+5)` pour `m=1` (janvier) et `m=2` (fevrier) et
`bi+(iquo(23*m,9)+3)` pour les autres mois.
On tape (plus besoin de listes !) :

```
jour_semaine2(j,m,a):={
  local L,SL,a1,b,bi;
  si j<=0 or m<=0 or a<=0 alors return "erreur : j>,m>0,a>0"; fsi;
```

```
  a1:=a-1;bi:=0;
  si (irem(a,4)==0 and irem(a,100)!=0) or irem(a,400)==0 alors
   bi:=1;
  fsi;
  si (m==1) ou (m==2) alors
   b:=a1+iquo(a1,4)+iquo(a1,400)-iquo(a1,100)+iquo(23*m,9)+5+j;
  sinon
   b:=bi+a1+iquo(a1,4)+iquo(a1,400)-iquo(a1,100)+iquo(23*m,9)+3+j;
  fsi;
return irem(b,7);
}:;
```

Ou bien si on utilise $a := a - 1$ pour janvier et février et a pour les autres mois, en mettant :
`a+iquo(a,4)+iquo(a,400)-iquo(a,100)` on ne doit pas tester si `a` est bissextile.
En effet `a+iquo(a,4)+iquo(a,400)-iquo(a,100)` est égale à :
`a1+1+bi+iquo(a1,4)+iquo(a1,400)-iquo(a1,100)` (avec les notations du programme précédent).
Donc pour janvier et février puisque `a:=a-1`, on a :
`b:=a+iquo(a,4)+iquo(a,400)-iquo(a,100)+iquo(23*m,9)+5+j;`
et
pour les autres mois
`b:=a+iquo(a,4)+iquo(a,400)-iquo(a,100)+iquo(23*m,9)+2+j;`
d'où la dernière commande :
`si (m==1) ou (m==2) alors return irem(b,7) sinon return irem(b-3,7);fsi;`
On tape et alors dans ce cas le programme marche pour l'année 0 qui est considérée comme bissextile :

```
jour_semaine3(j,m,a):={
  local L,SL,b;
   //si j<=0 or m<=0 or a<=0 alors return "erreur : j>,m>0,a>0"; fsi;
   si (m==1) ou (m==2) alors a:=a-1; fsi;
   b:=a+iquo(a,4)+iquo(a,400)-iquo(a,100)+iquo(23*m,9)+5+j;
   si (m==1) ou (m==2) alors return irem(b,7)
sinon return irem(b-3,7);fsi;
}:;
```

On peut rajouter :

```
Affichage_jour(n ):={
    if irem(n, 7) == 1 then
        return("Lundi");
    elif irem(n,7) == 2 then
        return("Mardi");
    elif irem(n, 7) == 3 then
        return("Mercredi");
    elif irem(n, 7) == 4 then
```

```
        return("Jeudi");
    elif irem(n, 7) == 5 then
        return("Vendredi");
    elif irem(n, 7) == 6 then
        return("Samedi");
    elif irem(n, 7) == 0 then
        return("Dimanche");
        end;
    }:;
jour_semaine4(j,m,a):={
  local L,SL,b;
   si (m==1) ou (m==2) alors a:=a-1; fsi;
   b:=a+iquo(a,4)+iquo(a,400)-iquo(a,100)+iquo(23*m,9)+5+j;
   si (m==1) ou (m==2) alors return Affichage_jour(irem(b,7));
sinon return Affichage_jour(irem(b-3,7));fsi;
}:;
```

5.4.4 Le programme commenté de l'étudiant Thomas Luka

Ce programme est valable pour les dates postérieures au 15 octobre 1582.

```
NomJour(jour, mois, an):={
    local nbJourTotal,nbMois,nbAnBis;
    if an<1582 then
      return("erreur la date doit etre apres le 15/10/1582")
   end_if;
    nbJourTotal := jour;//nb Jour du mois;
    nbMois := 1; //Traitement du nb jour dans l'annee
    while (nbMois <= 7) and (nbMois < mois) do
      if (irem(nbMois,2) == 1) then //mois est impair donc 31 jours
        nbJourTotal := 31+nbJourTotal;
      else
        nbJourTotal := 30+nbJourTotal;
      end_if;
      if (nbMois == 2) then
        if irem(an,4)== 0 and irem(an,100)!= 0 or irem(an,400) == 0 then
          nbJourTotal := nbJourTotal-1; //29 jours en fevrier
        else
          nbJourTotal := nbJourTotal-2; //28 jour en fevrier
        end_if;
      end_if;
      nbMois:=nbMois +1;
    end;
    while (nbMois > 7) and (nbMois < mois)  do
        if (irem(nbMois, 2) == 0) then
        //mois est pair donc 31 jours a partir d'aout
            nbJourTotal := nbJourTotal+31;
        elif (irem(nbMois, 2) == 1) then
            nbJourTotal := nbJourTotal+ 30;
```

```
      end;
      nbMois := nbMois +1;
     end;
    afficher(nbJourTotal);
    if (an<1582 and nbJourTotal<288 then
      return "erreur la date doit etre apres le 15/10/1582";
    end_if;
   //Ajout des jours de 1582 de ](15,10,1582),(31,12,1582)]
    nbJourTotal := nbJourTotal +16 + 30 + 31;
   //16 + 30 + 31 est l'ajout des jours de ](15,10,1582),(31,12,158
   //i.e le vendredi est alors le 0 ou bien
   //nbJourTotal += 17 + 30 + 31 si on ajoute les jours de
   //[(15,10,1582),(31,12,1582)] et le vendredi est alors le 1
   //Traitement du NbJour pour les annees entre [1583,an-1]
  //L'année en cours n'est pas fini et sera ajoutée avec les jours
   //Nbre d'annees bissextiles entre [1583,an-1]
   an := an-1;
   nbAnBis:= iquo(an, 4);
   nbAnBis := nbAnBis-iquo(an,100);
   nbAnBis := nbAnBis+iquo(an,400);
   nbAnBis :=nbAnBis- iquo(1582,4);
   nbAnBis :=nbAnBis+ iquo(1582,100);
   nbAnBis :=nbAnBis- iquo(1582,400);
   an := an-1582;
  //le nb de jour entre [1583,an-1] est an * 365 + nbAnBis
  /cela est valable pour 1583 qui est devenu l'an 0 (nbJourTotal+0*
  //pour 1582 qui est devenu l'an -1 (288+16+30+31-365 vaut 0 pour
  //1582 et l'annee en cours sont ajoutés donc
   nbJourTotal := nbJourTotal +an * 365 + nbAnBis;
  //Affichage du jour ( enfin !)
  if irem(nbJourTotal, 7) == 0 then
      return("Vendredi");
  elif irem(nbJourTotal,7) == 1 then
      return("Samedi");
  elif irem(nbJourTotal, 7) == 2 then
      return("Dimanche");
  elif irem(nbJourTotal , 7) == 3 then
      return("Lundi");
  elif irem(nbJourTotal, 7) == 4 then
      return("Mardi");
  elif irem(nbJourTotal, 7) == 5 then
      return("Mercredi");
  elif irem(nbJourTotal , 7) == 6 then
      return("Jeudi");
    end;
  }:;
```

On tape :

```
NomJour(1,1,1582),NomJour(15,10,1582),NomJour(31,12,1582)
```

On obtient :
```
"la date doit etre apres le 15/10/1582","Vendredi","Vendredi"
```
On tape :
```
NomJour(1,1,1584),NomJour(15,10,1584),NomJour(31,12,1584)
```
On obtient car 1584 est bissextile :
```
"Dimanche","Lundi","Lundi"
```
On tape :
```
NomJour(1,1,2000),NomJour1(7,10,2014),NomJour1(31,12,2016)
```
On obtient :
```
"Samedi","Mardi","Samedi"
```

Chapitre 6

Vue d'ensemble de Xcas pour le programmeur

En vous référant au menu Aide->Manuels->Algorithmes vous aurez plus de détails sur l'algorithmique et Xcas, toutefois cette introduction est suffisante pour commencer.
Il y a aussi beaucoup d'exemples traités dans les autres manuels....

6.1 Installation de Xcas

Le programme Xcas est un logiciel libre écrit en C++, (disponible sous licence GPL). La version à jour se récupère sur :
http : //www − fourier.ujf − grenoble.fr/~parisse/giac_fr.html ou
ftp://fourier.ujf-grenoble.fr/xcas
où l'on trouve le code source (giac.tgz) ou des versions précompilées pour Linux (PC ou ARM), Windows, Mac OS.

6.2 Les différents modes

Xcas propose un mode de compatibilité avec Maple, MuPAD et la TI89/92 : pour cela, il suffit de le spécifier dans Prog style du menu de configuration du cas (bouton Config ou menu Cfg->Configuration du CAS) ou avec le menu Cfg->Mode (syntax). On peut choisir, en cliquant sur la flèche située à coté de Prog style : Xcas ou Maple ou MuPAD ou TI89/92.
On a aussi la possibilité d'importer une session Maple ou une archive TI89/92 en choisissant Importer du menu Fich, ou importer dans un niveau éditeur de programmes un fichier écrit en syntaxe Maple, Mupad ou TI89/92 par le menu Prog->Inserer.

On présente ici le mode Xcas qui est proche de la syntaxe C. On a aussi la possibilité d'avoir toutes les instructions en français de façon à être proche du langage Algorithmique.

6.3 Éditer, sauver, exécuter un programme avec la syntaxe Xcas

On écrit directement un script (i.e. une suite de commandes séparées par ; ou par :; ou plusieurs programmes ou fonctions séparées par ; ou par :; dans un niveau éditeur de programmes (que l'on ouvre avec Alt+p).
Depuis un niveau éditeur de programmes, on peut tester facilement si le programme est syntaxiquement correct grâce au bouton OK ou sur F9 : la ligne où se trouve la faute de syntaxe est indiquée en bleu dans la zone intermédiaire.
On corrige les fautes lorsqu'il y en a...
Quand le programme est syntaxiquement correct, il y a dans la zone intermédiaire Success compilling ... et on a le programme en réponse ou simplement Done lorsqu'on a terminé l'écriture du programme par :;. On peut alors exécuter le programme dans une ligne de commande.
Vous sauvez le programme avec le bouton Save du niveau éditeur de programmes sous le nom que vous voulez lui donner en le terminant par le suffixe .cxx (ce nom s'inscrit alors à côté du bouton Save du niveau éditeur de programmes. Si ensuite, vous voulez lui donner un autre nom il faut le faire avec le menu Prog sous-menu Sauver comme de l'éditeur de programmes.

6.4 Débugger un programme avec la syntaxe Xcas

Pour utiliser le débuggeur, il faut que ce programme soit syntaxiquement correct : vous avez par exemple un programme syntaxiquement correct, mais qui ne fait pas ce qu'il devrait faire, il faut donc le corriger.

Avec le débuggeur, on a la possibilité d'exécuter le programme au pas à pas (sst), ou d'aller directement (cont) à une ligne précise marquée par un point d'arrêt (break), de voir (voir ou watch) les variables que l'on désire surveiller, d'exécuter au pas à pas les instructions d'une fonction utilisateur utilisée dans le programme (dans ou sst_in), ou de sortir brutalement du débuggeur (tuer ou kill).
On tape : debug(nom _du_programme(valeur_des_ arguments)).
Il faut bien sûr que le programme soit validé :

- si le programme est dans un niveau éditeur de programme, on appuie sur OK ou sur F9 pour le compiler, on corrige les fautes de syntaxe éventuelles et on appuie sur OK ou sur F9 jusqu'à obtenir Success compiling...
- si le programme qui est syntaxiquement correct se trouve dans un fichier, on tape : read("toto") si toto est le nom du fichier où se trouve ce programme.

Par exemple, si pgcd a été validé, on tape :
debug(pgcd(15,25))
L'écran du débugger s'ouvre : il est formé par trois écrans séparés par une ligne eval et une barre de boutons sst,dans,cont... :

1. dans l'écran du haut, le programme source est écrit et la ligne en surbrillance sera exécutée grâce au bouton sst.

2. dans la ligne `eval`, `Xcas` marque automatiquement l'action en cours par exemple `sst`. Cette ligne permet aussi de faire des calculs dans l'environnement du programme ou de modifier une variable, par exemple on peut y écrire `a:=25` pour modifier la valeur de `a` en cours de programme,

3. dans l'écran du milieu, on trouve, les points d'arrêts, le numéro de la ligne du curseur.

4. une barre de boutons `sst,dans,cont...`
   - `sst` exécute la ligne courante (celle qui est en surbrillance) sans entrer dans les fonctions et met en surbrillance l'instruction suivante,
   - `dans` ou `sst_in` exécute la ligne courante (celle qui est en surbrillance) en entrant dans les fonctions utilisées dans le programme et qui ont été définies précédemment par l'utilisateur, puis met en surbrillance l'instruction suivante du programme en incluant les instructions de la fonction. Cela permet ainsi d'exécuter pas à pas les instructions de cette fonction.
   - `cont` exécute les instructions du programme situées entre la ligne courante et la ligne d'un point d'arrêt et met en surbrillance cette ligne,
   - `tuer` ou `kill` ferme brutalement l'écran du débuggeur.
     Attention il faut fermer l'écran du débuggeur pour pouvoir utiliser `Xcas`.
   - `break` ajoute un point d'arrêt. Les points d'arrêts permettent d'aller directement à un point précis avec le bouton `cont`. On marque les points d'arrêts grâce au bouton `break` ou à la commande `breakpoint` d'arguments le nom du programme et le numéro de la ligne où l'on veut un point d'arrêt : par exemple `breakpoint(pgcd,3)`. Pour faciliter son utilisation, il suffit de cliquer dans l'écran du haut sur la ligne où l'on veut le point d'arrêt pour avoir : `breakpoint` dans la ligne `eval`, avec le nom du programme et le bon numéro de ligne, puis de valider la commande. Il suffit donc de cliquer et de valider !
   - `rmbrk` enlève un point d'arrêt. On doit, pour réutiliser d'autres points d'arrêts, d'effacer les points d'arrêts utilisés précédemment avec la commande `rmbreakpoint` qui a les mêmes arguments que `breakpoint`. Là encore, pour faciliter son utilisation, il suffit de cliquer sur la ligne où l'on veut enlever le point d'arrêt pour avoir : `rmbreakpoint` dans la ligne de commande, avec le nom du programme et le bon numéro de ligne. **Attention** si il n'y a pas de point d'arrêt à cet endroit `Xcas` en mettra un !
   - `voir` ou `watch` ajoute la variable que l'on veut voir évoluer. Si on ne se sert pas de `voir` ou `watch` toutes les variables locales et tous les arguments du programme sont montrées. Si on se sert de `voir` ou `watch` seules les variables designées seront montrées : on appuie sur le bouton `voir` ou `watch` et la commande `watch` s'écrit dans la ligne d'évaluation `eval`. On tape alors, les arguments de `watch` qui sont les noms des variables que l'on veut surveiller, par exemple : `watch(b,r)` et on valide la commande.
   - `rmwtch` efface les variables désignées précédemment avec `watch` et que l'on ne veut plus voir, par exemple : `rmwatch(r)`.

5. dans l'écran du bas, on voit soit l'évolution de toutes les variables locales et de tous les arguments du programme, soit l'évolution des variables désignées par `watch`.

Chapitre 7

Les instructions avec la syntaxe Xcas

7.1 Les commentaires

Les commentaires sont des chaînes de caractères, ils sont précédés de `//` ou sont parenthésés par `/* */`

7.2 Le bloc

Une `action` ou `bloc` est une séquence d'une ou plusieurs instructions.
Quand il y a plusieurs instructions il faut les parenthéser avec `{ }` et séparer les instructions par un point virgule (`;`)
Un `bloc` est donc parenthésé par `{ }` et commence éventuellement par la déclaration des variables locales (`local...`).

7.3 Les variables globales et les variables locales

Voir aussi **??** Les variables sont les endroits où l'on peut stocker des valeurs, des nombres, des expressions, des objets.
Le nom des variables est formé par une suite de caractères et commence par une lettre : attention on n'a pas droit aux mots réservés ...ne pas utiliser par exemple la variable `i` dans un `for` ou un `pour` car `i` représente le nombre complexe de module 1 et d'argument $\frac{\pi}{2}$.
L'affectation se fait avec `:=` (par exemple `a:=2; b:=a;`).

7.4 Les programmes et les fonctions

- Les paramètres sont mis après le nom du programme ou de la fonction entre parenthèses : par exemple `f(a,b):=...`.
  Ces paramètres sont initialisés lors de l'appel du programme ou de la fonction et se comportent comme des variables locales.
- L'affectation se fait avec `:=` (par exemple `a:=2; b:=a;`),
- Les entrées se font par passage de paramètres ou avec `input` ou `saisir`,

— Les sorties se font avec `print` ou `afficher`,
— Il n'y a pas de distinction entre programme et fonction : la valeur d'une fonction est précedée du mot réservé `retourne` ou `retourne`.
Remarque `retourne` n'est pas obligatoire car `Xcas` renvoie toujours la valeur de la dernière instruction, mais `retourne` est très utile car il fait sortir de la fonction : les instructions situées après `retourne` ou `retourne` ne sont jamais effectuées.

7.5 Les tests

Avec le langage `Xcas`, les tests ont, soit une syntaxe similaire au langage `C++`, soit une version française proche du langage algorithmique.
Pour les tests, les syntaxes admises sont :

— `if` *`(condition) instruction`*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`if` *`(condition)`* {*`instructions`*}
ou
`si` *`condition`* `alors` *`instructions`* `fsi`
on teste la condition : si elle est vraie, on fait les instructions et si elle est fausse on ne fait rien c'est à dire on passe aux instructions qui suivent le `if` ou le `si`.
Par exemple :

```
testif1(a,b):={
if  (a<b)
    b:=b-a;
retourne [a,b];
};
```

ou

```
testsi1(a,b):={
si a<b alors
    b:=b-a;
fsi;
retourne [a,b];
};
```

et on a :

```
testif1(3,13)=testsi1(3,13)=[3,10]
testif1(13,3)=testsi1(13,3)=[13,3]
```

— `if` *`(condition) instruction1`*`; else` *`instruction2`*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`if` *`(condition)`* {*`instructions1`*} `else` {*`instructions2`*}
ou
`si` *`condition`* `alors` *`instructions1`* `sinon` *`instructions2`*
`fsi`
on teste la condition : si elle est vraie, on fait les instructions1 et si elle est fausse on fait les instructions2.
Par exemple :

```
testif(a,b):={
```

```
if (a==10 or a<b)
    b:=b-a;
else
    a:=a-b;
retourne [a,b];
};
```

ou

```
testsi(a,b):={
si a==10 or a<b alors
    b:=b-a;
sinon
    a:=a-b;
fsi;
retourne [a,b];
};
```

et on a :

```
testif(3,13)=testsi(3,13)=[3,10]
testif(13,3)=testsi(13,3)=[10,3]
testif(10,3)=testsi(10,3)=[10,-7]
```

7.6 Les boucles

Avec le langage Xcas, les boucles ont soit une syntaxe similaire au langage C++ soit une version française proche du langage algorithmique.
Pour les boucles, les syntaxes admises sont :

— la boucle `for` ou `pour` permet de faire des instructions un nombre de fois qui est connu :
`for (init;condition;increment) instruction;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`for (init;condition;increment) {instructions}`
Le plus souvent (*init ;condition ;increment*) s'écrit en utilisant une variable (par exemple `j` ou `k`...mais pas `i` qui désigne un nombre complexe ! ! !). Cette variable sera initialisée dans *init*, utilisée dans *condition* et incrementée dans *increment*, on écrit par exemple :
(*j :=1 ;j<=10 ;j++*) (l'increment ou le pas est de 1) ou
(*j :=10 ;j>=1 ;j–*) (l'increment ou le pas est de -1) ou
(*j :=1 ;j<=10 ;j :=j+2*) (l'increment ou le pas est de 2)
ou

```
pour j de 1 jusque 10 faire instructions fpour;
```

ou

```
pour j de 10 jusque 1 pas -1 faire instructions fpour;
```

ou

```
pour j de 1 jusque 10 pas 2 faire instructions fpour;
```

On initialise `j` puis on teste la condition :
— si elle est vraie, on fait les instructions puis on incrémente `j`, puis, on teste la condition : si elle est vraie, on fait les instructions etc...
— si elle est fausse on ne fait rien c'est à dire on passe aux instructions qui

suivent le `for` ou le `pour`.

Par exemple :

```
testfor1(a,b):={
local j,s:=0;
for (j:=a;j<=b;j++)
   s:=s+1/j^2;
retourne s;
};
```

ou

```
testpour1(a,b):={
local j,s:=0;
pour j de a jusque b faire
   s:=s+1/j^2;
fpour;
retourne s;
};
```

Si `a>b`, l'instruction ou le bloc d'instructions du `for` ou du `pour` ne se fera pas et la fonction retournera `0`,
Si `a<=b` la variable `j` va prendre successivement les valeurs `a, a+1,...b` (on dit que le pas est de 1) et pour chacune de ces valeurs l'instruction ou le bloc d'instructions qui suit sera exécuté. Par exemple `testfor1(1,2)` renverra `1+1/4=5/4`.

```
testfor2(a,b):={
local j,s:=0;
for (j:=b;j>=a;j--)
    s:=s+1/j^2;
retourne s;
};
```

ou

```
testpour2(a,b):={
local j,s:=0;
pour j de b jusque a pas -1 faire
   s:=s+1/j^2;
fpour;
retourne s;
};
```

Dans ce cas, si `a<=b` la variable `j` va prendre successivement les valeurs `b, b-1,...a` (on dit que le pas est de -1) et pour chacune de ces valeurs l'instruction ou le bloc d'instructions qui suit sera exécuté. Par exemple `testfor2(1,2)` renverra `1/4+1=5/4`.

```
testfor3(a,b):={
local j,s:=0;
for (j:=a;j<=b;j:=j+3)
    s:=s+1/j^2;
retourne s;
};
```

ou

```
testpour3(a,b):={
local j,s:=0;
pour j de a jusque b pas 3 faire
   s:=s+1/j^2;
fpour;
retourne s;
};
```

Dans ce cas, si `a<=b` la variable `j` va prendre successivement les valeurs `a, a+3,...a+3k` avec `k` le quotient de `b-a` par 3 (`3k<=b-a<3(k+1)`) (on dit que le pas est de 3) et pour chacune de ces valeurs l'instruction ou le bloc d'instructions qui suit sera exécuté. Par exemple `testfor3(1,5)` renverra `1+1/16=17/16`.

— la boucle `while` ou `tantque` permet de faire plusieurs fois des instructions avec une condition d'arrêt au début de la boucle : `while (`*`condition`*`)` *`instruction`*`;`
on met {..} lorsqu'il faut faire plusieurs instructions :
`while (`*`condition`*`) {`*`instructions`*`}`
ou
`tantque` *`condition`* `faire` *`instructions`* `ftantque;`
On teste la condition :
  — si elle est vraie, on fait les instructions puis, on teste la condition : si elle est vraie, on fait les instructions etc...
  — si elle est fausse on ne fait rien c'est à dire on passe aux instructions qui suivent le `while` ou le `tantque`.

Par exemple :

```
testwhile(a,b):={
while (a==10 or a<b)
   b:=b-a;
retourne [a,b];
};
```

ou

```
testtantque(a,b):={
tantque a==10 or a<b faire
   b:=b-a;
ftantque
retourne [a,b];
};
```

Un exemple : le PGCD d'entiers

```
pgcd(a,b):={
 local r;
 while (b!=0) {
    r:=irem(a,b);
    a:=b;
    b:=r;
 }
 retourne a;
```

```
};
```
ou
```
pgcd(a,b):={
 local r;
 tantque b!=0 faire
    r:=irem(a,b);
    a:=b;
    b:=r;
 ftantque;
 retourne a;
};
```

— La boucle `repeat` ou `repeter` permet de faire plusieurs fois des instructions avec une condition d'arrêt à la fin de la boucle :
`repeat` *`instructions`* `until` *`condition`*`;`
ou
`repeter` *`instructions`* `jusqua` *`condition`*`;`
ou
`repeter` *`instructions`* `jusqu_a` *`condition`*`;`
On fait les instructions, puis on teste la condition :
— si elle est vraie, on fait les instructions puis, on teste la condition etc...
— si elle est fausse on passe aux instructions qui suivent le `repeat` ou le `repeter`.

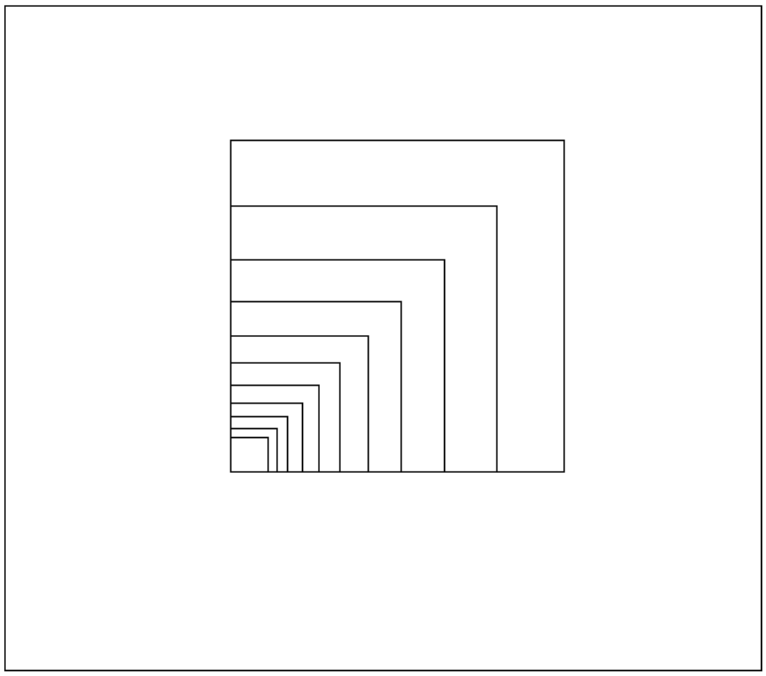
Par exemple :
```
f():={
  local a;
  repeat
    saisir("entrez un reel entre 1 et 10",a);
  until a>=1 et a<=10;
  retourne a;
}
```
ou
```
f():={
  local a;
  repeter
    saisir("entrez un reel entre 1 et 10",a);
  jusqua a>=1 et a<=10;
  retourne a;
}
```

Chapitre 8

Des exemples de programmes

8.1 Des carrés emboités

Soient A le point (0 ;0), B le point (0 ;5) et K le carré direct de côté AB.
À partir de K on définit K_1 le carré direct de côtés AB_1 avec $AB_1 = 0.8 * AB$, puis K_2 le carré direct de côtés AB_2 avec $AB_2 = 0.8 * AB_1$ etc...
Faire le dessin de $K, K_1, ...K_{10}$ avec `Xcas`.



On va montrer ici qu'un même dessin peut être obtenu avec des programmes de complexité très différentes : au pas à pas dans un niveau de géométrie, en écrivant un script dans un niveau d'éditeur de programmes, en mettant des paramètres, en écrivant une fonction itérative ou récursive.

1. On ouvre un niveau de géométrie (`Alt+g`.

On tape :

```
carre(0,5);
carre(0,5*0.8);
carre(0,5*0.8^2);
carre(0,5*0.8^3);
carre(0,5*0.8^4);
carre(0,5*0.8^5);
carre(0,5*0.8^6);
carre(0,5*0.8^7);
carre(0,5*0.8^8);
carre(0,5*0.8^9);
carre(0,5*0.8^10);
```

1ière Amélioration introduction d'une variable
On veut pouvoir changer le coefficient 0.8, on met alors 0.8 dans une variable `a` et on rajoute une ligne au début du niveau de géométrie (on clique sur son niveau 1 et on tape sur `Alt+n`). On tape alors dans cette nouvelle ligne :
`a:=0.8` et on remplace tous les 0.8 par `a`.
En changeant la première ligne en `a:=0.7` et en la validant , on obtient les nouveaux carrés car dans un niveau de géométrie la modification d'une ligne entraine l'exécution des lignes suivantes.

2ième Amélioration introduction d'un curseur
Dans un niveau de géométrie, on peut changer le paramètre `a` d'un seul click en ajoutant un curseur qui fera varier le paramètre. Pour cela on supprime le niveau `a:=0.8` rajouté précédemment (on clique sur son numéro et tape sur la touche `Suppr`. On clique sur la première ligne et avec le menu `Edit` du niveau de géométrie, on sélectionne `Ajouter parametre` :
il faut changer les valeurs proposées, on garde le nom a et on met comme valeur `t-: 0, t+: 1, t :0.8, step: 0.1` et cocher `symb`. On obtient un niveau supplementaire et dans ce niveau :
`supposons(a=[0.8,0,1,0.1])`
on valide cette commande et il apparait un curseur sous le pavé de boutons situé à droite de l'écran graphique. En bougeant ce curseur, on obtient les carrés correspondant à la valeur de `a` du curseur.

3ième Amélioration introduction d'une boucle
Plutôt que taper 11 instructions `carre` on veut utiliser une boucle et taper :
`pour j de 0 jusque 10 faire carre(0,5*0.8^j); fpour;`
Mais cela ne dessine que le dernier carré car le `pour` renvoie seulement la dernière instruction. Il faut donc créer une séquence qui condiendra toutes les instructions `carre`.
On tape :

```
supposons(a=[0.8,-5.0,5.0,0.0]);
K:=carre(0,5):;
pour j de 1 jusque 10 faire K:=K,carre(0,5*a^j);fpour:;
K;
```

2. On tape le script dans un niveau d'éditeur de programmes (`Alt+p`) :

```
a:=0.8;
carre(0,5);
carre(0,5*a);
carre(0,5*a^2);
carre(0,5*a^3);
carre(0,5*a^4);
carre(0,5*a^5);
carre(0,5*a^6);
carre(0,5*a^7);
carre(0,5*a^8);
carre(0,5*a^9);
carre(0,5*a^10);
```

puis on appuie sur `OK(F9)` et on obtient le dessin désiré, mais on ne peut pas avoir de curseur.....
On tape :

```
a:=0.8;
K:=carre(0,5):;
pour j de 1 jusque 10 faire K:=K,carre(0,5*a^j);fpour:;
K;
```

ou en commençant par la séquence vide :

```
K:=NULL;
pour j de 0 jusque 10 faire K:=K,carre(0,5*0.8^j);fpour:;
K;
```

puis on appuie sur `OK(F9)` et on obtient le dessin désiré sans curseur....

3. On tape une fonction dans un niveau d'éditeur de programmes (`Alt+p`) :

```
carres(a):={
local j,K;
K:=carre(0,5);
pour j de 1 jusque 10 faire
K:=K,carre(0,5*a^j);
fpour;
retourne K;
}
:;
```

Remarque On peut aussi calculer a^j en introduisant une variable locale `puis` qui vaudra a^j à chaque étape :

```
carres(a):={
local j,puis,K;
K:=carre(0,5);
puis:=1;
pour j de 1 jusque 10 faire
puis:=a*puis;
K:=K,carre(0,5*puis);
fpour;
retourne K;
```

```
}
:;
```

puis on appuie sur `OK(F9)` et on obtient en bleu dans un niveau intermédiaire :
`// Parsing carres`
`// Success compiling carres`
et comme réponse `Done`.
Mais la fonction ne s'exécute pas, elle est juste "prête à être exécutée" .
On tape alors dans un niveau de géométrie :

```
supposons(a=[0.8,0,1,0.1]);
carres(a);
```

0n obtient le dessin désiré, avec un curseur pour faire varier le paramètre `a`.

Remarque
On peut rajouter des paramètres à la fonction `carres` comme par exemple le nombre n de carrés dessinés et la longueur du côte c du grand carré, on a :

```
carren(a,n,c):={
local j,K;
K:=carre(0,c);
pour j de 1 jusque n-1 faire
K:=K,carre(0,c*a^j);
fpour;
retourne K;
}
:;
```

4. On tape une fonction récursive dans un niveau d'éditeur de programmes (`Alt+p`) et on ouvre l'écran `DispG` (commande `DispG()`).
On tape dans l'éditeur de programmes :

```
carrer(a,n,c):={
si n==0 alors retourne 1 fsi;
carre(0,c);
carrer(a,n-1,c*a);
};
```

La fonction retourne 1 lorsque la récursivité s'est bien terminée.
Pour comprendre, il faut voir que `carrer(a,n,c)` est le dessin de n carrés où c est le côté de son grand carré. Il est composé d'un grand carré de côté c (`carre(0,c)`) et du dessin de $n-1$ carrés où $c*a$ est le côté de son grand carré c'est donc `carrer(a,n-1,c*a)`. On appuie sur `OK(F9)`, on obtient `// Success compiling carrer`
`carrer: recursive definition`
Le résultat d'un dessin récursif se fait dans l'écran `DispG`.
On tape dans une ligne d'entrée :
`DispG();ClrGraph(); carrer(0.8,10,5)`
l'écran `DispG` s'ouvre, s'efface et on obtient le dessin des 10 carrés dans l'écran `DispG`.

Remarque
On peut aussi voir `carrer(a,n,c)` comme le dessin formé par :
$n-1$ carrés où $c*a$ est le côté de son grand carré (`carrer(a,n-1,c*a)`) et d'un grand carré de côté c (`carre(0,c)`) On peut donc aussi taper dans l'éditeur de programmes et compiler :

```
carrer(a,n,c):={
si n==0 alors retourne 1 fsi;
carrer(a,n-1,c*a);
carre(0,c);
};
```

Puis on tape dans une ligne d'entrée :
`DispG();ClrGraph(); carrer(0.8,10,5)`

Si on veut se passer de l'écran `DispG` il faut gérer les listes et retourner une liste contenant toutes les instructions graphiques. On tape alors dans l'éditeur de programmes :

```
carrerL(a,n,c,L):={
si n==0 alors retourne L fsi;
L:=append(L,carre(0,c));
carrerL(a,n-1,c*a,L);
};
```

Puis on tape dans une ligne d'entrée :
`carrerL(0.8,10,5,[])`

Pour aller plus loin...
Essayez :

```
carresrL(a,b,k,n,L):={
local c;
si n==0 alors retourne L fsi;
L:=append(L,carre(a,b));
c:=(a+b)/2;
k:=(b-a)*k;
carresrL(c,b,k,n-1,carresrL(a,c,k,n-1,L));;
}:;
```

avec `carresrL(0,12,8,[])` et aussi

```
carresrL2(a,b,n,L):={
local c;
si n==0 alors retourne L fsi;
L:=append(L,carre(a,b));
c:=a+(b-a)/2;
carresrL2(c+i*(b-a)/2,b+i*(b-a)/2,n-1,carresrL2(a,c,n-1,L));
}:;
```

avec `carresrL2(0,12,8,[])` et aussi

```
carresrL3(a,b,n,L):={
local c;
si n==0 alors retourne L fsi;
L:=append(L,carre(a,b));
c:=a+(b-a)/2;
carresrL3(c+i*(b-a)/2,b+i*(b-a)/2,n-1,
     carresrL3(a,c,n-1,carresrL3(a+i*(b-a)/2,c+i*(b-a)/2,n-1,L)));
}:;
```

avec `carresrL3(0,12,8,[])`

8.2 Maximum de 3 nombres

On donne 3 nombres a, b, c trouver le maximum de ces 3 nombres.
On va décrire plusieurs méthodes :

— On peut utiliser la formule : `max(a,b,c)=max(max(a,b),c)`
et traduire cette formule par un algorithme.
On tape :

```
Max2(a,b):={
si a>b alors retourne a;
sinon retourne b; fsi
}:;
Max3(a,b,c):=Max2(Max2(a,b),c)
```

— On gére tous les cas avec des `si ...alors ...sinon...fsi`.
On tape :

```
Maxi(a,b,c):={
local M;
si a>b alors
  si a>c alors
    M:=a;
  sinon  M:=c;fsi
sinon
  si c>b alors
    M:=c;
  sinon  M:=b;fsi
fsi
retourne M;
}:;
```

— On utilise `retourne` qui fait sortir de la boucle.
Ou bien on teste si a est plus grand que b et est plus grand que c et dans ce cas on renvoie a. Puis on teste la même chose avec b puis avec c.
On tape :

```
Max(a,b,c):={
si a>=b et a>=c alors retourne a; fsi
si b>=a et b>=c alors retourne b; fsi
si c>=b et c>=a alors retourne c; fsi
}:;
```

— On peut aussi écrire une procédure récursive pour traduire que on teste si a est plus grand que b et est plus grand que c et dans ce cas on renvoie a et sinon on fait la même chose avec b puis avec c.
On tape :
```
Maxr(a,b,c):={
si a>=b et a>=c alors retourne a;fsi
Maxr(b,c,a);
};;
```
Ici le programme est plus difficile à comprendre car : dans le cas où `a<b ou a<c`, `Maxr(a,b,c)` appelle `Maxr(b,c,a)` qui dans le cas où `b<a ou b<c`, appelle `Maxr(c,a,b)`

8.3 Ordonner 3 nombres

On peut modifier les programmes ci-dessus pour que le résultat soit les 3 nombres ordonnés par ordre décroissant.

— On trie d'abord a, b, puis on insère c dans la liste ordonnée a, b.
On tape :
```
Tri2(a,b):={
si a>b alors retourne a,b;
sinon retourne b,a; fsi
};;
Tri3(a,b,c):={
a,b:=Tri2(a,b);
si c>=a alors retourne c,a,b;
si c<b alors retourne a,b,c;
retourne a,c,b;
};;
```
— On teste les 6 cas.
On tape :
```
Tri(a,b,c):={
si a>=b et b>=c alors retourne a,b,c; fsi
si a>=c et c>=b alors retourne a,c,b; fsi
si b>=c et c>=a alors retourne b,c,a; fsi
si b>=a et b>=c alors retourne b,a,c; fsi
si c>=a et a>=b alors retourne c,a,b; fsi
si c>=b et b>=a alors retourne c,b,a; fsi
};;
```
— On gére tous les cas avec des `si ...alors ...sinon...fsi`, mais en ordonnant au début a et b.
On tape :
```
Trii(a,b,c):={
local L,d;
si  a<b alors
d:=a;a:=b;b:=d;
fsi
si b>=c alors
```

```
    L:=a,b,c;
sinon
   si c>=a alors
    L:=c,a,b;
   sinon
    L:=a,c,b;
   fsi
fsi
retourne L;
}:;
```

— On utilise la récurivité.
On tape :

```
Trirr(a,b,c):={
si a>=b  alors
  si b>=c alors retourne a,b,c;
  sinon
    si a>=c alors retourne a,c,b;fsi
  fsi
fsi
Trirr(b,c,a);
}:;
```

Ou on tape :

```
Trir(a,b,c):={
si a>=b et b>=c alors
  retourne a,b,c;
fsi
si a>=c  et c>=b alors
  retourne a,c,b;
fsi
Trir(b,c,a);
}:;
```

8.4 Équation d'une droite définie par 2 points

On clique 2 points A et B dans l'écran de géométrie en mode `point`. On cherche l'équation de la droite AB.
Si `a1,a2` sont les coordonnées de `A`, si `b1,b2` sont les coordonnées de `B`, on cherche `a,b,c` pour que la droite d'équation `a*x+b*y+c=0` passe par `A` et `B`. Il faut donc résoudre en a, b, c :

$$\begin{aligned} aa1 + ba2 + c &= 0 \\ ab1 + bb2 + c &= 0 \end{aligned}$$

Par soustraction membre à membre on a

$$a * (a1 - b1) + b * (a2 - b2) = 0$$

On choisit : $a = a2 - b2$ et $b = -a1 + b1$ et alors $c = -a * a1 - b * a2$
On tape :

```
a1,a2:=coordonnees(A);
b1,b2:=coordonnees(B);
a:=a2-b2;
b:=-a1+b1;
c:=-a*a1-b*a2 print(a*x+b*y+c)
si (b!=0) alors y=-a/b*x-c/b; sinon a*x+b*y+c=0; fsi
```

On peut aussi l'écrire sous la forme de la fonction `Equation_droite` :

```
Equation_droite(A,B):={
 local a,a1,a2,b;b1,b2,c;
 a1,a2:=coordonnees(A);
 b1,b2:=coordonnees(B);
 a:=a2-b2;
 b:=-a1+b1;
 c:=-a*a1-b*a2
 si (b!=0) alors
  retourne y=-a/b*x-c/b;
 sinon
  retourne a*x+b*y+c=0;
 fsi
}:;
```

Remarque Dans `Xcas` la commande `equation` existe et renvoie l'équation de l'objet géométrique donné en argument.

8.5 L'automate à billet

Un distributeur de billets doit donner la somme S avec des billets de 10, 20 ou 50 euros. La somme S doit être un multiple de 10 et $S \leq 500$ euros. On impose les contraintes suivantes : Le nombre de billets de 50 euros est la partie entière de $\frac{S}{100}$.
Le nombre de billets de 20 euros est égal à 2 fois le nombre de billets de 50 euros.

1. Écrire une fonction `Automate1()` qui demande à l'utilisateur la somme S multiple de 10 $0 < S \leq 500$ et renvoie les nombres a, b, c de billets de 10, 20 et 50 euros avec ces 2 contraintes.
2. Écrire une fonction `N_automate1(S)` qui renvoie le nombre n de billets rendus avec ces 2 contraintes.
3. Tracer à l'aide d'un programme les points de coordonnées $(S; n)$, pour $0 < S \leq 500$

```
Automate1():={
local a,b,c,S;
repeter
saisir("0<S<=500 multiple de 10",S);
jusqua S<=500 et S>0 et irem(S,10)==0;
c:=floor(S/100);
```

```
b:=2*c;
a:=iquo(S-50*c-20*b,10);
print(a*10+20*b+50*c);
retourne(a,b,c);
}
:;
N_automate1(S):={
local a,b,c;
c:=floor(S/100);
b:=2*c;
a:=iquo(S-50*c-20*b,10);
retourne(a+b+c);
}
:;
Graph_automate1(S):={
local s,n,P;
P:=NULL;
pour s de 10 jusque 500 pas 10 faire
P:=P,point(s,N_automate1(s));
fpour;
retourne P;
}
:;
```

Variante
On prend comme 1 ière contrainte : Si $S \geq 100$ le nombre c de billets de 50 euros est l'entier le plus proche de $\frac{S}{100}$ et sinon ce nombre est 0.
Pourquoi ne peut-on pas garder comme 2ième contrainte : "Le nombre de billets b de 20 euros est égal à 2 fois le nombre de billets de 50 euros." ?
On rajoute alors comme 2ième contrainte : $a = b$ ou $a = b + 1$ ou $a = b - 1$ où a est le nombre de billets de 10 euros et b est le nombre de billets de 20 euros.

1. Écrire une fonction `Automate2()` qui demande à l'utilisateur la somme S multiple de 10 $0 < S \leq 500$ et qui renvoie les nombres a, b, c de billets de 10, 20 et 50 euros avec ces 2 nouvelles contraintes.
2. Écrire une fonction `N_automate2(S)` qui renvoie le nombre n de billets rendus avec ces 2 nouvelles contraintes.
3. Tracer à l'aide d'un programme les points de coordonnées $(S; n)$, pour $0 < S \leq 500$

```
Automate2():={
local a,b,c,S;
repeter
saisir("0<S<=500 multiple de 10",S);
jusqua S<=500 et S>0 et irem(S,10)==0;
si S>=100 alors
 c:=round(S/100);
sinon
```

```
 c:=0;
fsi;
S:=S-50*c;
si irem(S,30)==0 alors
   b:=iquo(S,30);print(b*10+20*b+50*c);
   retourne(b,b,c);
fsi;
si irem(S,30)==10 alors
   b:=iquo(S,30);print((b+1)*10+20*b+50*c);
   retourne(b+1,b,c);
fsi;
si irem(S,30)==20 alors
   a:=iquo(S,30);print(a*10+20*(a+1)+50*c);
   retourne(a,a+1,c);
fsi;
}
:;
N_automate2(S):={
local a,b,c;
si S>=100 alors
 c:=round(S/100);
sinon
 c:=0;
fsi;
S:=S-50*c;
si irem(S,30)==0 alors
   b:=iquo(S,30);
   retourne(2b+c);
fsi;
si irem(S,30)==10 alors
   b:=iquo(S,30);
   retourne(2b+1+c);
fsi;
si irem(S,30)==20 alors
   a:=iquo(S,30);
   retourne(2a+1+c);
fsi;

}
:;
Graph_automate2(S):={
local s,n,P;
P:=NULL;
pour s de 10 jusque 500 pas 10 faire
P:=P,point(s,N_automate2(s));
fpour;
retourne P;
}
```

```
:;
```

8.6 Un puzzle avec les pentaminos

8.6.1 L'activité

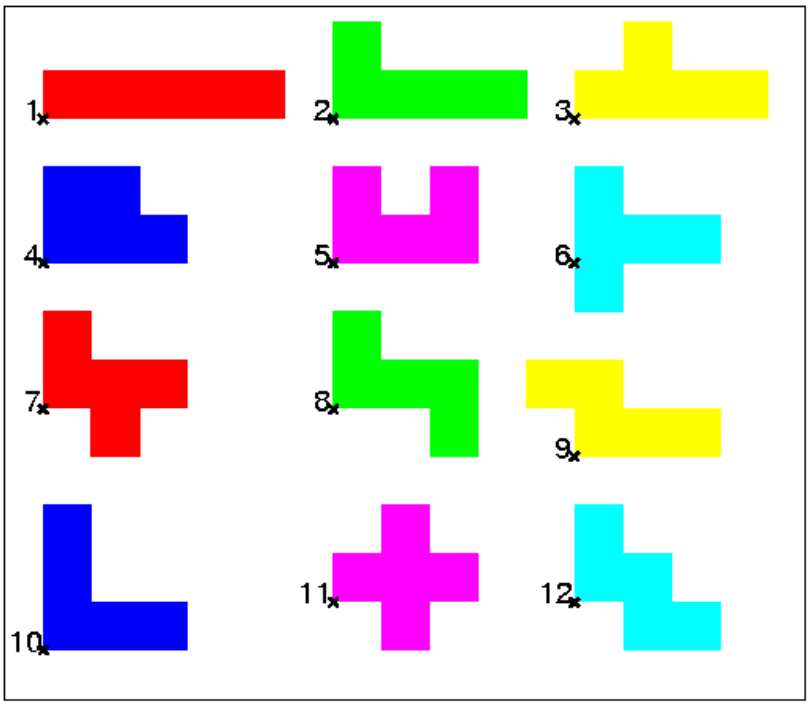
Les pentaminos s'obtiennent en juxtaposant 5 carrés unités : chaque carré doit avoir au moins un côté en commun avec un autre carré. Il y a 12 pentaminos car on s'autorise à retourner les pièces que l'on suppose colorées sur les 2 faces.
L'activité consiste :

- Dans un premier temps, à déterminer la forme des 12 pentaminos en insistant sur une recherche méthodique pour éviter les oublis. Les élèves pourront découper leurs dessins pour pouvoir par superposition éliminer les figures isométriques.
- Dans un deuxième temps, à programmer ces pièces. Pour pouvoir poser les pentaminos à un endroit précis du plan, il faut prévoir 3 paramètres : 2 paramètres `a1,a2` qui seront les coordonnées du point d'ancrage et qui permettront de changer la position de la pièce et un paramètre qui sera la couleur de la pièce. Il faut savoir que la couleur est codée par un entier `n` mais que l'on peut aussi utiliser le nom des couleurs élémentaires : 0 ou noir, 1 ou rouge, 2 ou vert, 3 ou jaune, 4 ou bleu,5 ou magenta, 6 ou cyan, 7 ou blanc.
  Ce travail permet de travailler sur le repérage d'un point dans le plan.
- Dans un troisième temps, de jouer avec ses pièces : par exemple de juxtaposer les 12 pentaminos pour obtenir un rectangle. On utilisera ensuite les transformations `rotation` pour changer l'orientation et `symetrie` pour faire un retournement.

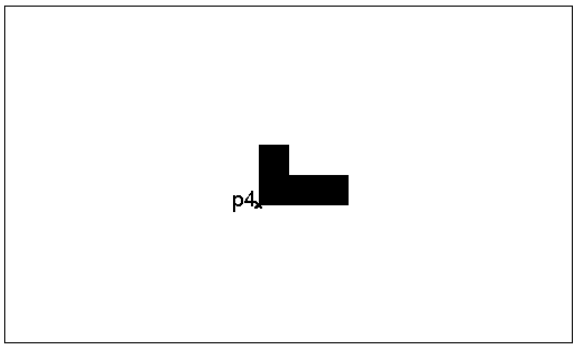
Remarques
On peut éviter, au début, d'utiliser `rotation` et `symetrie` à condition de faire le puzzle et de programmer les pièces selon la position qu'elles ont dans le puzzle...
On peut éviter, au début, d'utiliser des variables locales et des sous-procédures.

8.6.2 Programmes des pièces

Sur le dessin ci-après on a noté le point d'ancrage qui correspont aux paramètres `a1,a2` et le numéro de la pièce :



Pour obtenir les 12 pentaminos, on utilise l'instruction `carre` et `rectangle` et on définit la forme `p4` constituée de 4 carrés et que l'on retrouve dans plusieurs pentaminos :



On peut bien sûr n'utiliser que `carre` pour programmer les 12 pentaminos. On tape :

```
penta1(a1,a2,c):={
local P;
P:=rectangle(point(a1,a2),point(a1+5,a2),1/5);
retourne affichage(P,rempli+c);
}
:;
```

```
penta2(a1,a2,c):={
local P;
P:=rectangle(point(a1,a2),point(a1+4,a2),1/4),
carre(point(a1,a2+1),point(a1+1,a2+1));
retourne affichage(P,rempli+c);
}
:;
penta3(a1,a2,c):={
local P;
P:=rectangle(point(a1,a2),point(a1+4,a2),1/4),
carre(point(a1+1,a2+1),point(a1+2,a2+1));
retourne affichage(P,rempli+c);
}
:;
p4(a1,a2):={
local P;
P:=rectangle(point(a1,a2),point(a1+3,a2),1/3),
carre(point(a1,a2+1),point(a1+1,a2+1));
retourne P;
}:;
penta4(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1+1,a2+1),point(a1+2,a2+1));
retourne affichage(P,rempli+c);
}:;
penta5(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1+2,a2+1),point(a1+3,a2+1));
retourne affichage(P,rempli+c);
}:;
penta6(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1,a2-1),point(a1+1,a2-1));
retourne affichage(P,rempli+c);
}:;
penta7(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1+1,a2-1),point(a1+2,a2-1));
retourne affichage(P,rempli+c);
}:;
penta8(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1+2,a2-1),point(a1+3,a2-1));
retourne affichage(P,rempli+c);
}:;
penta9(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1-1,a2+1),point(a1,a2+1));
```

```
retourne affichage(P,rempli+c);
}:;
penta10(a1,a2,c):={
local P;
P:=p4(a1,a2),carre(point(a1,a2+2),point(a1+1,a2+2));
retourne affichage(P,rempli+c);
}:;
penta11(a1,a2,c):={
local P;
P:=rectangle(point(a1,a2),point(a1+3,a2),1/3),
carre(point(a1+1,a2+1),point(a1+2,a2+1)),
carre(point(a1+1,a2-1),point(a1+2,a2-1));
retourne affichage(P,rempli+c);
}:;
penta12(a1,a2,c):={
local P;
P:=carre(point(a1,a2+1),point(a1+1,a2+1)),
rectangle(point(a1,a2),point(a1+2,a2),1/2),
rectangle(point(a1+1,a2-1),point(a1+3,a2-1),1/2);
retourne affichage(P,rempli+c);
}:;
```

On peut aussi ne pas introduire la variable locale `P`, mais quand même utiliser la sous-procédure `p4` :

```
penta1(a1,a2,c):=
   affichage(rectangle(point(a1,a2),point(a1+5,a2),1/5),
             rempli+c):;
penta2(a1,a2,c):=
   affichage(rectangle(point(a1,a2),point(a1+4,a2),1/4),
             carre(point(a1,a2+1),point(a1+1,a2+1)),
             rempli+c):;
penta3(a1,a2,c):=
   affichage(rectangle(point(a1,a2),point(a1+4,a2),1/4),
             carre(point(a1+1,a2+1),point(a1+2,a2+1)),
             rempli+c):;
p4(a1,a2):={
local P;
P:=rectangle(point(a1,a2),point(a1+3,a2),1/3),
carre(point(a1,a2+1),point(a1+1,a2+1));
retourne P;
}:;
penta4(a1,a2,c):=
   affichage(p4(a1,a2),
             carre(point(a1+1,a2+1),point(a1+2,a2+1)),
             rempli+c):;
penta5(a1,a2,c):=
   affichage(p4(a1,a2),
             carre(point(a1+2,a2+1),point(a1+3,a2+1)),
```

```
              rempli+c):;
penta6(a1,a2,c):=
   affichage(p4(a1,a2),
              carre(point(a1,a2-1),point(a1+1,a2-1)),
              rempli+c):;
penta7(a1,a2,c):=
   affichage(p4(a1,a2),
              carre(point(a1+1,a2-1),point(a1+2,a2-1)),
              rempli+c):;
penta8(a1,a2,c):=
   affichage(p4(a1,a2),
              carre(point(a1+2,a2-1),point(a1+3,a2-1)),
              rempli+c):;
penta9(a1,a2,c):=
   affichage(p4(a1,a2),
              carre(point(a1-1,a2+1),point(a1,a2+1)),
              rempli+c):;
penta10(a1,a2,c):=
   affichage(p4(a1,a2),
              carre(point(a1,a2+2),point(a1+1,a2+2)),
              rempli+c):;
penta11(a1,a2,c):=
   affichage(rectangle(point(a1,a2),point(a1+3,a2),1/3),
              carre(point(a1+1,a2+1),point(a1+2,a2+1)),
              carre(point(a1+1,a2-1),point(a1+2,a2-1)),
              rempli+c):;
penta12(a1,a2,c):=
   affichage(carre(point(a1,a2+1),point(a1+1,a2+1)),
              rectangle(point(a1,a2),point(a1+2,a2),1/2),
              rectangle(point(a1+1,a2-1),point(a1+3,a2-1),1/2),
              rempli+c):;
```

Voici, à titre d'exemple pour le professeur, ce qui a été tapé pour obtenir les pièces avec leurs légendes : `depart()` permet de dire où se font les points d'ancrage des pièces et `pieces()` dessine les pièces avec leur point d'ancrage.
On peut aussi ne pas définir de fonctions et mettre seulement la suite des instructions par exemple :

```
point(-8,5,affichage=quadrant2+epaisseur_point_2,legend="1"),
penta1(-8,5,1)
```

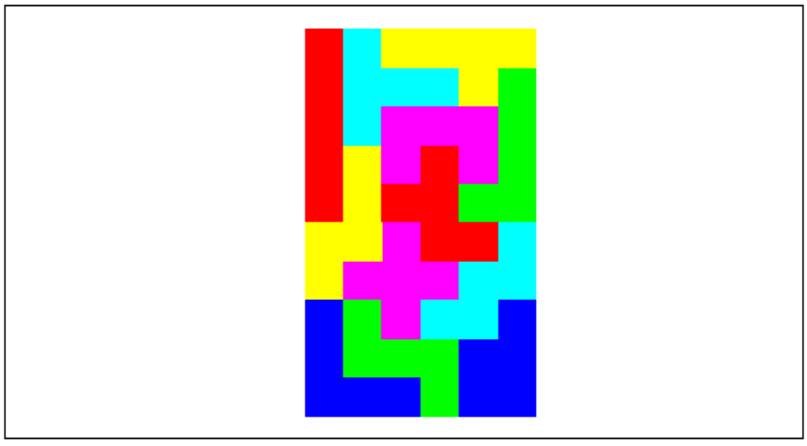
définit le pentamino1 avec son point d'ancrage.

```
depart():={
point(-8,5,affichage=quadrant2+epaisseur_point_2,legend="1"),
point(-2,5,affichage=quadrant2+epaisseur_point_2,legend="2"),
point(3,5,affichage=quadrant2+epaisseur_point_2,legend="3"),
point(-8,2,affichage=quadrant2+epaisseur_point_2,legend="4"),
point(-2,2,affichage=quadrant2+epaisseur_point_2,legend="5"),
point(3,2,affichage=quadrant2+epaisseur_point_2,legend="6"),
```

```
point(-8,-1,affichage=quadrant2+epaisseur_point_2,legend="7"),
point(-2,-1,affichage=quadrant2+epaisseur_point_2,legend="8"),
point(3,-2,affichage=quadrant2+epaisseur_point_2,legend="9"),
point(-8,-6,affichage=quadrant2+epaisseur_point_2,legend="10"),
point(-2,-5,affichage=quadrant2+epaisseur_point_2,legend="11"),
point(3,-5,affichage=quadrant2+epaisseur_point_2,legend="12");
}:;
pieces():={
penta1(-8,5,rouge),penta2(-2,5,vert),penta3(3,5,jaune),
penta4(-8,2,bleu),penta5(-2,2,magenta),penta6(3,2,cyan),
penta7(-8,-1,rouge),penta8(-2,-1,vert),penta9(3,-2,jaune),
penta10(-8,-6,bleu),penta11(-2,-5,magenta),penta12(3,-5,cyan),
depart();
}:;
```

8.6.3 Le puzzle du rectangle 6×10

La construction du rectangle 6×10 en juxtaposant les 12 pentaminos est le "Problème Fondamental du Jeu de Pentaminos". Il y a 2339 solutions, mais pour obtenir une solution, ce n'est pas si simple !
Voici deux solutions et leurs programmes au "Problème Fondamental". On remarquera qu'avec `Xcas` les objets géométriques perdent leurs attributs lorsqu'ils sont transformés par les instructions `rotation` ou `symetrie`, on doit donc les spécifier avec une commande `affichage`, par exemple on mettra :
`affichage(rotation(6,pi/2,penta4(6,0,4)),rempli+4)` ou
`affichage(rotation(6,pi/2,penta4(6,0,bleu)),rempli+bleu)`
On tape les fonctions `puzzle1()` et `puzzle2()` mais on peut aussi ne pas définir de fonctions et mettre seulement la suite des instructions ce qui permet de travailler avec des élèves qui n'ont pas vu les fonctions.
Voici le puzzle1 :

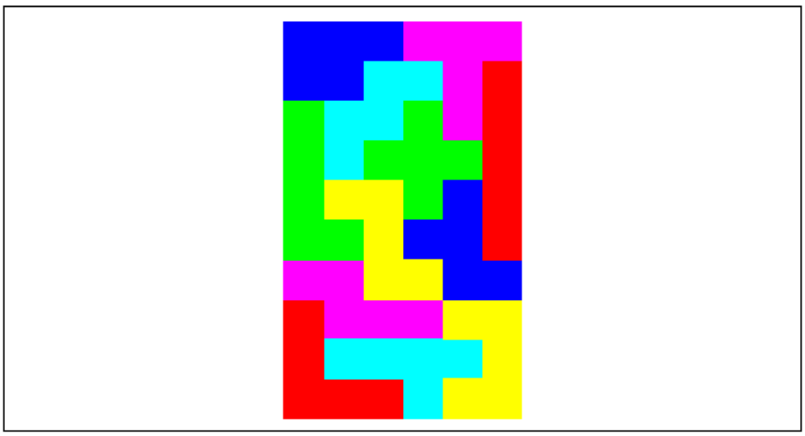


On tape :

```
puzzle1():={
```

```
penta10(0,0,bleu),penta8(1,1,vert),
affichage(rotation(6,pi/2,penta4(6,0,4)),rempli+bleu),
penta11(1,3,magenta),
affichage(rotation(2+4*i,pi/2,penta9(2,4,3)),rempli+jaune),
affichage(rotation(5+2*i,pi/2,penta12(5,2,6)),rempli+cyan),
affichage(rotation(1+5*i,pi/2,penta1(1,5,1)),rempli+rouge),
penta6(1,8,cyan),
affichage(symetrie(droite(3,3+i),
          rotation(3+4*i,pi/2,penta7(3,4,1))),rempli+rouge),
affichage(symetrie(droite(8*i,3+8*i),penta5(2,8,5)),
          rempli+magenta),
affichage(rotation(6+5*i,pi/2,penta2(6,5,2)),rempli+vert),
affichage(rotation(6+10*i,pi,penta3(6,10,3)),rempli+jaune);
}:;
```

Voici le puzzle2 :

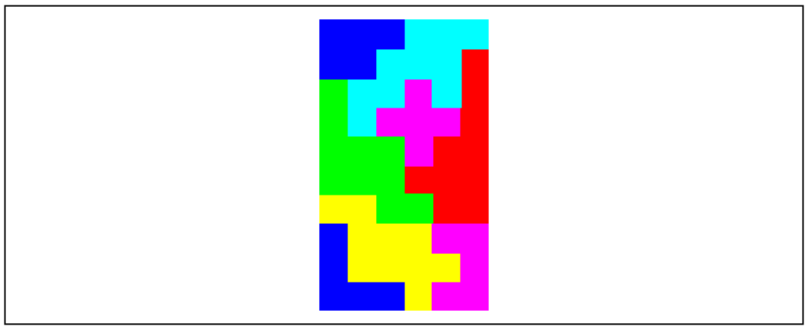


On tape :

```
puzzle2():={
penta10(0,0,rouge),penta9(1,2,magenta),
affichage(rotation(5+2*i,pi,penta3(5,2,6)),rempli+cyan),
affichage(rotation(6,pi/2,penta5(6,0,3)),rempli+jaune),
affichage(symetrie(droite(0,i),
          rotation(4*i,pi/2,penta2(0,4,2))),rempli+vert),
affichage(symetrie(droite(4,4+i),
          rotation(4+3*i,pi/2,penta7(4,3,4))),rempli+bleu),
affichage(symetrie(droite(2,2+i),
          rotation(2+3*i,pi/2,penta8(2,3,3))),rempli+jaune),
penta11(2,6,vert),
affichage(rotation(6+4*i,pi/2,penta1(6,4,1)),rempli+rouge),
affichage(rotation(2+9*i,-pi/2,penta12(2,9,6)),rempli+cyan),
affichage(symetrie(droite(10*i,1+10*i),penta4(0,10,4)),
          rempli+bleu),
```

```
affichage(rotation(4+10*i,-pi/2,penta6(4,10,5)),
          rempli+magenta);
}:;
```

ou bien si veut que pour `n=1..6`, la pièce `n` et la pièce `n+6` soient de couleur `n`, le puzzle2 devient le puzzle2b :



```
puzzle2b():={
penta10(0,0,4),penta9(1,2,3),
affichage(rotation(5+2*i,pi,penta3(5,2,3)),rempli+3),
affichage(rotation(6,pi/2,penta5(6,0,5)),rempli+5),
affichage(symetrie(droite(0,i),
                rotation(4*i,pi/2,penta2(0,4,2))),rempli+2),
affichage(symetrie(droite(4,4+i),
                rotation(4+3*i,pi/2,penta7(4,3,1))),
          rempli+1),
affichage(symetrie(droite(2,2+i),
                rotation(2+3*i,pi/2,penta8(2,3,2))),
          rempli+2),
penta11(2,6,5),
affichage(rotation(6+4*i,pi/2,penta1(6,4,1)),rempli+1),
affichage(rotation(2+9*i,-pi/2,penta12(2,9,6)),rempli+6),
affichage(symetrie(droite(10*i,1+10*i),penta4(0,10,4)),
          rempli+4),
affichage(rotation(4+10*i,-pi/2,penta6(4,10,6)),rempli+6);
}:;
```

Remarque

Tous les rectangles de dimension 3×20, 4×15, 5×12 et 6×10 peuvent être construits :

- de 2 façons pour le rectangle 3×20,
- de 368 façons pour le rectangle 4×15,
- de 1010 façons pour le rectangle 5×12,
- de 2339 façons pour le rectangle 6×10.

A vous de jouer ! ! ! !

8.7 Programmer un jeu

On considère le jeu suivant :

— Le joueur tire au hasard un nombre entre 1 et 10. Ce nombre est son gain, il peut le garder et la partie est finie.

— Le joueur peut choisir d'abandonner ce gain et de recommencer. Il lui est permis de tirer ainsi au plus 5 fois et son gain est le résultat du dernier tirage lorsqu'il a abandonné les gains des 4 tirages précédents.

1. Écrire un programme qui réalise ce jeu,
2. Le joueur choisit comme stratégie de garder son gain si celui-ci est supérieur ou égal à 8 sinon il recommence.
   Il joue 100 parties selon cette stratégie.
   Écrire un programme qui donne la moyenne des gains obtenus.
3. Le joueur choisit 4 nombres n_1, n_2, n_3, n_4 compris entre 1 et 10 et choisit comme stratégie de garder son gain du $j$ième essai si celui-ci est supérieur ou égal à n_j sinon il recommence.
   Il joue 100 parties selon cette stratégie avec les mêmes n_j.
   Exemple de parties avec n_1=9, n_2=8, n_3=6, n_4=5 : On tire 2,7,8 et le gain est 8 mais si on tire 8,7,5,4,2 le gain est 2 Écrire un programme qui donne la moyenne des gains obtenus.

On demande à l'utilisateur de répondre *o* pour dire oui et n'importe quoi d'autre pour dire non !

```
jeu1():={
local j,g,rep;
for (j:=1;j<=4;j++) {
g:=rand(10)+1;
print(g);
lis(rep);
if (rep==o) retourne g;
}
retourne rand(10)+1;
}
:;
```

`jeu102` est la moyenne des gains de 100 parties faites avec `jeu2`.

```
jeu2():={
local j,g;
for (j:=1;j<=5;j++) {
g:=rand(10)+1;
print(g);
if (g>=8) break;
}
retourne g;
}
:;
jeu102():={
```

```
local g,j;
g:=0;
for(j:=1;j<=100;j++) {
g:=g+jeu2();
}
retourne evalf(g/100);
}
:;
```

`jeu3` a comme paramètre `L` qui est la séquence n_1, n_2, n_3, n_4 puis LL est la suite $n_1, n_2, n_3, n_4, 0$ de façon à ce que $LL[j]$ soit définie pour $j = 0..4$ (Attention $LL[0] = n_1$).
`jeu103` est la moyenne des gains de 100 parties faites avec `jeu3`.

```
jeu3(L):={
local j,g,LL;
LL:=L,0;
for (j:=0;j<=4;j++) {
g:=rand(10)+1;
print(g);
if (g>=LL[j]) break;
}
retourne g;
}
:;
jeu103(L):={
local g,j;
g:=0;
for(j:=1;j<=100;j++) {
g:=g+jeu3(L);
}
retourne evalf(g/100);
}:;
```

8.8 Programmer le jeu de la marguerite

L'énoncé du jeu Un marguerite a 11 pétales numéroté de 0 à 10. Sur chaque pétale on pose un pion.
Ce jeu se joue à 2 joueurs : chaque joueur peut enlever un pion ou 2 pions si ils sont contigüs. Le joueur qui enlève le dernier pion ou les 2 derniers pion a gagné.
Il existe une stratégie gagnante pour le deuxième joueur.
Pour la trouver, on va considérer des marguerites ayant n pétales avec $n = 3..11$.
Pour $n = 3$ c'est évident, le 2ième joueur gagne.
Pour $n = 4$ si le 1ier joueur enlève 1 pion, il reste 3 pions, pour gagner le 2ième joueur doit enlever le pion du milieu pour ne pas laisser 2 pions côte à côte et si le 1ier joueur enlève 2 pions, il reste 2 pions que le 2ième joueur enlève et gagne.
Pour $n = 5$ si le 1ierjoueur enlève 1 pion, il reste 4 pions, pour gagner le 2ième joueur doit enlever les 2 pions du milieu pour ne pas laisser 2 pions côte à côte et

si le 1ier joueur enlève 2 pions, il reste 3 pions le 2ième joueur doit enlever le pion du milieu pour gagner.
Au cours du premier tour, le 2ième joueur doit séparer les pions en 2 ensembles symétriques et donc de même cardinal. Puis, le 2ième joueur joue de façon à garder la symétrie entre ces 2 ensembles.
La symétrie est la symétrie par rapport au centre de la marguerite et aussi la symétrie par rapport à l'axe déterminé par le 1ier tour.

La programmation du jeu

- Faire le programme qui réalise le dessin d'une marguerite ayant 11 pétales numérotés de 0 à 10 avec des pions noirs sur les pétales dont les numéros sont dans l'ensemble P et des pions blancs sur les autres pétales.
- Un joueur joue contre l'ordinateur et commence à jouer. L'ordinateur applique la stratégie gagnante à savoir :

  1. Au premier coup l'ordinateur enlève les 2 pions situés de part et d'autre du symétrique (par rapport au centre O de la marguerite) du pion enlevé par le joueur ou enlève le pion situé selon le symétrique du milieu des 2 pions par rapport au centre O de la marguerite.
     Par exemple si le joueur enlève le 0 l'ordinateur enlève le 5 et le 6 : il reste alors 2 groupes de 4 pions 1,2,3,4 et 7,8,9,10 (ces 2 groupes sont symétriques par rapport à l'axe $O0$) et si le joueur enlève le 0 et le 1 l'ordinateur enlève le 6 : il reste alors 2 groupes de 4 pions 2,3,4,5 et 7,8,9,10 (ces 2 groupes sont symétriques par rapport à l'axe $O6$).
     Ce premier tour détermine l'axe du jeu : dans les exemples précédents c'est $O1$ et $O6$.
     Faire le programme qui réalise le premier coup.
  2. Aux coups suivants l'ordinateur enlève le ou les pions symétriques par rapport à l'axe du jeu du ou des pions enlevés par le joueur. Par exemple si l'axe est $O1$, si le joueur enlève le 2, l'ordinateur enlève le 9 et il reste alors 1,3,4 7,8,10 et si le joueur enlève le 2 et le 3, l'ordinateur enlève le 8 et le 9. Il reste alors 1,4 7,10. Dans ce dernier cas le joueur est maintenant obligé d'enlever un seul pion car les pions restants ne sont pas contigüs.
     Faire le programme qui réalise les coups suivants.

Les programmes solutions.

```
// margot(z,P) est le dessin de la marguerite de centre z
// pion noir selon l'ensemble P
margot(z,P):={
 local z0,L,k;
 cercle(z,1);
 z0(k):=exp(2*i*k*pi/11.);
 L:=ellipse(z+6/5*z0(k),z+5*z0(k),point(z+z0(k)))$(k=0..10);
 L:=L,cercle(z+4*z0(k),0.5)$(k=0..10);
 pour k in P faire
   L:=L,cercle(z+4*z0(k),0.5,affichage=rempli);
 fpour;
 L:=L,legende(z-0.5-i*0.5+5.8*z0(k),k)$(k=0..10);
```

```
 return L;
}:;
// coteacote(L,p) test si il y a des elements contigus ds L mod p
coteacote(L,p):={
  local s,R,j;
  L:=sort(L);
  s:=size(L)-1;
  R:=L[0]+p-L[s];
  pour j de 1 jusque s faire
    R:=R,L[j]-L[j-1];
  fpour;
  si count_eq(1,[R])==0 alors
    return 0
  fsi;
  return 1;
}:;
//jeumargot() : vous jouez contre l'ordi
jeumargot():={
local k,z,P,rep,pos,n,p,q,M,np,n0;
P:=set[k$(k=0..10)];
M:=NULL;
pos:=0;
rep:=0;
ClrGraph();
gl_x=-10..10;
gl_y=-7.18..7.18;
axes=0;
DispG;
margot(pos,P);
repeat
  saisir("nbre de pion", np);
until np==1 ou np==2;
afficher(np);
si np==1 alors
  repeat
    saisir("tapez n",n)
  until (set[n] intersect P)==set[n];
  afficher(n);
  P:=P minus set[n];
  p:=irem(n+5,11);
  q:=irem(n+6,11);
sinon
  si np==2 alors
     afficher("tapez p,q avec |p-q|=1");
     repeat
       saisir("tapez p,q (|p-q|=1)",p,q);
     until abs(p-q)==1 and (set[p,q] intersect P)==set[p,q];
     afficher(p,q);
```

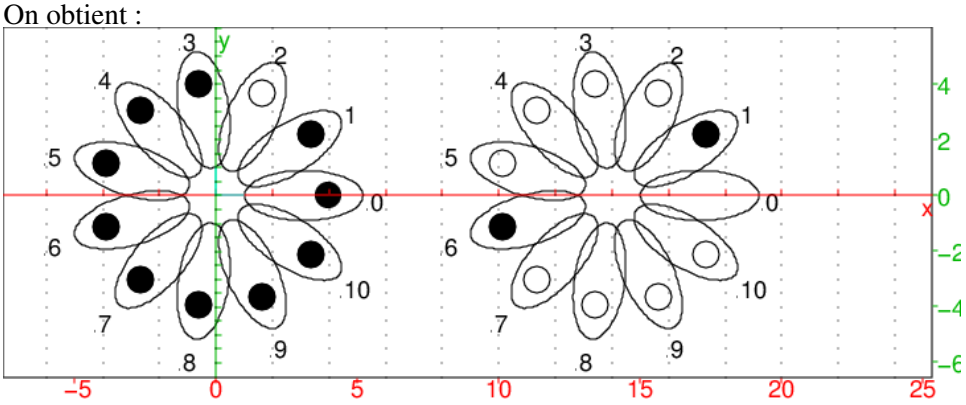
```
      P:=P minus set[p,q];
      n:=irem((p+q-1)/2-5,11);
  fsi;
fsi;
M:=NULL;
M:=M,margot(pos,P);
ClrGraph();
gl_x=-7..50;
gl_y=-35.5..10.49;
axes=0;
DispG;
margot(pos,P);
pos:=pos+14;
si np==1 alors
  afficher("l'ordi enleve les pions",p,q);
  P:=P minus set[p,q];
sinon
  afficher("l'ordi enleve le pion",n);
  P:=P minus set[n];
fsi;
M:=M,margot(pos,P);
DispG;
margot(pos,P);;
rep:=rep+1;
n0:=n;
tantque P!=set[NULL] faire
  si coteacote([op(P)],11) alors
    repeat
      saisir("nbre de pion", np);
    until np==1 ou np==2;
    afficher(np);
  sinon
    np:=1;
    afficher(np);
  fsi;
  si np==1 alors
    repeat
      saisir("tapez n",n)
    until (set[n] intersect P)==set[n];
    afficher(n);
    P:=P minus set[n];
    n:=irem(2*n0-n,11);
  sinon
    si np==2 alors
      afficher("tapez p,q avec |p-q|=1");
      repeat
        saisir("tapez p,q (|p-q|=1)",p,q);
      until abs(p-q)==1 and (set[p,q] intersect P)==set[p,q];
```

```
      afficher(p,q);
    P:=P minus set[p,q];
    p:=irem(2*n0-p,11);
    q:=irem(2*n0-q,11);
    fsi;
  fsi;
  pos:=pos+14;
  M:=M,margot(pos,P);
  DispG;
  margot(pos,P);
  si np==1 alors
    afficher("l'ordi enleve le pion",n);
    P:=P minus set[n];
  sinon
    afficher("l'ordi enleve les pions",p,q);
    P:=P minus set[p,q];
  fsi;
  pos:=pos+14;
  M:=M,margot(pos,P);
  DispG;
  margot(pos,P);
  rep:=rep+1;
  si rep==2 alors
    pos:=-14-i*14;
  sinon
    si rep==4 alors
      pos:=-14-i*28;
    fsi;
  fsi
ftantque;
saisir(np);
afficher("l'ordi a gagné en",rep,"coups");
return M;
}:;
```

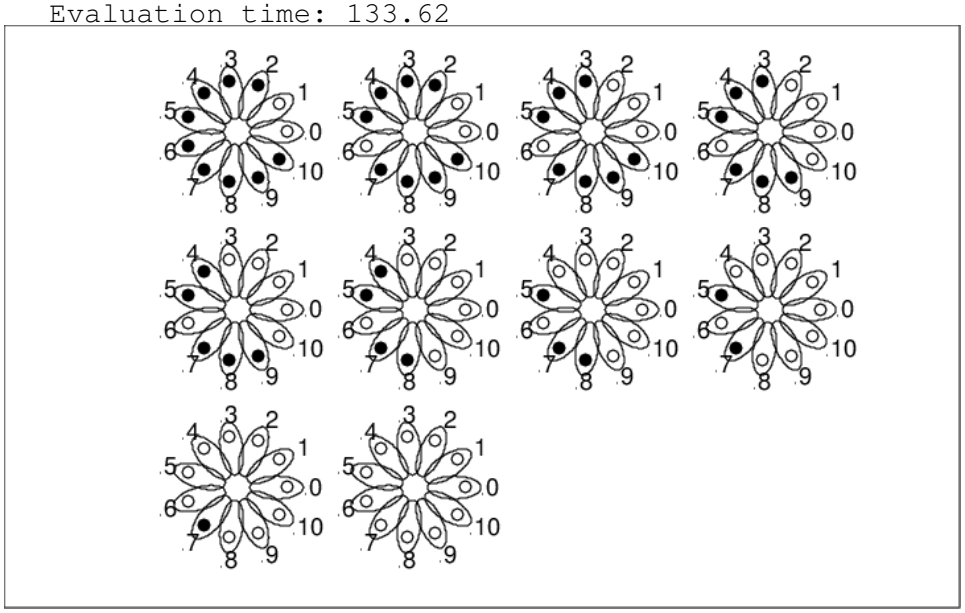
On tape :

```
margot(0,[0,1,3,4,5,6,7,8,9,10]),margot(14,[1,6])
```

On obtient :



On tape :
```
jeumargot()
```
On obtient en vert par exemple :
```
"nbre de pion" np:2 "tapez p,q avec |p-q|=1" 0,1 "l'ordi
enleve le pion",6 "nbre de pion" np:1 n:2 "l'ordi enleve
le pion",10 "nbre de pion" np:1 n:3 "l'ordi enleve le pion",9
   np:1 n:4 "l'ordi enleve le pion",8 n:5 "l'ordi enleve
le pion",7 "l'ordi a gagné en",5,"coups"
   Evaluation time: 133.62
```



8.9 Un jeu de hasard

Dans un casino particulier on joue à un jeu de hasard avec n jetons et un capital C de 1 euro représenté par une plaque de 1 euro (que l'on doit, bien sûr, acheter au prix de $p(n)+1$).
Au début, chaque joueur dispose de n jetons et d'un capital C de 1 euro. Le joueur doit enchainer plusieurs parties jusqu'à épuisement de ses jetons, les jetons mis en jeu étant perdus.
Chaque partie consiste à miser 1 ou plusieurs jetons au jeu de pile ou face. Si il perd, son capital est inchangé et si il gagne, son capital est multiplié par le nombre de jetons mis en jeu.
Par exemple, si il gagne les 2 premières parties en misant 2 jetons, puis 3 jetons son capital sera représenté par une plaque de 6 euros.
Si $n=4$ quelle est la stratégie qui vous donne la plus grande espérance de gains ?
Si $n=8$ quelle est la stratégie qui vous donne la plus grande espérance de gains ?
Si $n=40$ quelle est la stratégie qui vous donne la plus grande espérance de gains ?
Solution
Remarques

Il est inutile de jouer un jeton puisque cela ne change pas le capital C
L'ordre n'est pas important car jouer 5 jetons puis 3 jetons le capital sera modifié avec la même probabilité.
Pour $n = 4$
On a 2 écritures différentes de 4 (on élimine 4=1+3=3+1) :

1. 4=4, on joue 1 seule partie et l'espérance de la valeur du capital C est $1 * 1/2 + 4 * 1/2 = 5/2 = 2.5$

2. 4=2+2, on joue 2 parties et l'espérance de la valeur du capital C est $1 * 1/2^2 + 2 * 2 * 1/2^2 + 4 * 1/2^2 = 3^2/2^2 = 9/4 = 2.25$.

Il faut choisir comme stratégie de jouer une seule fois les 4 jetons car vous obtiendrez en moyenne un capital C de 2.5 euros avec la possibilité d'obtenir un capital maximum de 4 euros avec une probabilité de 1/2.
Pour $n = 8$
On a différentes écritures de 8 :

1. 8=8 (on joue 1 seule partie)

2. 8=2+6 (on joue 2 parties)

3. 8=3+5 (on joue 2 parties)

4. 8=4+4 (on joue 2 parties)

5. 8=2+3+3 (on joue 3 parties)

6. 8=2+2+4 (on joue 3 parties)

7. 8=2+2+2+2 (on joue 4 parties)

Calculons l'espérance de la valeur du capital C pour chacun des cas

1. 8=8, on joue 1 seule partie et $C = 1$ ou $C = 8$
   $E(C) = 1 * 1/2 + 8 * 1/2 = 9/2 = (1 + 8)/2 = 4.5$
2. 8=2+6, on joue 2 parties et $C = 1$ ou $C = 2$ ou $C = 6$ ou $C = 12$
   $E(C) = 1 * 1/2^2 + 2 * 1/2^2 + 6 * 1/2^2 + 12 * 1/2^2 = 21/4 = (1 + 2) * (1 + 6)/2^2 = 5.25$
3. 8=3+5, on joue 2 parties et $C = 1$ ou $C = 3$ ou $C = 5$ ou $C = 15$
   $E(C) = 1 * 1/2^2 + 3 * 1/2^2 + 5 * 1/2^2 + 15 * 1/2^2 = 21/4 = (1 + 3) * (1 + 5)/2^2 = 6$
4. 8=4+4, on joue 2 parties et $C = 1$ ou $C = 4$ ou $C = 16$
   $E(C) = 1 * 1/2^2 + C_2^1 * 4 * 1/2^2 + 16 * 1/2^2 = (1 + 4)^2/4 = 25/4 = 6.25$
5. 8=2+3+3, on joue 3 parties et $C = 1$ ou $C = 2$ ou $C = 3$ ou $C = 6$ ou $C = 9$ ou $C = 18$
   $E(C) = 1 * 1/2^3 + 2 * 1/2^3 + 2 * 3 * 1/2^3 + 2 * 6 * 1/2^3 + 9 * 1/2^3 + 18 * 1/2^3 = (1 + 2) * (1 + 3)^2/2^3 = 6$

6. 8=2+2+4, on joue 3 parties et $C = 1$ ou $C = 2$ ou $C = 4$ ou $C = 8$ ou $C = 16$
   $E(C) = 1*1/2^3+2*2*1/2^3+4*1/2^3+4*1/2^3+2*8*1/2^3+16*1/2^3 = 45/8 = (1+4)*(1+2)^2/2^3 5.625$
7. 8=2+2+2+2 on joue 4 parties et $C = 1$ ou $C = 2$ ou $C = 4$ ou $C = 8$ ou $C = 16$
   = $E(C) = 1*1/2^4+C_4^1*2*1/2^4+C_4^2*4*1/2^4+C_4^3*8*1/2^4+16*1/2^4 = (1+2)^4/2^4 = 81/16 = 5.0625$

Conclusion
Il faut choisir comme stratégie de jouer 2 fois 4 jetons car vous obtiendrez en moyenne un capital C de 6.25 euros avec la possibilité d'obtenir un capital maximum de 16 euros avec une probabilité de 1/4.
Vous pouvez choisir de jouer 3 fois : 2 jetons, puis 3 jetons, puis 3 jetons, vous obtiendrez en moyenne un capital C de 6 euros avec la possibilité d'obtenir un capital maximum de 18 euros avec une probabilité de 1/8.

Pour $n = 40$
Il y a beaucoup trop d'écritures différentes de 40.
On va chercher, tout d'abord, les espérances de la valeur du capital C lorsqu'on joue toujours le même nombre de jetons.
Lorsqu'on joue avec n jetons et que $n = p * k$, cherchons l'espérance de la valeur du capital C, lorsqu'on fait k parties en misant p jetons.
Les valeurs possibles de C sont :
$[1, p, p^2, p^3...p^k]$ qui ont comme probabilité respectivement :
$displaystyle[\frac{1}{2^k}, C_k^1\frac{1}{2^k}, C_k^2\frac{1}{2^k}, C_k^3\frac{1}{2^k}...C_k^k\frac{1}{2^k}]$
Donc l'espérance de C est :
$displaystyle\frac{1}{2^k} + C_k^1\frac{p}{2^k}, C_k^2\frac{p^2}{2^k}, C_k^3\frac{p^3}{2^k}...C_k^k\frac{p^k}{2^k} = \frac{(1+p)^k}{2^k}$

1. 40=1*40, on joue 1 seule partie
   On a :
   $E(C) = 20$
2. 40=2*20, on joue 2 parties
   On a :
   $E(C) = 21^2/2^2 = 110.25$
3. 40=4*10 on joue 4 parties
   On a :
   $E(C) = 11^4/2^4 = 915.0625$
4. 40=5*8, on joue 5 parties
   On a :
   $E(C) = 9^5/2^5 = 1845.28125$
5. 40=8*5, on joue 8 parties
   On a :
   $E(C) = 6^8/2^8 = 3^8 = 6561.0$
6. 40=10*4, on joue 10 parties
   On a :

$E(C) = 5^{10}/2^{10} = 9536.74316406$

7. 40=20*2, on joue 20 parties
   On a :
   $E(C) = 3^{20}/2^{20} = 3325.25673008$

Donc, lorsqu'on joue toujours le même nombre de jetons, la meilleure stratégie est de jouer 10 parties de 4 jetons car vous obtiendrez en moyenne un capital C de 9536.74 euros avec la possibilité d'obtenir un capital maximum de $4^{10} = 1048576$ euros avec une probabilité de $1/2^{10} = 1/1024$.
Remarque : pourquoi 4 joue-il un role particulier ?
Lorsqu'on a n jetons :
si $n = 2$ ou $n = 3$ on n'a pas le choix on ne peut faire qu'une partie.
si $n = 4$ on a vu qu'il etait préférable de ne faire qu'une partie,
si $n > 4$ Que faut-il choisir pour avoir un capital C ayant la plus grande espérance jouer 1 partie de n jetons ou 2 parties de $n - 2$ jetons puis de 2 jetons ? pour 1 partie $E(C_1) = (n + 1)/2)$ et
pour 2 partie $E(C_2) = 3 * (n - 1)/4)$
On a $E(C_1) \leq E(C_2)$ est équivalent à $2(n + 1) \leq 3n - 3$ ce qui est équivalent à $5 \leq n$.
Donc on ne peut considérer dans la décompostion de n que les nombres $4, 3, 2$.
Faire cette remarque au début nous aurait éviter des calculs ! ! ! **Calcul de** $E(C)$ **dans le cas général** Lorsqu'on joue avec n jetons et que $n = k * p + m * q$, cherchons l'espérance de la valeur du capital C, lorsqu'on fait k parties en misant p jetons et m parties en misant q jetons.
Les valeurs possibles de C sont :
$[p^l * q^j]$ lorsque $l = 0..k$ et $j = 0.m$ qui ont comme probabilité :
$displaystyle[\frac{C_k^l * C_m^j}{2^{k+m}}]$
Donc l'espérance de C est :
$displaystyle \sum_{l=0}^{k} \sum_{j=0}^{m} \frac{C_k^l * C_m^j * p^l * q^j}{2^{k+m}} = \sum_{l=0}^{k} C_k^l * p^l \sum_{j=0}^{m} \frac{C_m^j * q^j}{2^{k+m}} =$
$displaystyle \sum_{l=0}^{k} C_k^l * p^l \frac{(1+q)^m}{2^{k+m}} = \frac{(1+p)^k * (1+q)^m}{2^{k+m}}$
De même si $n = k_1 p_1 + k_2 p_2 + ...k_l p_l$, lorsque pour $j = 1..l$, on fait,k_j parties en misant p_j jetons, la valeur du capital C est $displaystyle \frac{\Pi_{j=1}^{l} (1+p_j)^{k_j}}{2^{\sum_{j=1}^{l} k_j}}$.
Donc :

$$E(C_n) = E(C)$$

D'après l'étude faite pour $n = 4$, on n'a pas intérêt à remplacer une partie en jouant 4 jetons par deux parties de 2 jetons.
D'après l'étude faite pour $n = 8$, on n'a pas intérêt à remplacer 2 parties de 4 jetons par une autre décomposition.
A-t-on intérêt a remplacer 3 parties de 4 jetons par une autre stratégie ?
Les décompositions de 12 ne contenant ni 1, ni 4, ni 2+2 sont :
12=2+10=3+9=5+7=6+6
12=2+3+7=2+5+5=2+3+7=3+3+6
12=3+3+3+3
D'après la remarque on ne peut considérer dans la décompostion de n que les nombres $4, 3, 2$.
Cherchons quand même les différentes valeurs de $E(C$.

Les espérances de C, à comparer avec $5^3/2^3 = 15.625$, sont respectivement :
$11 * 3/4 = 33/4 = 8.25$, $10 * 4/4 = 10$, $8 * 6/4 = 12$; $7 * 7/4 = 49/4 = 12.25$,
$3 * 4 * 8/8 = 12$, $3 * 6^2/8 = 27/2 = 13.5$, $3 * 4 * 8/8 = 12$, $4^2 * 7/8 = 14$
$4^4/2^4 = 16$
Donc la meilleure stratégie est de remplacer 3 parties de 4 jetons par 4 parties de 3 jetons.
Puisque $40 = 12 * 3 + 4$, la meilleure stratégie pour $n = 40$ est de faire 12 parties de 3 jetons et 1 partie de 4 jetons avec comme espérance de C de :
$E(C) = 4^{12} * 5/2^{13} = 10240$. Vous obtiendrez en moyenne un capital C de 10240 euros avec la possibilité d'obtenir un capital maximum de $4 * 3^{12} = 2125764$ euros avec une probabilité de $1/2^{13} = 1/8192$.

Écriture d'un programme pour jouer avec 8 jetons

Pour jouer avec un programme pour 8 jetons, on écrit le programme `jeujetons` de paramètre `n` qui est :
si `n=6` on joue 6 jetons puis 2 jetons,
si `n=5` on joue 5 jetons puis 3 jetons,
si `n=4` on joue 2 fois 4 jetons,
si `n=3` on joue 2 jetons puis 2 fois 3 jetons,
si `n=2` on joue 4 fois 2 jetons
si `n=1` on joue 1 fois 8 jetons
On tape :

```
jeujetons(n):={
  local r,s,j;
    s:=1;
     switch(n){
     case 6:{si rand(2)==1 alors s:=6*s; fsi;
       si rand(2)==1 alors s:=2*s; fsi;break;}
     case 5:{si rand(2)==1 alors s:=5*s; fsi;
       si rand(2)==1 alors s:=3*s; fsi;break;}
     case 4:{pour j de 1 jusque 2 faire
       si rand(2)==1 alors s:=4*s; fsi;
       fpour;break;}
     case 3:{ si rand(2)==1 alors s:=2*s; fsi;
        pour j de 1 jusque 2 faire
          si rand(2)==1 alors s:=3*s; fsi;
        fpour;break;}
     case 2:{pour j de 1 jusque 4 faire
       si rand(2)==1 alors s:=2*s; fsi;
       fpour; break;}
     case 1:{si rand(2)==1 alors s:=8*s; fsi;
       break;}
};
         return s;
  }:;
```

On tape :

```
L1:=jeujetons(1)$(k=1..1000):;
```

```
L2:=jeujetons(2)$(k=1..1000):;
L3:=jeujetons(3)$(k=1..1000):;
L4:=jeujetons(4)$(k=1..1000):;
L5:=jeujetons(5)$(k=1..1000):;
L6:=jeujetons(6)$(k=1..1000):;
evalf(mean([L1]),mean([L2]),mean([L3]),mean([L4]),mean([L5]),mean([L6]))
```

On obtient :

```
4.563, 4.949, 6.105, 6.451, 6.16, 5.357
```

8.10 Un autre jeu

9 pions ayant chacun une face noire et une face blanche sont disposés en carré selon la figure ci-dessous. Le jeu consiste à avoir un carrè unicolore de couleur noire avec comme règle : on a le droit de retourner les pions d'une même ligne, d'une même colonne ou d'une même diagonale. Cela est-il possible ? En combien de coups minimum ?

8.10.1 Le dessin

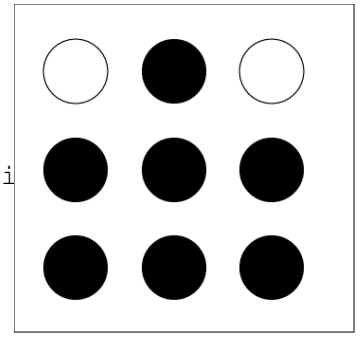
Pour faire le dessin , on tape :

```
pion(C):={
local L,j,k,l,coul;
L:=NULL;
l:=0;
pour j de 2 jusque 0 pas -1 faire
pour k de 0 jusque 2  faire
coul:=C[l];
si coul==0 alors
L:=L,cercle(k+i*j,1/3,affichage=rempli
sinon L:=L,cercle(k+i*j,1/3); fsi;
l:=l+1;
fpour;
fpour;
retourne L;
}:;
```

On tape :

```
pion([1,0,1,0,0,0,0,0,0])
```

On obtient :



Avec `Xcas`, `a%2` désigne un nombre de $\mathbb{Z}/2\mathbb{Z}$ lorsque $a \in \mathbb{Z}$ et toutes les opérations faites avec des nombres de ce type se feront dans $\mathbb{Z}/2\mathbb{Z}$.
Dans la suite, pour avoir des résultats plus lisibles on fera afficher ces nombre sans `%2`, pour cela on utilisera `%0 : (a%2)%0` renvoie `1` si a est impair et `0` sinon. Par exemple `(5+5%2)%0` renvoie `0` et `(5*5%2)%0` renvoie `1`.

Il y a 2 façons de voir les choses :

— **Première façon**

On travaille sur les lignes.
On applatit le carré de façon à le transformer en un vecteur de longueur 9. Ce vecteur V est la configuration de départ et à comme composantes les 0 et les 1 qui sont la valeur des faces des pions de ces cases. Si on retourne par exemple la première ligne la nouvelle configuration sera obtenue par : $V + [1,1,1,0,0,0,0,0,0] \bmod 2$.
On a donc 8 vecteurs $W_1, W_2..W_8$ qui représente les mouvements autorisés. Le problème sera donc résoluble si l'espace vectoriel engendré par les vecteurs $W_1, W_2..W_8$ contient le vecteur de départ. On définit la matrice A ayant 8 lignes ($W_1, W_2..W_8$) et 9 colonnes (qui représentent les 9 pions).
On tape :

```
A:=[[1,1,1,0,0,0,0,0,0],[0,0,0,1,1,1,0,0,0],[0,0,0,0,0,
0,1,1,1],[1,0,0,1,0,0,1,0,0],[0,1,0,0,1,0,0,1,0],[0,0,
1,0,0,1,0,0,1],[1,0,0,0,1,0,0,0,1],[0,0,1,0,1,0,1,0,0]]
```

On obtient :

$$\mathtt{A} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Attention Avec `Xcas` $W_k = A[k-1]$
On cherche le rang de A. On tape :

```
rank(A)%2
```

On obtient : `7`
L'espace vectoriel sur $\mathbb{Z}/2\mathbb{Z}$ engendré par les 8 vecteurs $W_1, W_2..W_8$ représentant les mouvements autorisés est de dimension 7. Pour savoir si par exemple [1,0,1,0,0,0,0,0,0] est dans cet espace vectoriel on tape :
`rank(append(A,[1,0,1,0,0,0,0,0,0])%2)` On obtient : `8`
donc la réponse est : il n'y a pas de solution.
Pour savoir si par exemple [1,0,1,0,0,0,0,0,0] est dans cet espace vectoriel on tape : `rank(append(A,[0,1,0,0,0,0,1,0,1])%2)`
On obtient : `7`
donc la réponse est : il y a une solution.
Il reste à trouver cette solution et à résoudre le problème dans le cas général :

— Quand y-a-t-il une solution ?

On cherche les relations sur les coordonnées du vecteur de départ $V =$

$[a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9]$ pour qu'il y ait une solution.
On tape :
`B:=(rref(A%2)%0)`
On obtient une matrice dans laquelle les vecteurs lignes $V_1, V_2..V_8$ engendrent le même sous-espace vectoriel (de dimension 7) que l'espace engendré par les 8 vecteurs $W_1, W_2..W_8$:

$$\texttt{B} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Puisque V_8 est nul et que le rang de A (resp B) est 7, on en déduit que $V_1..V_7$ sont indépendants. Si $V = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9]$ est le vecteur de départ, il y aura une solution si et seulement si V est une combinaison linéaire modulo 2 des lignes de AB.
Les 7 premières colonnes de cette matrice disent que $a_1, a_2, a_3, a_4, a_5, a_6, a_7$ sont les coefficients de cette combinaison linéaire mais pour qu'il y ait une solution, il faut aussi que cette combinaison linéaire soit valide pour les 2 dernières colonnes. Il y a donc une solution si et seulement si :

$$a_8 = a_2 + a_3 + a_4 + a_6 + a_7 \text{ et } a_9 = a_1 + a_3 + a_7$$

— On cherche la ou les solutions quand ces conditions sont réalisées. Pour cela on décompose le vecteur de départ :
$V = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_2 + a_3 + a_4 + a_6 + a_7, a_1 + a_3 + a_7]\%2$ selon les lignes de A i.e. selon $W_1, W_2..W_8$. Mais attention $W_1, W_2..W_8$ ne sont pas indépendants et donc cette décomposition n'est en général pas unique. On cherche $x_1, x_2, x_3, y_1, y_2, y_3, d_1, d_2$ les coordonnées de V dans cette décomposition, c'est à dire puisqu'on travaille modulo 2 tel que :
$(x_1W_1 + x_2W_2 + x_3W_3 + y_1W_4 + y_2W_5 + y_3W_6 + d_1W_7 + d_2W_8 + [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9])\%2 = 0$.
On définit la matrice AB en mettant $[w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8]$ comme dernière colonne de A. Cela va nous permettre de connaitre les combinaisons faites sur les lignes de A par la réduction de Gauss-Jordan. Puis, on effectue cette réduction sur les 9 premières colonnes de AB en utilisant `rref(AB,9)`.
On tape : `AB:=border(A,[w1,w2,w3,w4,w5,w6,w7,w8])`
On obtient une matrice ayant 8 lignes et 10 colonnes.
On tape pour faire une réduction de Gauss-Jordan seulement les 9 premières colonnes de AB :
`BB:=(rref(AB,9)%2)%0;`

On obtient

$$\texttt{BB} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & w_1+w_3+w_5+w_8 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & w_1+w_3+w_7+w_8 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & w_1+w_5+w_7 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & w_1+w_4+w_5+w_8 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & w_1+w_3+w_5+w_7+w_8 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & w_1+w_5+w_6+w_7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & w_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_1+w_2+w_3+w_4+w_5+w_6 \end{bmatrix}$$

La dernière colonne de `BB` nous donne les opérations faites par la commande `rref(AB,9)%2`.
En particuler on voit :

— $w_1+w_2+w_3+w_4+w_5+w_6 = 0\%2$ i.e. `W1+W2+W3+W4+W5+W6=0%2`.

— Si $V = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9]$ est le vecteur de départ vérifiant $a_8 = a_2 + a_3 + a_4 + a_6 + a_7\%2$ et $a_9 = a_1 + a_3 + a_7\%2$, on a :
$V = a_1V_1 + a_2V_2 + a_3V_3 + a_4V_4 + a_5V_5 + a_6V_6 + a_7V_7 + kV_8\%2$
avec k=0 ou 1.

On tape : `rel:=col(BB,9)`
On obtient :
`[w1+w3+w5+w8,w1+w3+w7+w8,w1+w5+w7,w1+w4+w5+w8,`
`w1+w3+w5+w7+w8,w1+w5+w6+w7,w3,w1+w2+w3+w4+w5+w6]`
On tape : `P:=syst2mat(rel,[w1,w2,w3,w4,w5,w6,w7,w8])`
On obtient la matrice du système avec une colonne de 0 en dernier. En effet `syst2mat` transforme le système $MX + b$ en la natrice M bordée par b qui est nul ici. On veut transformer le système MX en M, donc on enlève cette dernière colonne.
On tape : `P:=delcols(P,8)`
On obtient :

$$\texttt{P} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Attention On a V_8 est le vecteur nul de $(\mathbb{Z}/2\mathbb{Z})^9$ et $V_1..V_7$ sont 7 vecteurs indépendants de $(\mathbb{Z}/2\mathbb{Z})^9$ $V = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9]$ est le vecteur de départ (vecteur de $(\mathbb{Z}/2\mathbb{Z})^9$), avec $a_8 = a_2 + a_3 + a_4 + a_6 + a_7$ et $a_9 = a_1 + a_3 + a_7$ alors les décompositions possibles de V selon les V_k pour $k = 1..8$ sont $V = a_1V_1 + a_2V_2 + a_3V_3 + a_4V_4 + a_5V_5 + a_6V_6 + a_7V_7 + b_8V_8$ avec b_8=0 ou 1. On cherche les décompositions possibles de V selon les W_k pour $k = 1..8$ On tape :
`[a1,a2,a3,a4,a5,a6,a7,b8]*P`
On obtient :

`[b8+a6+a5+a4+a3+a2+a1,b8,b8+a7+a5+a2+a1,b8+a4,`
`b8+a6+a5+a4+a3+a1,b8+a6,a6+a5+a3+a2,a5+a4+a2+a1]`
Donc on a :
$V = a_1V_1 + a_2V_2 + a_3V_3 + a_4V_4 + a_5V_5 + a_6V_6 + a_7V_7 + b_8V_8$ et
$V = (b_8 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)W_1 + b_8W_2 + (b_8 + a_7 + a_5 + a_2 + a_1)W_3 + (b_8 + a_4)W_4 +$
$(b_8 + a_6 + a_5 + a_4 + a_3 + a_1)W_5 + (b_8 + a_6)W_6 + (a_6 + a_5 + a_3 + a_2)W_7 + (a_5 + a_4 + a_2 + a_1)W_8$

— On peut aussi chercher les décompositions possibles de V :
$V = [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_2 + a_3 + a_4 + a_6 + a_7, a_1 + a_3 + a_7]\%2$.
Pour cela on décompose V selon les lignes $W_1..W_8$ de $A\%2$.
On cherche $x_1, x_2, x_3, y_1, y_2, y_3, d_1, d_2$ les coordonnées de V dans cette décomposition, c'est à dire puisqu'on travaille modulo 2 tel que :
$(x_1 * W_1 + x_2 * W_2 + x_3 * W_3 + y_1 * W_4 + y_2 * W_5 + y_3 * W_6 + d_1 * W_7 + d_2 * W_8 + [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9])\%2 = 0$.
On tape :
`W:=[x1,x2,x3,y1,y2,y3,d1,d2]*A+[a1,a2,a3,a4,`
`a5,a6,a7,a2+a3+a4+a6+a7,a1+a3+a7])%2`
`linsolve(W,[x1,x2,x3,y1,y2,y3,d1,d2])%0`
On obtient les valeurs (%2 sous-entendu) de `[x1,x2,x3,y1,y2,y3,d1,d2]`
c'est à dire les mouvements à effectuer :
`[a1+a2+a3+a4+a5+y3,a6+y3,a1+a2+a5+a6+a7+y3,a4+a6+y3,`
`a1+a3+a4+a5+y3,y3,a2+a3+a5+a6,a5+a4+a2+a1].`
Remarque :
Dans ce résultat `y3` vaut 0 ou 1 alors que dans le résultat précédent :
$V = (b_8 + a_6 + a_5 + a_4 + a_3 + a_2 + a_1)W_1 + b_8W_2 + (b_8 + a_7 + a_5 + a_2 + a_1)W_3 + (b_8 + a_4)W_4+$
$(b_8 + a_6 + a_5 + a_4 + a_3 + a_1)W_5 + (b_8 + a_6)W_6 + (a_6 + a_5 + a_3 + a_2)W_7 + (a_5 + a_4 + a_2 + a_1)W_8$
c'est b_8 qui valait 0 ou 1.
Les 2 formules sont identiques puisque :
$b_8 = y_3\%2$ si $a_6 = 0$ et
$b_8 = 1 + y_3\%2$ si $a_6 = 1$.

— **Deuxième façon**
On travaille sur les colonnes.
On donne des noms de variables aux lignes, colonnes et diagonales : x_1, x_2, x_3 pour les 3 lignes, y_1, y_2, y_3 pour les 3 colonnes, d_1, d_2 pour les 2 diagonales.
Au départ ces variables valent 0 et chaque retournement va changer la valeur de la variable correspondante qui passera de 0 à 1 ou de 1 à 0.
Si par exemple [1,0,1,0,0,0,0,0,0] est la configuration de départ, chaque case a pour valeur :

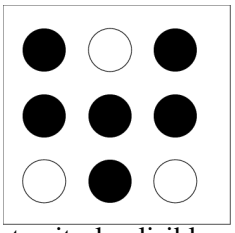
(x1+y1+d1+1)%2	(x1+y2)%2	(x1+y3+d2+1)%2
(x2+y1)%2	(x2+y2+d1+d2)%2	(x2+y3)%2
(x3+y1+d2)%2	(x3+y2)%2	(x3+y3+d1)%2

Pour résoudre le problème il suffit de résoudre le système linaire dans $\mathbb{Z}/2\mathbb{Z}$ pour que toutes les cases deviennent noires i.e. soient de valeurs 0.
On tape :
`linsolve([x1+y1+d1+1,x1+y2,x1+y3+d2+1,x2+y1,x2+y2+d1+d2,x2+y3,`

```
x3+y1+d2,x3+y2,x3+y3+d1]%2,[x1,x2,x3,y1,y2,y3,d1,d2])
```
On obtient : `[ ]`
Donc le problème n'est pas résoluble.

Une disposition où cela est possible

On tape :
`pion([0,1,0,0,0,0,1,0,1])`
On obtient :



On tape (on rajoute `%0` pour que le résultat soit plus lisible car il ne sera pas affiché avec `%2` :
```
linsolve([x1+y1+d1,x1+y2+1,x1+y3+d2,x2+y1,x2+y2+d1+d2,x2+y3,
x3+y1+d2+1,x3+y2,x3+y3+d1+1]%2,[x1,x2,x3,y1,y2,y3,d1,d2])%0
```
On obtient (`%2` est ici sous-entendu) : `[y3+1,y3,y3,y3,y3,y3,1,1]`
Puisque l'ordre des variables est `[x1,x2,x3,y1,y2,y3,d1,d2]`, cela veut dire que :
si on choisit `y3=0` la solution est `[1,0,0,0,0,0,1,1]` donc `x1=1`, `d1=1` et `d2=1` c'est à dire :
il faut retourner les 2 diagomales et la première ligne ou
si on choisit `y3=1` la solution est `[0,1,1,1,1,1,1,1]` donc `x2=1`, `x3=1`, `c1=1`, `c2=1`, `c3=1`, `d1=1` et `d2=1` donc il faut retourner les 2 diagomales et les 2 dernières lignes et les 3 colonnes.
On peut aussi taper en notant `a1,a2,a3,a4,a5,a6,a7,a8,a9` les couleurs par ligne des pions du carré :
```
M:=syst2mat([x1+y1+d1+a1,x1+y2+a2,x1+y3+d2+a3,x2+y1+a4,
x2+y2+d1+d2+a5,x2+y3+a6,x3+y1+d2+a7,x3+y2+a8,x3+y3+d1+a9],
[x1,x2,x3,y1,y2,y3,d1,d2])
```
On obtient :
```
[[1,0,0,1,0,0,1,0,a1],[1,0,0,0,1,0,0,0,a2],[1,0,0,0,0,1,0,1,a3],
[0,1,0,1,0,0,0,0,a4],[0,1,0,0,1,0,1,1,a5],[0,1,0,0,0,1,0,0,a6],
[0,0,1,1,0,0,0,1,a7],[0,0,1,0,1,0,0,0,a8],[0,0,1,0,0,1,1,0,a9]]
```
Pour comparer avec la première méthode, on remarquera que `M` est :
`border(tran(A),[a1,a2,a3,a4,a5,a6,a7,a8,a9])` où A est la matrice de la première méthode.
Pour donner l'emplacement des couleurs, on tape par exemple :
```
(a1,a2,a3,a4,a5,a6,a7,a8,a9):=(1,0,1,0,0,0,0,1,0)
linsolve(M*[x1,x2,x3,y1,y2,y3,d1,d2,1]%2,
[x1,x2,x3,y1,y2,y3,d1,d2])%0
```
On obtient (`%2` est ici sous-entendu) :
`[y3,y3,y3+1,y3,y3,y3,1,1]`
Dans ce cas si on choisit $y3 = 0$, il faut retourner les 2 diagonales et la troisième ligne et si on choisit $y3 = 1$, il faut retourner les 2 diagonales et les lignes 1 et 2 ainsi que les colonnes 1,2,3.
On tape pour donner l'emplacement des couleurs si on a :
```
(a1,a2,a3,a4,a5,a6,a7,a8,a9):=(1,0,1,0,1,0,0,0,0)
linsolve(M*[x1,x2,x3,y1,y2,y3,d1,d2,1]%2,
```

`[x1,x2,x3,y1,y2,y3,d1,d2])`
On obtient :
`[ ]`
Remarque si les coins ne comporte qu'une case noire ou qu'une case blanche, on est sûr que le problème n'a pas de solution car chaque retournement retourne simultanément deux coins donc la parité du nombre de coins noirs est constante d'où une condition d'impossibilité est :
$a1 + a3 + a7 + a9 \neq 0 \bmod 2$.
Résolution dans le cas général
On doit donc savoir pour quelles valeurs de `a1,a2,a3,a4,a5,a6,a7,a8,a9` le système d'inconnues `x1,x2,x3,y1,y2,y3,d1,d2` a des solutions :

$$\begin{cases} x_1 + y_1 + d_1 + a_1 & = 0 \\ x_1 + y_2 + a_2 & = 0 \\ x_1 + y_3 + d_2 + a_3 & = 0 \\ x_2 + y_1 + a_4 & = 0 \\ x_2 + y_2 + d_1 + d_2 + a_5 & = 0 \\ x_2 + y_3 + a_6 & = 0 \\ x_3 + y_1 + d_2 + a_7 & = 0 \\ x_3 + y_2 + a_8 & = 0 \\ x_3 + y_3 + d_1 + a_9 & = 0 \end{cases}$$

On tape :
`purge(a1,a2,a3,a4,a5,a6,a7,a8,a9);`
On a déjà définit M :

$$\mathtt{M} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & a_1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & a_2 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & a_3 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & a_4 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & a_5 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & a_6 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & a_7 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & a_8 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & a_9 \end{bmatrix}$$

On fait une réduction de Gauss-Jordan sur les 8 premières colonnes de M, on tape :
`RM:=rref(M%2,8)%0`
On obtient :

$$\mathtt{RM} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & a_1 + a_2 + a_3 + a_4 + a_5 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & a_6 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & a_1 + a_2 + a_5 + a_6 + a_7 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & a_4 + a_6 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & a_1 + a_3 + a_4 + a_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & a_2 + a_3 + a_5 + a_6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & a_1 + a_2 + a_4 + a_5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_2 + a_3 + a_4 + a_6 + a_7 + a_8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_1 + a_3 + a_7 + a_9 \end{bmatrix}$$

On lit sur les 7 premières lignes de RM :
$x_1 + y_3 = a_1 + a_2 + a_3 + a_4 + a_5 \bmod 2$
$x_2 + y_3 = a_6 \bmod 2$
$x_3 + y_3 = a_1 + a_2 + a_5 + a_6 + a_7 \bmod 2$
$y_1 + y_3 = a_4 + a_6 \bmod 2$
$y_2 + y_3 = a_1 + a_3 + a_4 + a_5 \bmod 2$
$d_1 = a_2 + a_3 + a_5 + a_6 \bmod 2$
$d_2 = a_1 + a_2 + a_4 + a_5 \bmod 2$
et sur les 2 dernières lignes de RM :
$a_2 + a_3 + a_4 + a_6 + a_7 + a_8 = 0 \bmod 2$ et
$a_1 + a_3 + a_7 + a_9 = 0 \bmod 2$
On tape les conditions trouvées :
`a8:=((a2+a3+a4+a6+a7)%2)%0;a9:=((a1+a3+a7)%2)%0`
On peut ussi utiliser `linsolve` comme dans la première méthode.
On tape pour enlever la dernière colonne de M :
`N:=delcols(M,8)`
On a alors $N = tran(A)$ où A est la matrice de la première méthode.
On tape : `rank(N)`
On obtient : `7`
On résout dans $\mathbb{Z}/2\mathbb{Z}$ le système linéaire :
$N * [x_1, x_2, x_3, y_1, y_2, y_3, d_1, d_2] + [a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9])\%2 = 0$
On tape :
```
linsolve((N*[x1,x2,x3,y1,y2,y3,d1,d2]+[a1,a2,a3,a4,a5,a6,a7,
a2+a3+a4+a6+a7,a1+a3+a7]),2, [x1,x2,x3,y1,y2,y3,d1,d2])%0
```
On obtient les valeurs (%2 sous-entendu) de `[x1,x2,x3,y1,y2,y3,d1,d2]`
c'est à dire les mouvements à effectuer :
```
([a1+a2+a3+a4+a5+y3,a6+y3,a1+a2+a5+a6+a7+y3,a4+a6+y3,
a1+a3+a4+a5+y3,y3,a2+a3+a5+a6,a5+a4+a2+a1])%2
```
avec `y3` égal à 0 ou 1.

Les programmes qui permettent de jouer

```
//affiche le jeu de configuration C
//p est la position du pion en bas a gauche
pion(C,p):={
local L,j,k,l,coul,x1,x2,x3,y1,y2,y3,d1,d2,y3,sol0,sol1;
L:=NULL;l:=0;
pour j de 2 jusque 0 pas -1 faire
 pour k de 0 jusque 2  faire
  coul:=C[l];
  si coul==0 alors
    L:=L,affichage(cercle(p+k+i*j,1/3),rempli);
  sinon
    L:=L,cercle(p+k+i*j,1/3);
  fsi;
  l:=l+1;
 fpour;
fpour;
retourne L,carre(p-0.5-i*0.5,p+2.5-i*0.5);
```

```
};;
pion_sol(C):={
local x1,x2,x3,y1,y2,y3,d1,d2,sol0,sol1;
si (C[0]+C[2]+C[6]+C[8])%2==0 and
   (C[1]+C[3]+C[5]+C[7]+C[0]+C[8])%2==0 alors
  x1:=(C[1]+C[2]+C[3]+C[4]+C[0])%2;
  x2:=(C[5])%2;
  x3:=(C[1]+C[0]+C[5]+C[6]+C[4])%2;
  y1:=(C[5]+C[3])%2;
  y2:=(C[0]+C[3]+C[4]+C[2])%2;
  y3:=0%2;
  d1:=(C[2]+C[1]+C[5]+C[4])%2;
  d2:=(C[1]+C[0]+C[4]+C[3])%2;
  sol0:=[x1,x2,x3,y1,y2,y3,d1,d2];
  y3:=1 %2;
  x1,x2,x3,y1,y3:=x1+y3,x2+y3,x3+y3,y1+y3,y2+y3;
  sol1:=[x1,x2,x3,y1,y2,y3,d1,d2];
  si count_eq(1%2,sol0)<count_eq(1%2,sol1) alors
    retourne sol0%0;
  sinon
    retourne sol1%0 ;
  fsi;
sinon
  retourne [];
fsi;
};;
operations(C):={
local L,j,sol;
sol:="";
L:=pion_sol(C);
si L==[] alors retourne "Pas de solution" fsi;
pour j de 0 jusque 2 faire
  si L[j]==1 alors sol:=sol+"ligne "+(j+1)+"  ";fsi;
fpour;
pour j de 3 jusque 5 faire
  si L[j]==1 alors sol:=sol+"colonne "+(j-2)+"  ";fsi;
fpour;
si L[6]==1 alors sol:=sol+"diagonale 1  ";fsi;
si L[7]==1 alors sol:=sol+"diagonale 2  ";fsi;
retourne "Retournez "+sol;
};;
//affiche le jeu de configuration C puis
//sa config selon les mvts m=1,2..ou 8
//m est entre par l'utilisateur au plus 5 fois
jeu(C):={
local A,M,m,p,L,k,s,s1;
A:=[[1,1,1,0,0,0,0,0,0],[0,0,0,1,1,1,0,0,0],
[0,0,0,0,0,0,1,1,1],[1,0,0,1,0,0,1,0,0],[0,1,0,0,1,0,0,1,0],
```

```
[0,0,1,0,0,1,0,0,1],[1,0,0,0,1,0,0,0,1],[0,0,1,0,1,0,1,0,0]];
ClrGraph();
gl_x=-1..18;
gl_y=-3..5;
axes=0;
DispG;
L:=pion(C,0);
s1:=operations(C);
k:=1;
input("m=(1,..8)",m);
p:=0;
tantque m>0 et m<9  faire
  M:=A[m-1];
  p:=p+3;
  C:=(C+M) %2;
  L:=L,pion(C,p)
  k:=k+1;
  si k==6 or C==[0,0,0,0,0,0,0,0,0] alors break; fsi;
  input("m=(1,..8)",m);
ftantque;
si operations(C)=="Retournez " alors
  s:="Bravo !"
sinon
  s:=operations(C)
fsi;
print("sol min :"+s1);
L:=L,legende(-i,s);
retourne L;
}:;
//pion_op(C) renvoie la liste des operations
pion_op(C):={
ClrGraph();
DispG;
pion(C,0);
return operations(C)
}:;
//affiche le jeu de configuration C puis
//sa config selon le mvt m=1,2..ou 8
//m est entre par l'utilisateur
pion_modif(C):={
local M,m,p,L;
ClrGraph();
L:=pion(C,0);
m:=1;
p:=0;
tantque  m>0 et m<9 faire
 input("m=(1,..8)",m);
 M:=[[1,1,1,0,0,0,0,0,0],[0,0,0,1,1,1,0,0,0],
```

```
 [0,0,0,0,0,0,1,1,1],[1,0,0,1,0,0,1,0,0],[0,1,0,0,1,0,0,1,0],
 [0,0,1,0,0,1,0,0,1],[1,0,0,0,1,0,0,0,1],[0,0,1,0,1,0,1,0,0]];
 p:=p+3;
 C:=(C+M[m-1])%2;
 L:=L,pion(C,p)
ftantque;
retourne L;
}:;
```

On tape :
`pion_op([0,1,0,0,0,0,0,1,0])`
On obtient :
`"retournez ligne 1 ligne 3 diagonale 1 diagonale 2 "` et l'ouverture de l'écran `DispG` sur lequel est dessiné le problème à résoudre.

On peut montrer avec un programme que si le problème est possible, on peut le résoudre en faisant au plus 5 opérations (si il n'y a qu'un pion blanc au centre 5 opérations sont nécessaires.
On tape :

```
combien():={
local a1,a2,a3,a4,a5,a6,a7,nb,rep,y3,c0,c1;
nb:=0;
pour a1 de 0 jusque 1 faire
pour a2 de 0 jusque 1 faire
pour a3 de 0 jusque 1 faire
  pour a4 de 0 jusque 1 faire
  pour a5 de 0 jusque 1 faire
  pour a6 de 0 jusque 1 faire
    pour a7 de 0 jusque 1 faire
    y3:=0;
    rep:=([a1+a2+a3+a4+a5+y3,a6+y3,a1+a2+a5+a6+a7+y3,
         a4+a6+y3,a1+a3+a4+a5+y3,y3,
        a2+a3+a5+a6,a5+a4+a2+a1]%2)%0;
    c0:=count_eq(1,rep);
    y3:=1;
    rep:=([a1+a2+a3+a4+a5+y3,a6+y3,a1+a2+a5+a6+a7+y3,
        a4+a6+y3,a1+a3+a4+a5+y3,y3,
        a2+a3+a5+a6,a5+a4+a2+a1]%2)%0;
    c1:=count_eq(1,rep);nb:=max(nb,min(c0,c1));
    fpour;
  fpour;
  fpour;
  fpour;
fpour;
fpour;
fpour;
return nb;
}:;
```

On tape `combien()` et on obtient `5`

8.11 Analyse du jeu du dobble

Soit pour $n \in \mathbb{N}$, l'ensemble $E = \{1, 2, ...n\}$.
On veut construire un jeu comportant n cartes C_j pour $j = 1..n$ de façon que chaque carte C_j ait comme valeur un sous ensemble de E :
$C_j \subset E$,
Card(C_j) $= m$ et
Card($C_j \cap C_k$) $= 1$ pour tout j et tout $k \neq j$.

8.11.1 Le point de vue naïf

Comment choisir n ? Écrire un programme qui teste que dans un jeu on a Card($C_j \cap C_k$) $= 1$
et trouver un algorithme qui renvoie la valeur des n cartes C_j $j = 1..n$.

Par exemple, en écrivant la valeur des cartes sous la forme d'une liste :
si $n = 3$ on a $m = 2$ et les 3 cartes du jeu J_2 sont : $[1, 2], [1, 3], [2, 3]$
et le jeu J_2 est $[[1, 2], [1, 3], [2, 3]]$
si $n = 4, 5, 6$ il n'y a pas de solution.
si $n = 7$ on a $p = 3$ et les 7 cartes du jeu J_3 sont : $[1, 2, 3], [1, 4, 5], [1, 6, 7], [2, 4, 6], [2, 5, 7], (3, 4, 7], [3, 5$
et le jeu J_3 est $[[1, 2, 3], [1, 4, 5], [1, 6, 7], [2, 4, 6], [2, 5, 7], (3, 4, 7], [3, 5, 6]]$
Sur chacune des n cartes il y a m nombres : on a donc en tout $n * m$ nombres et pour des raisons de symétrie, chaque nombre figure m fois donc :
il y a m cartes qui contiennent 1,
il y a m cartes qui contiennent 2,
....
il y a m cartes qui contiennent n.
L'ensemble des éléments des m cartes qui contiennent 1 est E donc :
puisque l'ensemble des éléments des m cartes qui contiennent 1 est constitué de 1 et de $m - 1$ éléments différents de 1, on a donc en tout $m(m - 1)$ éléments différents de 1, donc en tout $m(m - 1) + 1$ éléments qui constituent E donc :

$$n = m(m - 1) + 1$$

Ainsi en écrivant la valeur des cartes sous la forme d'une liste pour :
$m = 2$ on a $n = 3$ et
$J_2 := [[1, 2], [1, 3], [2, 3]]$
$m = 3$ on a $n = 7$ et
$J_3 := [[1, 2, 3], [1, 4, 5], [1, 6, 7], [2, 4, 6], [2, 5, 7], [3, 4, 7], [3, 5, 6]]$
$m = 4$ on a $n = 13$ et
$J_4 := [[1, 2, 3, 4], [1, 5, 6, 7], [1, 8, 9, 10], [1, 11, 12, 13],$
$[2, 5, 8, 11], [2, 6, 9, 12], [2, 7, 10, 13], [3, 5, 9, 13],$
$[3, 6, 10, 11], [3, 7, 8, 12], [4, 5, 10, 12], [4, 6, 8, 13], [4, 7, 9, 11]]$
$m = 5$ on a $n = 21$ et
$J5 := [[1, 2, 3, 4, 5],$
$[1, 6, 7, 8, 9], [1, 10, 11, 12, 13], [1, 14, 15, 16, 17], [1, 18, 19, 20, 21],$
$[2, 6, 10, 14, 18], [2, 7, 11, 15, 19], [2, 8, 12, 16, 20], [2, 9, 13, 17, 21],$
$[3, 6, 12, 17, 19], [3, 7, 13, 16, 18], [3, 8, 10, 15, 21], [3, 9, 11, 14, 20],$

$[4,6,13,15,20],[4,7,12,14,21],[4,8,11,17,18],[4,9,10,16,19],$
$[5,6,11,16,21],[5,7,10,17,20],[5,8,13,14,19],[5,9,12,15,18]]$
$m=6$ on a $n=31$ et
$J_6 := [[1,2,3,4,5,6],$
$[1,7,8,9,10,11],[1,12,13,14,15,16],[1,17,18,19,20,21],[1,22,23,24,25,26],[1,27,28,29,30,31],$
$[2,7,12,17,22,27],[2,8,13,18,23,28],[2,9,14,19,24,29],[2,10,15,20,25,30],[2,11,16,21,26,31],$
$[3,7,13,19,25,31],[3,8,14,20,26,27],[3,9,15,21,22,28],[3,10,16,17,23,29],[3,11,12,18,24,30],$
$[4,7,14,21,23,30],[4,8,15,17,24,31],[4,9,16,18,25,27],[4,10,12,19,26,28],[4,11,13,20,22,29],$
$[5,7,16,20,24,28],[5,8,12,21,25,29],[5,9,13,17,26,30],[5,10,14,18,22,31],[5,11,15,19,23,27],$
$[6,7,15,18,26,29],[6,8,16,19,22,30],[6,9,12,20,23,31],[6,10,13,21,24,27],[6,11,14,17,25,28]]$

Écrivons un programme qui teste que dans un jeu on a :
Card($C_j \cap C_k$) $= 1$ pour tout j et tout $k \neq j$.
On tape :

```
testcarte(L):={
local j,k,s,S;
s:=size(L);
S:=set[op(L[j])] $ (j=0..s-1);
pour j de 0 jusque s-2 faire
  pour k de j+1 jusque s-1 faire
    si size(S[j] intersect S[k])!=1 alors
      retourne [j,k];
    fsi;
  fpour;
fpour;
retourne 1;
}:;
```

On tape :

```
J6:=[[1,2,3,4,5,6],[1,7,8,9,10,11],[1,12,13,14,15,16],[1,17,18,19,20,21],
[1,22,23,24,25,26],[1,27,28,29,30,31],[2,7,12,17,22,27],[2,8,13,18,23,28],
[2,9,14,19,24,29],[2,10,15,20,25,30],[2,11,16,21,26,31],[3,7,13,19,25,31],
[3,8,14,20,26,27],[3,9,15,21,22,28],[3,10,16,17,23,29],[3,11,12,18,24,30],
[4,7,14,21,23,30],[4,8,15,17,24,31],[4,9,16,18,25,27],[4,10,12,19,26,28],
[4,11,13,20,22,29],[5,7,16,20,24,28],[5,8,12,21,25,29],[5,9,13,17,26,30],
[5,10,14,18,22,31],[5,11,15,19,23,27],[6,7,15,18,26,29],[6,8,16,19,22,30],
[6,9,12,20,23,31],[6,10,13,21,24,27],[6,11,14,17,25,28]];
testcarte(J6)
```

On obtient :

```
1
```

Mais pour l'instant je n'ai pas trouvé d'algorithme...
Par exemple pour $p=7$ et $n=43$? ? ? ? ?
Pour $p=6$ et $n=31$, les cartes contenant C_1, C_2, C_3 sont obtenues systématiquement puis je fais différents raisonnements ce qui donne le programme qui suit.
On tape :

```
dobbleprog(m):={
local n,T,L1,L2,j,Rep,LL2,L,l;
n:=m*(m-1)+1;
```

```
T:=[k$(k=1..m)];
L1:=T;
pour j:=1 jusque m-1 faire
  T:=[T[0],(T[k]+m-1)$(k=1..m-1)];
  L1:=L1,T;
fpour;
Rep:=L1;
L1:=[L1];
L2:=[col(L1,k)$(k=1..m-1)];
pour j:=1 jusque m-2 faire
  L2[j,0]:=2;
fpour;
L2:=op(L2);
Rep:=Rep,L2;
LL2:=NULL;
pour j de 1 jusque m-1 faire
  LL2:=LL2,L2;
  fpour;
  LL2:=[LL2];
pour l de 3 jusque m faire
  L:=NULL;
  pour j de 0 jusque m-2 faire
    L:=L,[l,LL2[(l-2)*(k-1)+j,k]$(k=1..m-1)];
  fpour;
Rep:=Rep,L;
fpour;
retourne [Rep];
}:;
```

Puis on teste en tapant :

```
D:=[dobbleprog(m)$(m=2..32)]:;
testcarte(D[k])$(k=0..30)
```

On obtient :

```
1,1,1,[5,13],1,[7,19],1,[9,25],[10,37],[11,31],1,
[13,37],1,[15,43],[16,61],[17,49],1,[19,55],1,[21,61]
[22,85],[23,67],1,[25,73],[26,151],[27,79],[28,109],
[29,85],1,[31,91],1
```

ce qui veut dire que le programme renvoie une solution pour :
$m = 2, 3, 4, 6, 8, 12, 14, 18, 20, 24, 30, 32$ c'est à dire pour :
$n = 3, 7, 13, 31, 57, 133, 183, 307, 381, 553, 871, 993$ $p = m-1 = 1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31$

8.11.2 Avec un peu de Mathématiques

Soit le corps $K = \mathbb{Z}/3\mathbb{Z}$. K a comme éléments 0,1,2.
Considérons le plan projectif $P = K^{3*}/K^*$.
P a 13 points qui sont :

— 4 points à l'infini :
(1,0,0), (0,1,0), (1,1,0), (1,2,0)

— 9 points à distance finie :
(0,0,1), (1,0,1), (2,0,1)
(0,1,1), (1,1,1), (2,1,1)
(0,2,1), (1,2,1), (2,2,1)

P a 13 droites :

— la droite de l'infini qui contient les 4 points à l'infini,
— 3 droites paralléles de direction (1,0,0) contenant chacune 4 points : le point à l'infini (1,0,0) et 3 points à distance finie.
On a :
la droite passant par les 3 points à distance finie $(0,0,1)+k(1,0,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,0,1), (1,0,1),(2,0,1).
la droite passant par les 3 points à distance finie $(0,1,1)+k(1,0,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,1,1), (1,1,1),(2,1,1).
la droite passant par les 3 points à distance finie $(0,2,1)+k(1,0,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,2,1), (1,2,1),(2,2,1).
— 3 droites paralléles de direction (0,1,0) contenant chacune 4 points : le point à l'infini (0,1,0) et 3 points à distance finie.
On a :
la droite passant par les 3 points à distance finie $(0,0,1)+k(0,1,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,0,1), (0,1,1),(0,2,1).
la droite passant par les 3 points à distance finie $(1,0,1)+k(0,1,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(1,0,1), (1,1,1),(1,2,1) .
la droite passant par les 3 points à distance finie $(2,0,1)+k(0,1,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(2,0,1), (2,1,1),(2,2,1).
— 3 droites paralléles de direction (1,1,0) contenant chacune 4 points : le point à l'infini (1,1,0) et 3 points à distance finie.
On a :
la droite passant par les 3 points à distance finie $(0,0,1)+k(1,1,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,0,1), (1,1,1),(2,2,1).
la droite passant par les 3 points à distance finie $(0,1,1)+k(1,1,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,1,1), (1,2,1),(2,0,1).
la droite passant par les 3 points à distance finie $(0,2,1)+k(1,1,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,2,1), (1,0,1),(2,1,1).
— 3 droites paralléles de direction (1,2,0) contenant chacune 4 points : le point à l'infini (1,2,0) et 3 points à distance finie.
On a :
la droite passant par les 3 points à distance finie $(0,0,1)+k(1,2,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,0,1), (1,2,1),(2,1,1).

la droite passant par les 3 points à distance finie $(0,1,1)+k(1,2,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,1,1), (1,0,1),(2,2,1).
la droite passant par les 3 points à distance finie $(0,2,1)+k(1,2,0)$ pour $k \in K$ i.e. $k=0,1,2$:
(0,2,1), (1,1,1),(2,0,1).

Chacune de ces 13 droites sont concourantes et correspondent à 13 cartes contenant chacune 4 objets.
On peut généraliser facilement pour lorsque K est le corps $K = \mathbb{Z}/p\mathbb{Z}$ pour p premier : le plan projectif $P = K^{3*}/K^*$ a $p^2+p+1 = p(p+1)+1$ points :
p^2 points à distance finie et $p+1$ points l'infini et dans ce plan on a $p+1$ droites contenant chacune $p+1$ points : le m du programme naïf est donc $m = p+1$ avec p premier.
En effet :
Pour connaitre les valeurs de $p = m-1$ pour lesquelles le programme naïf renvoie une solution, on tape :

```
lesuns(D):={
local k,M,s;
M:=NULL;
s:=size(D)-1;
pour k de 0 jusque s faire
si testcarte(D[k])==1 alors M:=M,k+1; fsi;
fpour
retourne M;
}:;
```

On tape :
`D:=[dobbleprog(m)$(m=2..50)]`
On obtient :
`Done`
On tape :
`lesuns(D)`
On obtient la liste des 16 premiers nombres premiers (Temps mis pour l'évaluation : 569.29) :
`(1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47)`
On tape :
`ithprime(k)$(k=0..15)`
On obtient :
`(1,2,3,5,7,11,13,17,19,23,29,31,37,41,43,47)`

8.11.3 Programme lorsque K est un corps fini

Puisque le cardinal d'un corps fini K est q^n avec q premier et $n \in \mathbb{N}$, on peut trouver une solution lorsque $p = m-1 = q^n$ avec q premier et $n \in \mathbb{N}$.
On peut construire un corps fini K de cardinal q^n avec q premier et $n \in \mathbb{N}$ en prenant :

$K = \mathbb{Z}/q\mathbb{Z}[x] \bmod (P[x] \bmod q)$ où $P[x] \bmod q$ est un polynôme irréductible de $\mathbb{Z}/q\mathbb{Z}[x]$ de degré n.
Par exemple :
Pour $m-1 = p = 4 = 2^2$, on a $K4 = \mathbb{Z}/2\mathbb{Z}[x] \bmod (x^2 + x + 1 \bmod 2)$
Pour $m-1 = p = 8 = 2^3$, on a $K8 = \mathbb{Z}/2\mathbb{Z}[x] \bmod (x^3 + x + 1 \bmod 2)$
Pour ,$m-1 = p = 9 = 3^2$, on a $K9 = \mathbb{Z}/3\mathbb{Z}[x] \bmod (x^2 + 1 \bmod 3)$
etc...

Les tables d'addition et de multiplication

Soit P le polynôme à coefficients dans $\mathbb{Z}/q\mathbb{Z}$ avec q premier tel que $\mathbb{Z}/q\mathbb{Z}[x] \bmod P$ soit un corps.
On note K la liste des éléments de $\mathbb{Z}/q\mathbb{Z}[x] \bmod P$. On représente les éléments de $\mathbb{Z}/q\mathbb{Z}[x] \bmod P$ par l'indice qu'ils ont dans K.
On a donc besoin de la commande indice qui renvoie l'indice d'un élément dans une liste et "erreur" lorsque l"élément n'est pas dans la liste.
On tape :

```
// renvoie l'indice de a dans la liste L sinon renvoie  erreur
indice(a,L):={
indice(a,L):={
local k;
k:=member(normal(a),L);
 si  k!=0 alors
   retourne k-1;
 fsi;
 retourne "erreur";
}:;
```

Si $K[a] \in \mathbb{Z}/q\mathbb{Z}[x] \bmod P$ et $K[b] \in \mathbb{Z}/q\mathbb{Z}[x] \bmod P$ on a :
`addi(a,b,K,P)` est égal à l'indice de la somme de $K[a] + K[b]$ dans K et
`prod(a,b,K,P)` est égal à l'indice du produit $K[a] * K[b]$ dans K.
Par exemple si $q = 2$ et $P = (x^2 + x + 1) \bmod 2$ on a :
$K = [0\%2, 1\%2, x * 1\%2, 1\%2 + x * 1\%2]$
`addi(0,j)=j` pour $j = 0..3$
`addi(1,2)=3` car $(1\%2) + (x * 1\%2) = 1\%2 + x * 1\%2$
`prod(3,2)=1` car $(1\%2 + x * 1\%2) * (x * 1\%2) = 1\%2 \bmod (x^2 + x + 1)\%2$
On tape :

```
//addi(a,b,K,P) renvoie l'indice dans la liste K de la somme (K[a]+K[b])mod (
addi(a,b,K,P):={
   retourne indice((K[a]+K[b]) mod (P),K);
}:;
//prod(a,b,K,P)renvoie l'indice dans la liste K du produit (K[a]*K[b])mod (P)
prod(a,b,K,P):={
   retourne indice((K[a]*K[b])mod (P),K);
}:;
```

Lorsque L est une liste d'indice de K, on rajoute la commande `prodl(a,L,K,P))` qui renvoie la liste d'éléments $a * L[j]$ pour $j = 0..\texttt{size}(L) - 1$.
Lorsque L_1 et L_2 sont 2 listes d'indice de K de même longueur, on rajoute la commande `addill(L1,L2,K,P)` qui renvoie la liste d'éléments $L_1[j] + L_2[j]$ pour $j = 0..\texttt{size}(L_1) - 1$.
On tape :

```
//pour L une liste d'indice de taille s, prodl(a,L,K,P) renvoie
//la liste des indices dans la liste K du produit (K[a]*K[L[k]])mod (
prodl(a,L,K,P):={
  local s;
  s:=size(L)-1;
  retourne [prod(a,L[k],K,P)$(k=0..s)];
  }:;
//pour L1 et L2 2 listes d'indice de taille s addill(L1,L2,K,P) renvo
//la liste des indices dans la liste K de la somme (K[L1[k]]+K[L2[k]]
addill(L1,L2,K,P):={
  local s1,s2;
  s1:=size(L1)-1;
  s2:=size(L2)-1;
  si s1==s2 alors
  retourne [addi(L1[k],L2[k],K,P)$(k=0..s1)];
  fsi;
retourne "erreur";
}:;
```

La répartition pour $p = 4$

On définit le corps $L4$ de cardinal 4 et le polygône P4.
Puis on écrit la fonction de repartition : on remarquera que pour que les indices des éléments du plan projectif PP commencent à 1 on a rajouté l'élément $[0, 0, 0]$ au début de la séquence constituée par les éléments de PP.

```
K4():={
  local L4,j,k,s,x;
  purge(x);
  L4:=NULL;
  pour k de 0 jusque 1 faire
    pour j de 0 jusque 1 faire
      s:=x*k+j;
      L4:=L4,normal(s %2) ;
    fpour;
  fpour;
  return [L4];
    }:;
P4:=(x^2+x+1)%2:;
L4:=K4():;
repart4(L4,P4):={
```

```
  local k,j,L,Le,n,Rep,PP;
  PP:=[0,0,0],[1,0,0],[0,1,0],([1,k,0])$(k=1..3);
  pour k de 0 jusque 3 faire
    pour j de 0 jusque 3 faire
      PP:=PP,[j,k,1];
    fpour;
  fpour;
  PP:=[PP];
  Rep:=[1,(k+j*4)$(k=2..5)]$(j=0..4);
  pour j de 2 jusque 5 faire
    L:=NULL;
    pour k de 6 jusque 9 faire
      Le:=j;
      pour n de 0 jusque 3 faire
        Le:=Le,indice(addill(PP[k],prodl(n,PP[j],L4,P4),L4,P4),PP);
      fpour;
      L:=L,[Le];
    fpour;
    Rep:=Rep,L;
  fpour;
  retourne Rep;
  }:;
```

On tape :
`D4:=repart4(L4,P4)`
On obtient :
`[1,2,3,4,5],[1,6,7,8,9],[1,10,11,12,13],[1,14,15,16,17],[1,18,19,20,21],`
`[2,6,10,14,18],[2,7,11,15,19],[2,8,12,16,20],[2,9,13,17,21],`
`[3,6,11,16,21],[3,7,10,17,20],[3,8,13,14,19],[3,9,12,15,18],`
`[4,6,15,20,13],[4,7,14,21,12],[4,8,17,18,11],[4,9,16,19,10],`
`[5,6,19,12,17],[5,7,18,13,16],[5,8,21,10,15],[5,9,20,11,14]`
On rappelle le fonction `testcarte(L)` qui permet de tester le résultat :

```
testcarte(L):={
local j,k,s,S;
s:=size(L);
S:=set[op(L[j])] $ (j=0..s-1);
pour j de 0 jusque s-2 faire
  pour k de j+1 jusque s-1 faire
    si size(S[j] intersect S[k])!=1 alors
      retourne [j,k]
    fsi;
  fpour;
fpour;
retourne 1;
}:;
```

On tape :
`testcarte(D4)`

On obtient :
1

La répartition pour $p = 8$

On rappelle les fonctions qui permettent d'additionner et de multiplier les éléments du corps fini, puis on définit `K8()` et on écrit la fonction de répartition `repart8` :

```
indice(a,L):={
local k;
k:=member(normal(a),L);
 si  k!=0 alors
   retourne k-1;
 fsi;
 retourne "erreur";
}:;
addi(a,b,K,P):={
   retourne indice((K[a]+K[b]) mod (P),K);
}:;
prod(a,b,K,P):={
   retourne indice((K[a]*K[b])mod (P),K);
}:;
prodl(a,L,K,P):={
  local s;
  s:=size(L)-1;
  retourne [prod(a,L[k],K,P)$(k=0..s)];
  }:;
addill(L1,L2,K,P):={
  local s1,s2;
  s1:=size(L1)-1;
  s2:=size(L2)-1;
  si s1==s2 alors
  retourne [addi(L1[k],L2[k],K,P)$(k=0..s1)];
  fsi;
retourne "erreur";
}:;
K8():={
  local L8,j,k,m,s,x;
  purge(x);
  L8:=NULL;
  pour j de 0 jusque 1 faire
  pour k de 0 jusque 1 faire
    pour m de 0 jusque 1 faire
      s:=j*x^2+x*k+m;
      L8:=L8,normal(s %2) ;
      fpour;
    fpour;
  fpour;
```

```
  return [L8];
    }:;
P8:=(x^3+x+1)%2:;
L8:=K8():;
repart8(L8,P8):={
  local k,j,L,Le,n,Rep;
  PP:=[0,0,0],[1,0,0],[0,1,0],([1,k,0])$(k=1..7);
  pour k de 0 jusque 7 faire
    pour j de 0 jusque 7 faire
      PP:=PP,[j,k,1];
    fpour;
  fpour;
  PP:=[PP];
  Rep:=[1,(k+j*8)$(k=2..9)]$(j=0..8);
  pour j de 2 jusque 9 faire
    L:=NULL;
    pour k de 10 jusque 17 faire
      Le:=j;
      pour n de 0 jusque 7 faire
       Le:=Le,indice(addill(PP[k],prodl(n,PP[j],L8,P8),L8,P8),PP);
      fpour;
      L:=L,[Le];
    fpour;
    Rep:=Rep,L;
  fpour;
  retourne Rep;
  }:;
```

On tape :

```
D8:=repart8(L8,P8)
```

On obtient :

```
[1,2,3,4,5,6,7,8,9],[1,10,11,12,13,14,15,16,17],[1,18,19,20,21,22,23,24,25],
[1,26,27,28,29,30,31,32,33],[1,34,35,36,37,38,39,40,41],
[1,42,43,44,45,46,47,48,49],[1,50,51,52,53,54,55,56,57],
[1,58,59,60,61,62,63,64,65],[1,66,67,68,69,70,71,72,73],
[2,10,18,26,34,42,50,58,66],[2,11,19,27,35,43,51,59,67],
[2,12,20,28,36,44,52,60,68],[2,13,21,29,37,45,53,61,69],
[2,14,22,30,38,46,54,62,70],[2,15,23,31,39,47,55,63,71],
[2,16,24,32,40,48,56,64,72],[2,17,25,33,41,49,57,65,73],
[3,10,19,28,37,46,55,64,73],[3,11,18,29,36,47,54,65,72],
[3,12,21,26,35,48,57,62,71],[3,13,20,27,34,49,56,63,70],
[3,14,23,32,41,42,51,60,69],[3,15,22,33,40,43,50,61,68],
[3,16,25,30,39,44,53,58,67],[3,17,24,31,38,45,52,59,66],
[4,10,27,44,61,38,23,72,57],[4,11,26,45,60,39,22,73,56],
[4,12,29,42,59,40,25,70,55],[4,13,28,43,58,41,24,71,54],
[4,14,31,48,65,34,19,68,53],[4,15,30,49,64,35,18,69,52],
[4,16,33,46,63,36,21,66,51],[4,17,32,47,62,37,20,67,50],
[5,10,35,60,53,70,47,24,33],[5,11,34,61,52,71,46,25,32],
```

```
[5,12,37,58,51,72,49,22,31],[5,13,36,59,50,73,48,23,30],
[5,14,39,64,57,66,43,20,29],[5,15,38,65,56,67,42,21,28],
[5,16,41,62,55,68,45,18,27],[5,17,40,63,54,69,44,19,26],
[6,10,43,36,69,62,31,56,25],[6,11,42,37,68,63,30,57,24],
[6,12,45,34,67,64,33,54,23],[6,13,44,35,66,65,32,55,22],
[6,14,47,40,73,58,27,52,21],[6,15,46,41,72,59,26,53,20],
[6,16,49,38,71,60,29,50,19],[6,17,48,39,70,61,28,51,18],
[7,10,51,20,45,30,71,40,65],[7,11,50,21,44,31,70,41,64],
[7,12,53,18,43,32,73,38,63],[7,13,52,19,42,33,72,39,62],
[7,14,55,24,49,26,67,36,61],[7,15,54,25,48,27,66,37,60],
[7,16,57,22,47,28,69,34,59],[7,17,56,23,46,29,68,35,58],
[8,10,59,68,21,54,39,32,49],[8,11,58,69,20,55,38,33,48],
[8,12,61,66,19,56,41,30,47],[8,13,60,67,18,57,40,31,46],
[8,14,63,72,25,50,35,28,45],[8,15,62,73,24,51,34,29,44],
[8,16,65,70,23,52,37,26,43],[8,17,64,71,22,53,36,27,42],
[9,10,67,52,29,22,63,48,41],[9,11,66,53,28,23,62,49,40],
[9,12,69,50,27,24,65,46,39],[9,13,68,51,26,25,64,47,38],
[9,14,71,56,33,18,59,44,37],[9,15,70,57,32,19,58,45,36],
[9,16,73,54,31,20,61,42,35],[9,17,72,55,30,21,60,43,34]
```

On tape :

```
testcarte(D8)
```

On obtient :

```
1
```

La répartition pour $p = 9$

On définit `K9()` et on écrit la fonction de répartition `repart9` :

```
K9():={
  local L9,j,k,s,x;
  purge(x);
  L9:=NULL;
  pour k de 0 jusque 2 faire
    pour j de 0 jusque 2 faire
      s:=x*k+j;
      L9:=L9,normal(s %3) ;
      fpour;
    fpour;
  return [L9];
    }:;
P9:=(x^2+1)%3:;
L9:=K9():;
repart9(L9,P9):={
  local k,j,L,Le,n,Rep,PP;
  PP:=[0,0,0],[1,0,0],[0,1,0],([1,k,0])$(k=1..8);
  pour k de 0 jusque 8 faire
    pour j de 0 jusque 8 faire
      PP:=PP,[j,k,1];
    fpour;
```

```
  fpour;
  PP:=[PP];
  Rep:=[1,(k+j*9)$(k=2..10)]$(j=0..9);
  pour j de 2 jusque 10 faire
    L:=NULL;
    pour k de 11 jusque 19 faire
      Le:=j;
      pour n de 0 jusque 8 faire
        Le:=Le,indice(addill(PP[k],prodl(n,PP[j],L9,P9),L9,P9),PP);
      fpour;
      L:=L,[Le];
    fpour;
    Rep:=Rep,L;
  fpour;
  retourne Rep;
  }:;
```

On tape :

```
D9:=repart(L9,P9))
```

On obtient :

```
[1,2,3,4,5,6,7,8,9,10],[1,11,12,13,14,15,16,17,18,19],
[1,20,21,22,23,24,25,26,27,28],[1,29,30,31,32,33,34,35,36,37],
[1,38,39,40,41,42,43,44,45,46],[1,47,48,49,50,51,52,53,54,55],
[1,56,57,58,59,60,61,62,63,64],[1,65,66,67,68,69,70,71,72,73],
[1,74,75,76,77,78,79,80,81,82],[1,83,84,85,86,87,88,89,90,91],
[2,11,20,29,38,47,56,65,74,83],[2,12,21,30,39,48,57,66,75,84],
[2,13,22,31,40,49,58,67,76,85],[2,14,23,32,41,50,59,68,77,86],
[2,15,24,33,42,51,60,69,78,87],[2,16,25,34,43,52,61,70,79,88],
[2,17,26,35,44,53,62,71,80,89],[2,18,27,36,45,54,63,72,81,90],
[2,19,28,37,46,55,64,73,82,91],[3,11,21,31,41,51,61,71,81,91],
[3,12,22,29,42,52,59,72,82,89],[3,13,20,30,43,50,60,73,80,90],
[3,14,24,34,44,54,64,65,75,85],[3,15,25,32,45,55,62,66,76,83],
[3,16,23,33,46,53,63,67,74,84],[3,17,27,37,38,48,58,68,78,88],
[3,18,28,35,39,49,56,69,79,86],[3,19,26,36,40,47,57,70,77,87],
[4,11,30,22,68,87,79,44,63,55],[4,12,31,20,69,88,77,45,64,53],
[4,13,29,21,70,86,78,46,62,54],[4,14,33,25,71,90,82,38,57,49],
[4,15,34,23,72,91,80,39,58,47],[4,16,32,24,73,89,81,40,56,48],
[4,17,36,28,65,84,76,41,60,52],[4,18,37,26,66,85,74,42,61,50],
[4,19,35,27,67,83,75,43,59,51],[5,11,39,67,32,60,88,26,54,82],
[5,12,40,65,33,61,86,27,55,80],[5,13,38,66,34,59,87,28,53,81],
[5,14,42,70,35,63,91,20,48,76],[5,15,43,68,36,64,89,21,49,74],
[5,16,41,69,37,62,90,22,47,75],[5,17,45,73,29,57,85,23,51,79],
[5,18,46,71,30,58,83,24,52,77],[5,19,44,72,31,56,84,25,50,78],
[6,11,48,85,59,69,25,80,36,46],[6,12,49,83,60,70,23,81,37,44],
[6,13,47,84,61,68,24,82,35,45],[6,14,51,88,62,72,28,74,30,40],
[6,15,52,86,63,73,26,75,31,38],[6,16,50,87,64,71,27,76,29,39],
[6,17,54,91,56,66,22,77,33,43],[6,18,55,89,57,67,20,78,34,41],
[6,19,53,90,58,65,21,79,32,42],[7,11,57,76,86,24,43,53,72,37],
[7,12,58,74,87,25,41,54,73,35],[7,13,56,75,88,23,42,55,71,36],
```

```
[7,14,60,79,89,27,46,47,66,31],[7,15,61,77,90,28,44,48,67,29],
[7,16,59,78,91,26,45,49,65,30],[7,17,63,82,83,21,40,50,69,34],
[7,18,64,80,84,22,38,51,70,32],[7,19,62,81,85,20,39,52,68,33],
[8,11,66,40,23,78,52,35,90,64],[8,12,67,38,24,79,50,36,91,62],
[8,13,65,39,25,77,51,37,89,63],[8,14,69,43,26,81,55,29,84,58],
[8,15,70,41,27,82,53,30,85,56],[8,16,68,42,28,80,54,31,83,57],
[8,17,72,46,20,75,49,32,87,61],[8,18,73,44,21,76,47,33,88,59],
[8,19,71,45,22,74,48,34,86,60],[9,11,75,58,50,33,70,89,45,28],
[9,12,76,56,51,34,68,90,46,26],[9,13,74,57,52,32,69,91,44,27],
[9,14,78,61,53,36,73,83,39,22],[9,15,79,59,54,37,71,84,40,20],
[9,16,77,60,55,35,72,85,38,21],[9,17,81,64,47,30,67,86,42,25],
[9,18,82,62,48,31,65,87,43,23],[9,19,80,63,49,29,66,88,41,24],
[10,11,84,49,77,42,34,62,27,73],[10,12,85,47,78,43,32,63,28,71],
[10,13,83,48,79,41,33,64,26,72],[10,14,87,52,80,45,37,56,21,67],
[10,15,88,50,81,46,35,57,22,65],[10,16,86,51,82,44,36,58,20,66],
[10,17,90,55,74,39,31,59,24,70],[10,18,91,53,75,40,29,60,25,68],
[10,19,89,54,76,38,30,61,23,69]
```

On tape :

```
testcarte(D9)
```

On obtient :

```
1
```

8.11.4 Pour un corps fini K ayant $p = q^n$ éléments avec q premier

Si $Lp = \mathbb{Z}/q\mathbb{Z}[x] \bmod Pp$ On rappelle les fonctions qui permettent d'additionner et de multiplier les éléments du corps fini, puis on écrit la fonction de répartition `repartp` :

```
indice(a,L):={
local k;
k:=member(normal(a),L);
 si  k!=0 alors
   retourne k-1;
 fsi;
 retourne "erreur";
}:;
addi(a,b,K,P):={
   retourne indice((K[a]+K[b]) mod (P),K);
}:;
prod(a,b,K,P):={
   retourne indice((K[a]*K[b])mod (P),K);
}:;
prodl(a,L,K,P):={
  local s;
  s:=size(L)-1;
  retourne [prod(a,L[k],K,P)$(k=0..s)];
  }:;
addill(L1,L2,K,P):={
  local s1,s2;
```

```
  s1:=size(L1)-1;s2:=size(L2)-1;
  si s1==s2 alors
  retourne [addi(L1[k],L2[k],K,P)$(k=0..s1)];
  fsi;
retourne "erreur";
}:;
//si p=q^n avec q premier, Lp est la liste des poly de Z/qZ de degré<=n-1
//Pp est un poly premier du corps Lp
//PP est le plan projectif et s=p-1
repartp(Lp,Pp):={
  local k,j,L,Le,m,Rep,PP,s;
  s:=size(Lp)-1;
  PP:=[0,0,0],[1,0,0],[0,1,0],([1,k,0])$(k=1..s);
  pour k de 0 jusque s faire
    pour j de 0 jusque s faire
      PP:=PP,[j,k,1];
    fpour;
  fpour;
  PP:=[PP];
  Rep:=[1,(k+j*(s+1))$(k=2..s+2)]$(j=0..s+1);
  pour j de 2 jusque s+2 faire
    L:=NULL;
    pour k de s+3 jusque 2*s+3 faire
      Le:=j;
      pour m de 0 jusque s faire
        Le:=Le,indice(addill(PP[k],prodl(m,PP[j],Lp,Pp),Lp,Pp),PP);
      fpour;
      L:=L,[Le];
    fpour;
    Rep:=Rep,L;
  fpour;
  retourne Rep;
  }:;
```

Par exemple si $Lp = \mathbb{Z}/3\mathbb{Z}[x] \bmod x^3 + 2x + 1$ on construit un corps de cardinal $3^3 = 27$.
On tape :

```
K27():={
  local Lp,j,k,l,s,x;
  purge(x);
  Lp:=NULL;
  pour j de 0 jusque 2 faire
   pour k de 0 jusque 2 faire
    pour l de 0 jusque 2 faire
      s:=j*x^2+x*k+l;
      Lp:=Lp,normal(s % 3) ;
    fpour;
```

```
   fpour;
  fpour;
  return [Lp];
  }:;
L27:=K27():;
P27:=(x^3+2*x+1)%3:;
```

On tape `D27:=repartp(L27,P27):;`
On obtient (Temps mis pour l'évaluation : 80)
`Done`
On tape `size(L27),size(D27)`
On obtient car $27 * 28 + 1 = 757$:
`27,757`
On a donc 757 objets différents et $D27$ donne la répartition de 27+1=28 objets sur chacune des 757 cartes.
On tape pour vérifier :
`textcarte(D27)`
On obtient `1`

8.11.5 Exercice

Trouver la répartition lorsque l'on veut un jeu ayant :

1. 6 objets par carte et 5*6+1=31 cartes
2. 12 objets par carte et 11*12+1=133 cartes
3. 17 objets par carte et 16*17+1=273 cartes
4. 26 objets par carte et 25*26+1=651 cartes

Solution

1. 6 objets par carte et 5*6+1=31 cartes
   On tape puisque 11 est premier :

```
K5():={
  local Lp,j,s,x;
  purge(x);
  Lp:=NULL;
  pour j de 0 jusque 4 faire
      s:=j;
      Lp:=Lp,normal(s %5) ;
  fpour;
  return [Lp];
    }:;
P5:=(x)%5:;
L5:=K5():;
```

   On tape `D5:=repartp(L5,P5):;`
   On obtient :
   `Done`

On tape :
`size(L5),size(D5)`
On obtient car $5*6+1=31$:
`5,31`
On tape pour vérifier :
`testcarte(D5)`
On obtient :
`1`
On tape :
`D5`
On obtient :
```
[1,2,3,4,5,6],[1,7,8,9,10,11],[1,12,13,14,15,16],[1,17,18,19,20,21],
[1,22,23,24,25,26],[1,27,28,29,30,31],[2,7,12,17,22,27],[2,8,13,18,23,28
[2,9,14,19,24,29],[2,10,15,20,25,30],[2,11,16,21,26,31],[3,7,13,19,25,31
[3,8,14,20,26,27],[3,9,15,21,22,28],[3,10,16,17,23,29],[3,11,12,18,24,30
[4,7,18,29,15,26],[4,8,19,30,16,22],[4,9,20,31,12,23],[4,10,21,27,13,24]
[4,11,17,28,14,25],[5,7,23,14,30,21],[5,8,24,15,31,17],[5,9,25,16,27,18]
[5,10,26,12,28,19],[5,11,22,13,29,20],[6,7,28,24,20,16],[6,8,29,25,21,12
[6,9,30,26,17,13],[6,10,31,22,18,14],[6,11,27,23,19,15]
```

2. 12 objets par carte et 11*12+1=133 cartes
On tape puisque 11 est premier

```
K11():={
  local Lp,j,s,x;
  purge(x);
  Lp:=NULL;
  pour j de 0 jusque 10 faire
      s:=j;
      Lp:=Lp,normal(s %11) ;
  fpour;
  return [Lp];
    }:;
P11:=(x)%11:;
L11:=K11():;
```

On tape `D11:=repartp(L11,P11):;`
On obtient (Temps mis pour l'évaluation : 1.8) :
`Done`
On tape :
`size(L11),size(D11)`
On obtient car $11*12+1=133$:
`11,133`
On tape pour vérifier :
`testcarte(D11)`
On obtient :
`1`

3. 17 objets par carte et 16*17+1=273 cartes
On définit le corps $K16$ ayant 16 éléments :
On tape puisque $16=2^4$:

```
K16():={
  local Lp,j,k,l,m,s,x;
  purge(x);
  Lp:=NULL;
  pour m de 0 jusque 1 faire
   pour j de 0 jusque 1 faire
    pour k de 0 jusque 1 faire
     pour l de 0 jusque 1 faire
      s:=m*x^3+j*x^2+x*k+l;
      Lp:=Lp,normal(s %2) ;
     fpour;
    fpour;
   fpour;
  fpour;
  return [Lp];
  }:;
L16:=K16():;
P16:=(x^4+x+1)%2:;
```

On tape `D16:=repartp(L16,P16):;`
On obtient (Temps mis pour l'évaluation : 13.78) :
`Done`
On tape :
`size(L16),size(D16)`
On obtient car $16 * 17 + 1 = 273$:
`16,273`
On tape pour vérifier :
`testcarte(D16)`
On obtient :
`1`

4. 26 objets par carte et 25*26+1=651 cartes
   On tape puisque $25 = 5^2$:

```
K25():={
  local Lp,j,k,s,x;
  purge(x);
  Lp:=NULL;
  pour k de 0 jusque 4 faire
    pour j de 0 jusque 4 faire
      s:=x*k+j;
      Lp:=Lp,normal(s %5) ;
    fpour;
  fpour;
  return [Lp];
    }:;
P25:=(x^2+2)%5:;
L25:=K25():;
```

On tape :
`D25:=repartp(L25,P25):;`
On obtient (Temps mis pour l'évaluation : 48.21) :
`Done`
On tape :
`size(L25),size(D25)`
On obtient car $25 * 26 + 1 = 651$:
`25,651`
On tape pour vérifier :
`testcarte(D25)`
On obtient :
`1`

8.11.6 En utilisant la commande `GF` de `Xcas`

Tout d'abord il faut construire le corps Lp de cardinal $p = q^n$ (q premier), puis la commande `GF` nous donnera un polynôme irréductible Pp de $\mathbb{Z}/q\mathbb{Z}[x]$ de degré n, puis on réécrit un programme de répartition `repart(p)` ayant un seul patramètre $p = q^n$ avec q premier.

— Construction de Lp.
On utilise `a .+ L` pour renvoyer une liste d'éléments `a+L[k]` pour `k=0..size(L)-1`
On tape :

```
polyp(p):={
  local L1,j,m,LP,P,x,f,n,q;
  f:=ifactors(p);
  si size(f)==2 alors (q,n):=f; sinon return "erreur"; fsi;
  L1:=[(j % q)$(j=0..q-1)];
  LP:=L1;
  purge(x);
  P:=x;
  pour m de 1 jusque n-1  faire
     pour j de 1 jusque q-1  faire
        LP:=append(LP,op(L1[j]*P .+ L1));
    fpour;
    L1:=LP;
    P:=P*x;
  fpour;
  return normal(LP);
}:;
```

On tape pour $p = 2^3 = 8$:
`polyp(8)`
On obtient :
`[0 % 2,1 % 2,(1 % 2)*x,1 % 2+1 % 2*x,(1 % 2)*x^2,1 % 2+(1 % 2)*x^2,(1 % 2)*x+(1 % 2)*x^2,1 % 2+(1 % 2)*x+(1 % 2)*x^2]`
Ou bien on utilise l'écriture en base q des q^n entiers `0,1,.. size(Lp)-1`

c'est alors la fonction `poly(p)` (cf ci dessous). On tape $p = 2^3 = 8$ donc $q = 2$ et $n = 3$:

```
L:=[j$(j=0..7)]
PP:=revlist(convert(L[j],base,2))$(j=0..7)
[normal(poly2symb(PP[j] % 2,x))$(j=0..7)]
```

On obtient :

```
[[0 % 2,1 % 2,(1 % 2)*x,1 % 2+(1 % 2)*x,(1 % 2)*x^2,1
% 2+(1 % 2)*x^2,(1 % 2)*x+(1 % 2)*x^2,1 % 2+(1 % 2)*x+(1
% 2)*x^2]]
```

— Pour obtenir un polynôme irréductible Pp de $\mathbb{Z}/q\mathbb{Z}[x]$ de degré n si $p = q^n$.
La commande `GF` nous donne un polynôme irréductible Pp de $\mathbb{Z}/q\mathbb{Z}[x]$ de degré n lorsque $p = q^n$.
Par exemple pour $p = 2^3 = 8$, on tape :

```
purge(g);GF(8)
```

On obtient :

```
GF(2,k^3+k^2+1,[k,Q,g],undef)
```

On tape :

```
Pp:=normal(poly2symb(pmin(g),x))
```

On obtient :

```
1 % 2+(1 % 2)*x^2+(1 % 2)*x^3
```

— Réécriture du programme de répartition ayant comme seul patramètre p Si $Lp = \mathbb{Z}/q\mathbb{Z}[x] \bmod Pp$ On rappelle les fonctions qui permettent d'additionner et de multiplier les éléments du corps fini, puis on écrit la fonction `poly(p)` qui renvoie `Lp` et on écrit la fonction `repartip(p)` qui renvoie la répartition lorsque $p = q^n$ avec q premier à savoir : on a $p*(p+1)+1$ cartes et $p * (p + 1) + 1$ objets différents (ici les objets sont les entiers $1..p * (p + 1) + 1$) et on place sur chaque carte $p + 1$ objets de façon que 2 cartes quelconques aient en commun 1 et 1 seul objet.

```
indice(a,L):={
local k;
k:=member(normal(a),L);
 si  k!=0 alors
   retourne k-1;
 fsi;
 retourne "erreur";
}:;
addi(a,b,K,P):={
   retourne indice((K[a]+K[b]) mod (P),K);
}:;
prod(a,b,K,P):={
   retourne indice((K[a]*K[b])mod (P),K);
}:;
prodl(a,L,K,P):={
  local s;
  s:=size(L)-1;
  retourne [prod(a,L[k],K,P)$(k=0..s)];
  }:;
addill(L1,L2,K,P):={
```

```
  local s1,s2;
  s1:=size(L1)-1;
  s2:=size(L2)-1;
  si s1==s2 alors
  retourne [addi(L1[k],L2[k],K,P)$(k=0..s1)];
  fsi;
retourne "erreur";
};;
poly(p):={
  local Lp,j,L,L1,x,f,n,q,s;
  f:=ifactors(p);
  purge(x);
  si size(f)==2 alors (q,n):=f; sinon return "erreur"; fsi;
  s:=p-1;
  L:=[j$(j=0..s)];
  L1:=revlist(convert(L[j],base,q))$(j=0..s);
  Lp:=[normal(poly2symb(L1[j] % q,x))$(j=0..s)];
  return Lp;
};;
repartip(p):={
  local k,j,L,Le,Rep,PP,s,Lp,Pp,q,n,m,f,G,g,x;
  purge(x);
  f:=ifactors(p);
  si size(f)==2 alors (q,n):=f; sinon return "erreur1"; fsi;
  si est_premier(q)!=1 alors retourne "erreur2" fsi;
  Lp:=poly(p);
  s:=p-1;
  si n==1 alors Pp:=x; sinon
    purge(g);
    G:=GF(p);
    Pp:=normal(poly2symb(pmin(g),x));
  fsi;
  PP:=[0,0,0],[1,0,0],[0,1,0],([1,k,0])$(k=1..s);
  pour k de 0 jusque s faire
    pour j de 0 jusque s faire
      PP:=PP,[j,k,1];
    fpour;
  fpour;
  PP:=[PP];
  Rep:=[1,(k+j*(s+1))$(k=2..s+2)]$(j=0..s+1);
  pour j de 2 jusque s+2 faire
    L:=NULL;
    pour k de s+3 jusque 2*s+3 faire
      Le:=j;
      pour m de 0 jusque s faire
        Le:=Le,indice(addill(PP[k],prodl(m,PP[j],Lp,Pp),Lp,Pp),PP);
      fpour;
      L:=L,[Le];
```

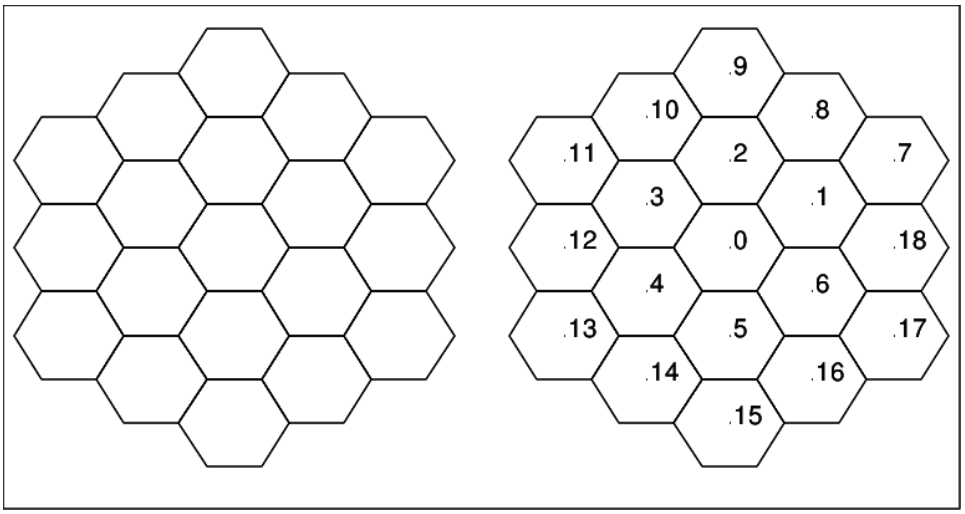
```
    fpour;
    Rep:=Rep,L;
  fpour;
  retourne Rep;
  }:;
```

Par exemple si $Lp = \mathbb{Z}/3\mathbb{Z}[x] \bmod x^3 + 2x + 1$ on construit un corps de cardinal $3^3 = 27$.
On tape :
`D3:=repartip(3):;` On obtient :
`Done`
On tape pour vérifier :
`testcarte(D3)`
On obtient :
`1` On tape :
`D4:=repartip(4):;`
On obtient :
`Done`
On tape pour vérifier :
`testcarte(D4):;`
On obtient :
`1`
On tape :
`D27:=repartip(27):;`
On obtient (Temps mis pour l'évaluation : 78.48) :
`Done`
On tape :
`size(D27)`
On obtient car $27 * 28 + 1 = 757$:
`757`
On tape pour vérifier :
`testcarte(D27):;`
On obtient (Temps mis pour l'évaluation : 10.4) :
`1`

8.12 Un puzzle

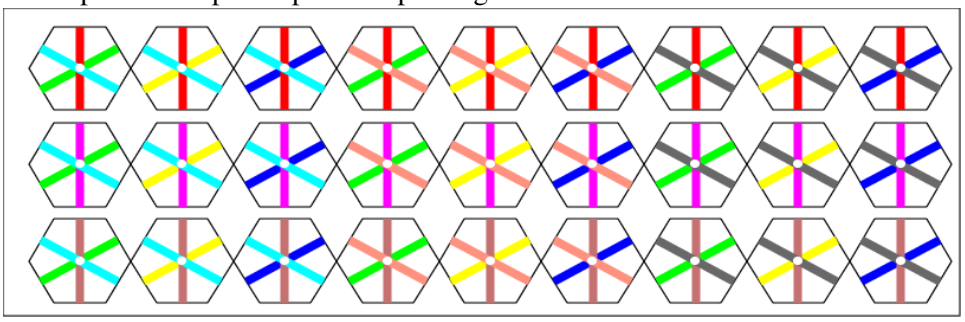
8.12.1 La consigne

La grille
La grille est formée de 19 hexagones.
Pour faciliter les explications, on repère ces hexagones en numérotant leur centre :



Les pièces
Pour faire les pièces on utilise 9 couleurs de code :
1,2,3,4,5,6,77,8 et 9. (on n'utilise pas le code 7 car c'est le code du blanc)

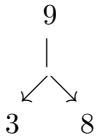
On dispose de 27 pièces pour remplir la grille :



Le puzzle à réaliser
Il faut remplir la grille de façon à assembler 2 hexagones selon un côté de même couleur.

8.12.2 Analyse du problème

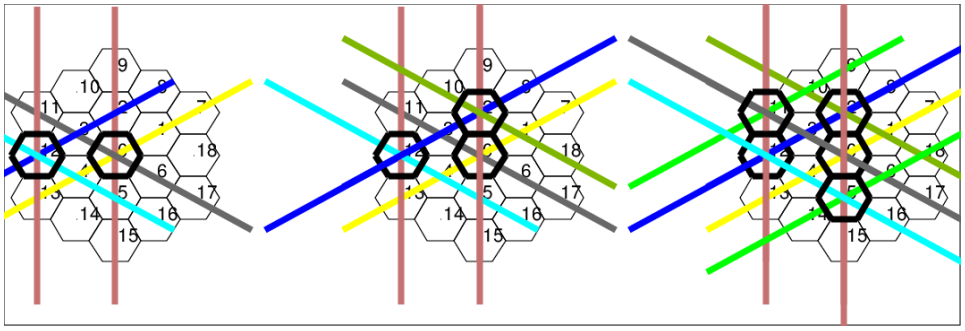
On met au départ une pièce au centre du puzzle, par exemple la pièce :



On nomme cette pièce 398.
Mettre la piece 398 en position 0 cela impose non seulement la couleur des droites passant par le centre du puzzle, mais aussi les couleurs possibles des obliques et des verticales.
Puisque les pièces contenant 9 sont 296,2977,298,396,3977,398,496,4977,496,
les 5 obliques de pente positive (notées $O+$) peuvent avoir comme couleur 2, 3 ou 4,
les 5 obliques de pente negative (notées $O-$) peuvent avoir comme couleur 6, 77 ou 8 et
les 5 verticales (notées $V1, V2, V3, V4, V5$) qui peuvent avoir comme couleur 1, 5 ou 9.
Remarque 1
On ne peut pas avoir 3 droites V (resp 3 droites $O+$, 3 droites $O-$) de la même couleur.

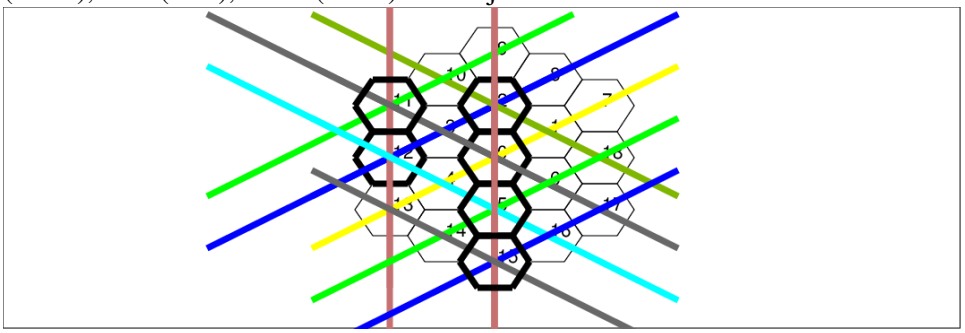
Hypothèse 1
$V2$ et $V3$ peuvent-elle avoir la même couleur ?
Par exemple, peut-on mettre la piece x9y en 3 ?
Cela entraine que toutes les pieces contenant 9 seront utilisées.
Cela n'est pas possible car en regardant la couleur des obliques $O+$ on a :
en position 0 et 4 il faut mettre 398 et 396 ou 3977
en position 3 et 2 il faut mettre 298 et 296 ou 2977
Donc on remplit les positions 10 et 9 avec 496 ou 4977 ou 498
Il reste donc 3 pièces pour remplir les positions 14 et 5 mais ces 3 pièces ont des $O+$ de couleurs différentes. C'est donc impossible.
Donc on a :
Conclusion 1 $V2$ **et** $V3$ **doivent être de couleurs différentes** :
les droites contenant 4 pièces et les droites passant par le centre ne peuvent pas avoir la même couleur.

Hypothèse 2
$V1$ et $V3$ peuvent-elle avoir la même couleur ?
Par exemple, peut-on mettre x9y en 12 ?
D'après la conclusion 1 on ne peut mettre que 29x ou 49x (x=6 ou 77)
Peut-on mettre la piece 496 en 12 ?

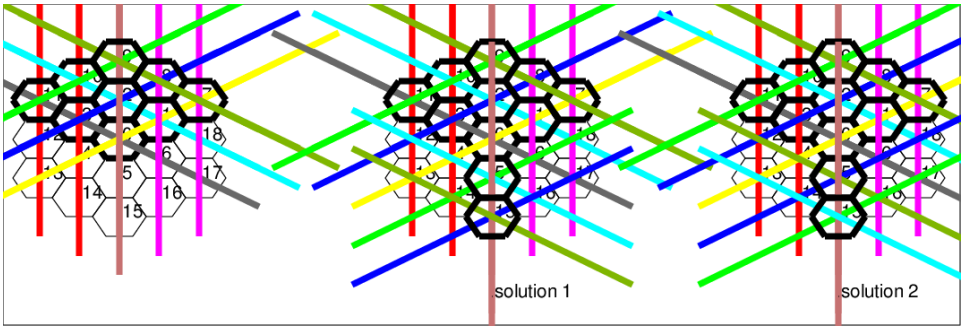


Si la piece 496 est en 12, cela entraine qu'il faut mettre :
4977 en 2 (puisque 498 n'est pas possible d'après la conclusion 1),
296 en 5 (puisque 396 n'est pas possible d'après la conclusion 1 et 496 n'est pas

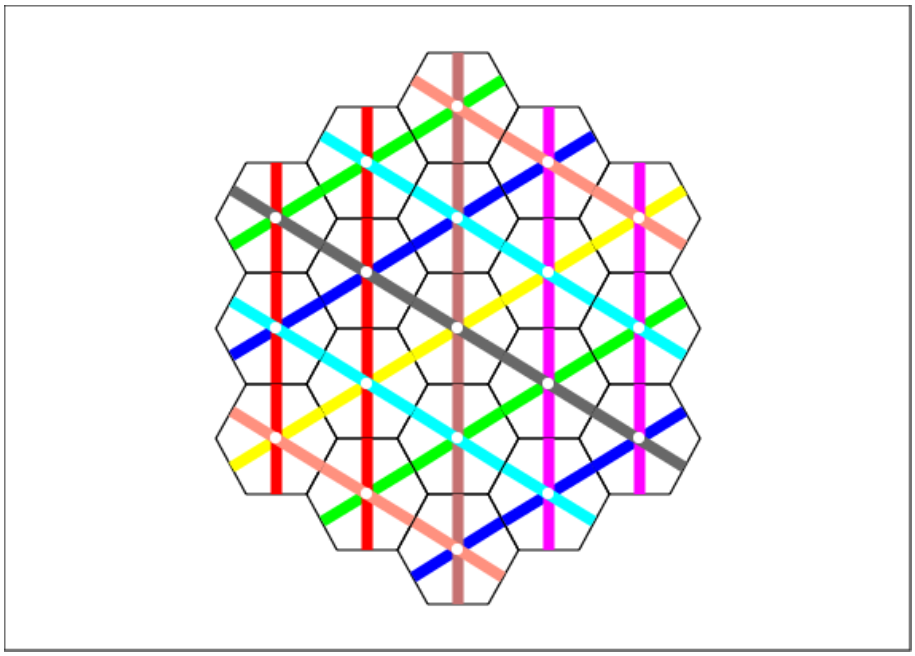
possible car 496 est en 12)
298 en 11 (car 398 est en 0 et on ne peut pas mettre 498 en 11 puisque cela entrainerai que l'on ne peut pas remplir la position 9 avec un 49x puisque les pièces 496 (en 12),4977 (en 2),et 498 (en 11) sont déjà utilisées.



Que peut-on mettre en 15 ? il y a 2 droites $O+$ de couleur 2 donc on ne peut pas mettre 29x (remarque 1) on ne peut pas mettre 396 ou 397 car alors les pièces en position 13 et 15 seraient de la même couleur
496 (en 12), 4977 (en 2) donc il reste 498 ce qui est impossible car alors il faudrait mettre 398 en 13 ! Donc on ne peut pas mettre 496 en 12.
De même on ne peut pas mettre 4977, ni 296 ni 2977 en 12.
Conclusion 2 $V1$ **et** $V3$ **doivent être de couleurs différentes** :
les droites contenant 3 pièces et les droites passant par le centre ne peuvent pas avoir la même couleur.
Hypothèse 3
$V1$ et $V2$ peuvent-elle avoir la même couleur ?
Supposons $V1$, $V2$ de couleur 1 et $V4$, $V5$ de couleur 5.
Plaçons par exemple 418 en 3 (on pourrait aussi mettre 218 en 3 ce qui revient à permuter les couleurs 4 et 2).
Cela entraine qu'il faut mettre 218 en 11 (car on ne peut pas mettre 318 d'après la conclusion2).
On peut mettre par exemple 216 en 10 (on pourrait aussi mettre 217 en 10 ce qui revient à permuter les couleurs 6 et 7).
Donc on doit mettre 356 en position 1 et 496 en position 2.
donc on doit mettre 3577 en position 7 (car 356 est en position 1) et on doit mettre 4577 en 8 et 2977 en 9.



En position 5 on peut mettre 296 (resp 4977)
En position 15 on peut mettre 4977 (resp 296)
On a donc obtenu comme solution :



On notera cette solution : [24324][11955][76867]

Conclusion 3

$V1$ et $V2$ peuvent être de la même couleur.

Hypothèse 4

Les droites situées d'un même côté des droites passant par le centre ont des couleurs différentes : par exemple, $V1$ et $V2$ peuvent-elle avoir des couleurs différentes avec $O1+$, $O2+$ $O1-$ et $O2-$ de couleurs différentes ?

Plaçons par exemple 218 en 11 et 458 en 3 (on pourrait aussi mettre 418 en 11 et 258 en 3 ce qui revient à permuter les couleurs 4 et 2). On ne met pas 218 en 11 et 258 en 3 car alors $O1+$ et $O2+$ serait de la couleur 2 et cela revient à l'étude précédente déjà faite (cf hypothèse 3).

On obtient alors comme grille possible : grille1 : [24324][15915][? ?8 ? ?]

grille2 : [24342][15915][? ?8 ? ?]

grille3 : [24324][15951][? ?8 ? ?]

grille4 : [24342][15951][? ?8 ? ?]

la grille1 est impossible car 218 se trouve en 11 et en 6

la grille4 est impossible car 218 se trouve en 11 et en 17

Essayons de compléter la grille2 [24342][15915][? ?8 ? ?] :

Plaçons par exemple 356 en 4 (resp 357 en 4), cela entraine qu'il faut mettre 357 en 7 (resp 356 en 7).

ce qui est impossible car 496 (resp 497) se trouve en 2 et en 5.

Essayons de compléter la grille3 [24324][15951][? ?8 ? ?] :

Plaçons par exemple 356 en 4 (resp 357 en 4), cela entraine qu'il faut mettre 357 en 7 (resp 356 en 7).

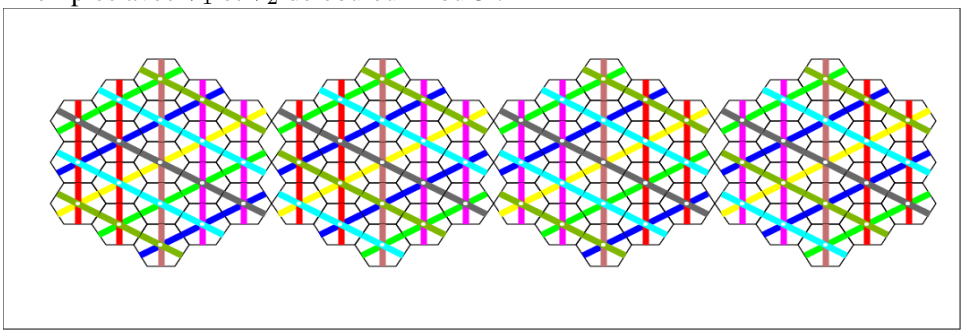
ce qui est impossible car 436 (resp 437) se trouve en 1 et en 4.

En résumé On obtient 8 solutions lorsque 398 est en 0 et $V1$ et $V2$sont de couleur 1 en mettant :

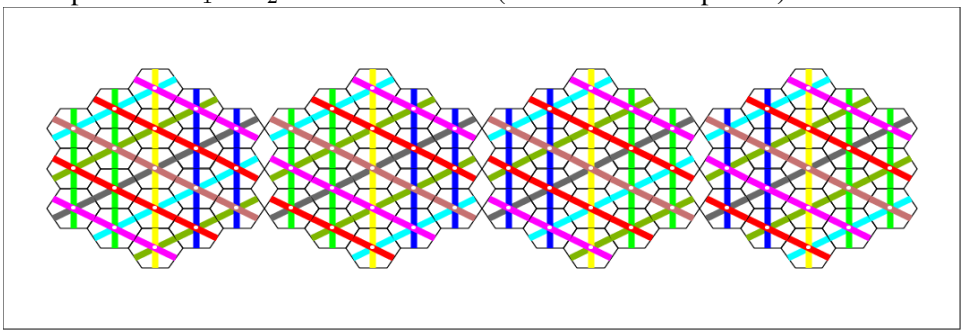
1. 418 en 3, 216 en 10, 296 en 5 et 497 en 15 -> solution : [24324][11955][76867], pièces non utilisées : 298 , 396, 397, 498, 257, 318, 358, 417.

2. 418 en 3, 2177 en 10, 2977 en 5 et 496 en 15 -> solution : [24324][11955][67867].
   pièces non utilisées : 298 , 396, 397, 498, 256, 318, 358, 416.
3. 418 en 3, 216 en 10, 496 en 5 et 2977 en 15 -> solution : [24342][11955][76876],
   pièces non utilisées : 298 , 396, 397, 498, 256, 318, 358, 217.
4. mettre 418 en 3, 2177 en 10 4977 en 5 et 296 en 15 -> solution : [24342][11955][67867],
   pièces non utilisées : 298 , 396, 397, 498, 257, 318, 358, 216.
5. 218 en 3, 416 en 10, 496 en 5 et 2977 en 15 -> solution : [42342][11955][76867],
   pièces non utilisées : 298 , 396, 397, 498, 457, 318, 358, 217.
6. 218 en 3, 4177 en 10, 4977 en 5 et 296 en 15 -> solution : [42342][11955][67876],
   pièces non utilisées : 298 , 396, 397, 498, 456, 318, 358, 216.
7. 218 en 3, 416 en 10, 296 en 5 et 4977 en 15 -> solution : [42324][11955][76876].
   pièces non utilisées : 298 , 396, 397, 498, 456, 318, 358, 417.
8. 218 en 3, 4177 en 10, 2977 en 5 et 496 en 15 -> solution : [42324][11955][67867],
   pièces non utilisées : 298 , 396, 397, 498, 457, 318, 358, 416.

On obtient 8 solutions lorsque 398 est en position 0 et V_1 et V_2 sont de couleur 5.
Exemples avec V_1 et V_2 de couleur 1 ou 5 :

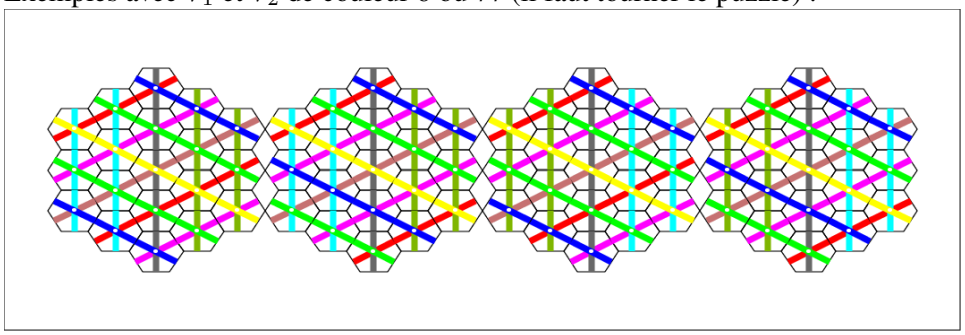


On obtient 8 solutions lorsque 398 est en position 0 avec $O1+$ et $O2+$ sont de couleur 2 (avec la fonction f :[1,2,3,4,5,6,77,8,9]->[2,6,8,77,4,1,5,9,3]).
Puis il faut tourner le puzzle terminé pour avoir 398 en position 0.
On obtient aussi 8 solutions lorsque 398 est en position 0 et $O1+$ et $O2+$ sont de la couleur 4 (pour cela on utilise la fonction g :[1,2,3,4,5,6,77,8,9]->[4,6,8,77,2,1,5,9,3]).
Puis il faut tourner le puzzle terminé pour avoir 398 en position 0.
Exemples avec V_1 et V_2 de couleur 2 ou 4 (il faut tourner le puzzle) :

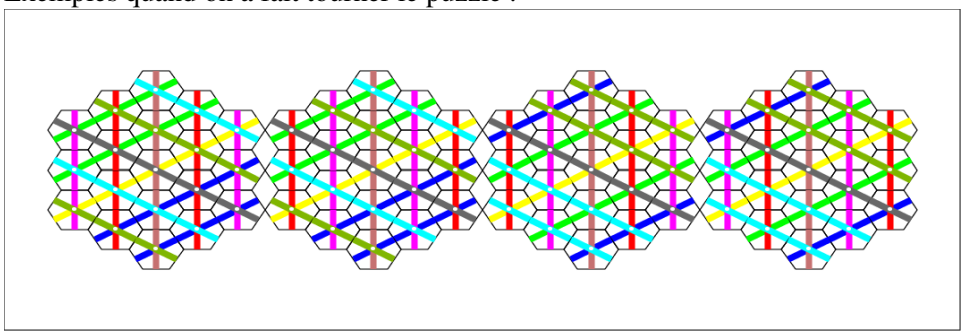


On obtient 8 solutions lorsque 398 est en position 0 et $O1-$ et $O2-$ sont de la couleur 6 (pour cela on utilise la fonction $f \circ f$ et la rotation r).
On obtient 8 solutions lorsque 398 est en position 0 avec $O1+$ et $O2+$ sont de la couleur 7 (pour cela on utilise la fonction $g \circ g$).

Exemples avec V_1 et V_2 de couleur 6 ou 77 (il faut tourner le puzzle) :

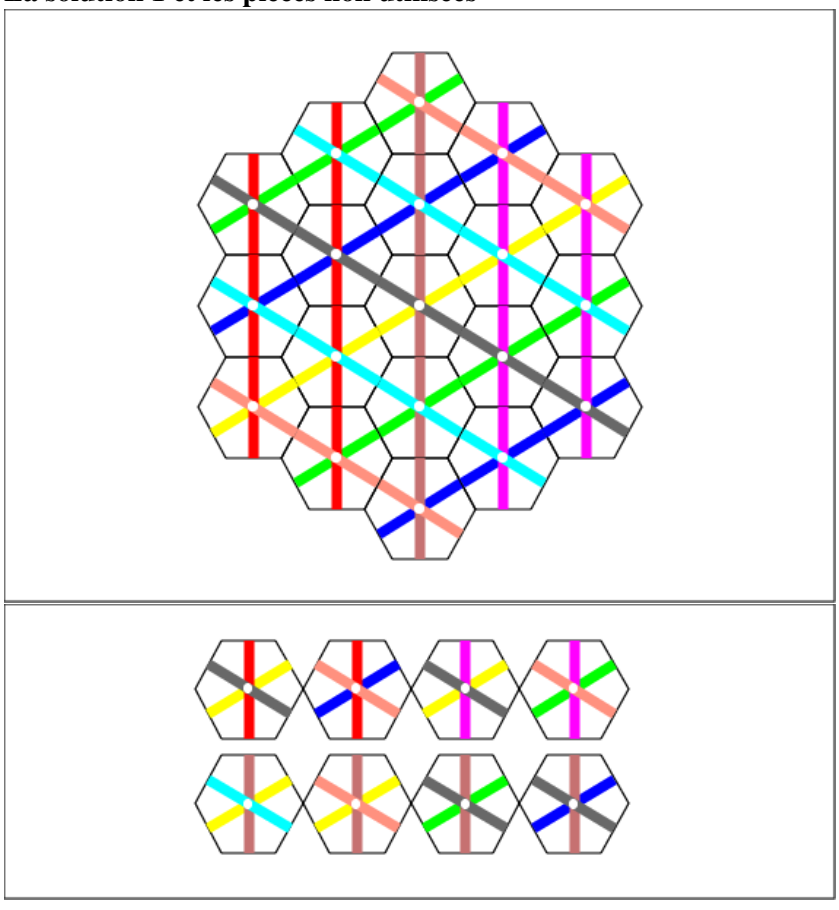


Exemples quand on a fait tourner le puzzle :



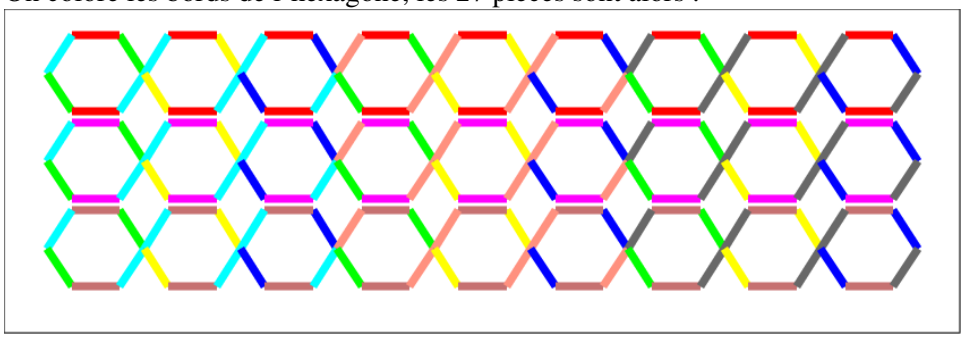
Soit en tout 48 solutions lorsqu'on a posé une pièce au centre du puzzle.

La solution 1 et les pièces non utilisées

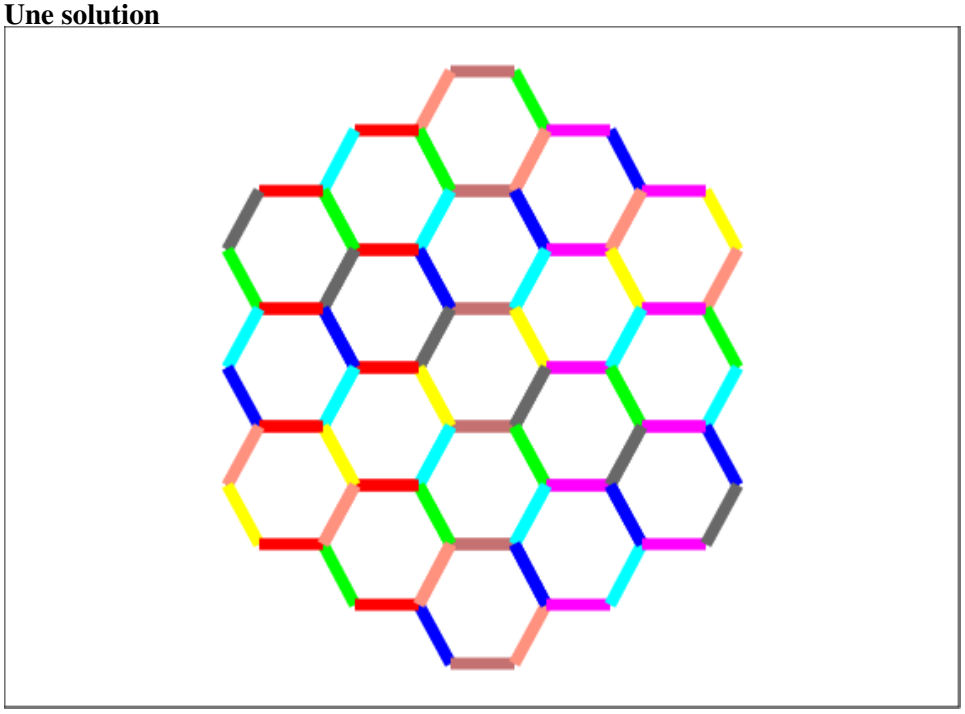


La variante1

On colore les bords de l'hexagone, les 27 pièces sont alors :

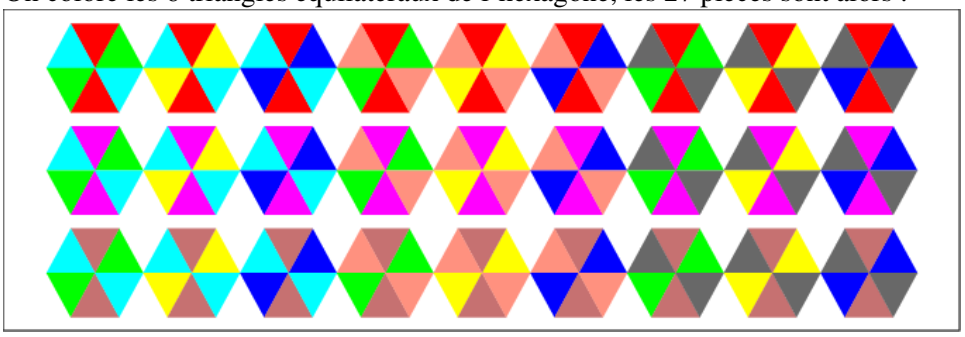


Une solution

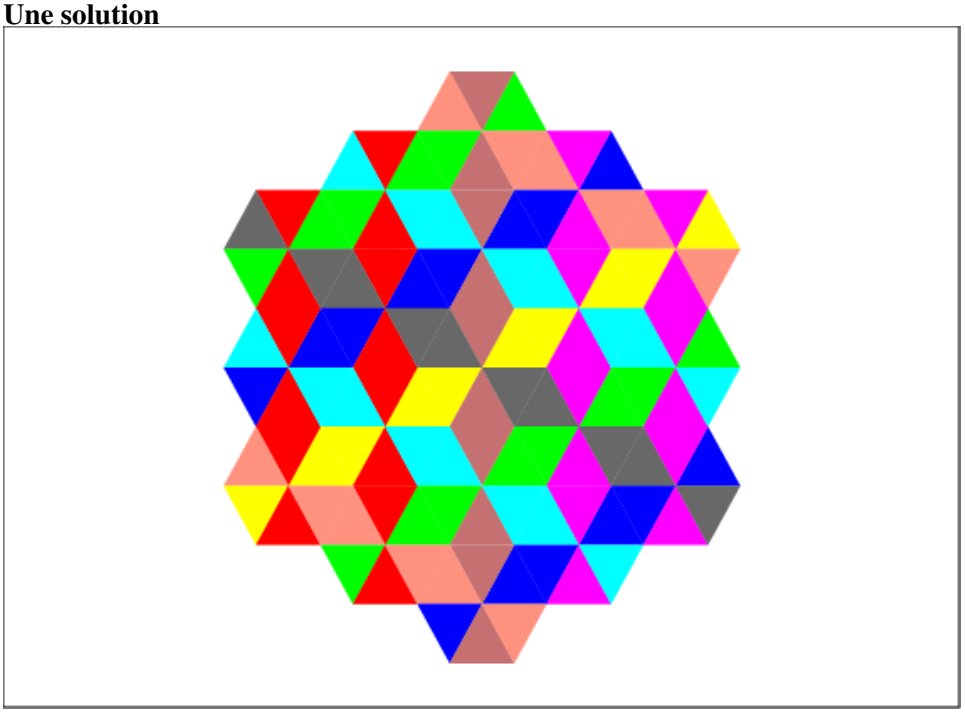


La variante2

On colore les 6 triangles équilatéraux de l'hexagone, les 27 pièces sont alors :



Une solution



8.12.3 Les programmes

Les programmes en géométrie `coul(c)` renvoie les couleurs de code 1,2,3,4,5,6,c,8,9.
`cg(c,r)` renvoie les centres des hexagones de côtés `r` de la grille de centre `c`.
`lg(c,r)` affiche le centre des hexagones de la grille avec comme légende un numéro.
`grille(c,r)` renvoie la grille de centre `c` constituée de 19 hexagones de côtés `r`.
`puzzle(c,r,c1,c2,c3,c4,c5,c6,c7,c8,c9)` renvoie la solution avec comme couleur $c_k = k$ si $k! = 7$ et $c_7 = 77$ ou avec d'autres couleurs. On tape dans un niveau de programation :

```
coul(c):={
local L,j;
L:=NULL;
pour j de 1 jusque 9 faire L:=L,affichage(carre(j,j+1),j+rempli);
L:=L,legende(j+(3+4i)/8,j);
```

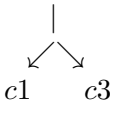
```
fpour;
L:=L,affichage(carre(7,8),c+rempli);
L:=L,legende(7+(3+4i)/8,c);
retourne L;
}:;
cg(c,r):={
  local L,h,j;
  h:=r*sqrt(3)/2;
  L:=point(c);
  pour j de 0 jusque 5 faire
 L:=L,point(c+2h*exp(i*(pi/6+j*pi/3)));
fpour;
pour j de 0 jusque 5 faire
 L:=L,point(c+4h*exp(i*(pi/6+j*pi/3)));
 L:=L,point(c+3r*exp(i*(pi/3+j*pi/3)));
fpour;
retourne L;
}:;
lg(c,r):={
  local L,h,j,A;
  h:=r*sqrt(3)/2;
  L:=legende(point(c),"0");
  pour j de 1 jusque 18 faire
 L:=L,legende(cg1(c,r)[j],j);
 fpour;
retourne [L];
}:;
grille(c,r):={
local L,h,j;
h:=r*sqrt(3)/2;
L:=isopolygone(c,point(c+r),-6);
pour j de 1 jusque 18 faire
 L:=L,isopolygone(cg(c,r)[j],cg(c,r)[j]+r,-6);
fpour;
retourne L;
}:;
solpuzzle1(c,r,c1,c2,c3,c4,c5,c6,c7,c8,c9):={
local L;
L:=NULL;
L:=L,piece(c+cg1(0,1)[0],r,c3,c9,c8);
L:=L,piece(c+cg1(0,1)[2],r,c4,c9,c6);
L:=L,piece(c+cg1(0,1)[9],r,c2,c9,c7);
L:=L,piece(c+cg1(0,1)[5],r,c2,c9,c6);
L:=L,piece(c+cg1(0,1)[15],r,c4,c9,c7);
L:=L,piece(c+cg1(0,1)[10],r,c2,c1,c6);
L:=L,piece(c+cg1(0,1)[3],r,c4,c1,c8);
L:=L,piece(c+cg1(0,1)[4],r,c3,c1,c6);
L:=L,piece(c+cg1(0,1)[14],r,c2,c1,c7);
```

```
L:=L,piece(c+cg1(0,1)[11],r,c2,c1,c8);
L:=L,piece(c+cg1(0,1)[12],r,c4,c1,c6);
L:=L,piece(c+cg1(0,1)[13],r,c3,c1,c7);
L:=L,piece(c+cg1(0,1)[8],r,c4,c5,c7);
L:=L,piece(c+cg1(0,1)[1],r,c3,c5,c6);
L:=L,piece(c+cg1(0,1)[6],r,c2,c5,c8);
L:=L,piece(c+cg1(0,1)[16],r,c4,c5,c6);
L:=L,piece(c+cg1(0,1)[7],r,c3,c5,c7);
L:=L,piece(c+cg1(0,1)[18],r,c2,c5,c6);
L:=L,piece(c+cg1(0,1)[17],r,c4,c5,c8);
retourne L;
}:;
solpuzzle2(c,r,c1,c2,c3,c4,c5,c6,c7,c8,c9):={
local L;
L:=NULL;
L:=L,piece(c+cg1(0,1)[0],r,c3,c9,c8);
L:=L,piece(c+cg1(0,1)[2],r,c4,c9,c6);
L:=L,piece(c+cg1(0,1)[9],r,c2,c9,c7);
L:=L,piece(c+cg1(0,1)[5],r,c4,c9,c7);
L:=L,piece(c+cg1(0,1)[15],r,c2,c9,c6);
L:=L,piece(c+cg1(0,1)[10],r,c2,c1,c6);
L:=L,piece(c+cg1(0,1)[3],r,c4,c1,c8);
L:=L,piece(c+cg1(0,1)[4],r,c3,c1,c7);
L:=L,piece(c+cg1(0,1)[14],r,c4,c1,c6);
L:=L,piece(c+cg1(0,1)[11],r,c2,c1,c8);
L:=L,piece(c+cg1(0,1)[12],r,c4,c1,c7);
L:=L,piece(c+cg1(0,1)[13],r,c3,c1,c6);
L:=L,piece(c+cg1(0,1)[8],r,c4,c5,c7);
L:=L,piece(c+cg1(0,1)[1],r,c3,c5,c6);
L:=L,piece(c+cg1(0,1)[6],r,c4,c5,c8);
L:=L,piece(c+cg1(0,1)[16],r,c2,c5,c7);
L:=L,piece(c+cg1(0,1)[7],r,c3,c5,c7);
L:=L,piece(c+cg1(0,1)[18],r,c4,c5,c6);
L:=L,piece(c+cg1(0,1)[17],r,c2,c5,c8);
retourne L;
}:;

solpuzzle3(c,r,c1,c2,c3,c4,c5,c6,c7,c8,c9):={
local L,R,L1;
L1:=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18];
L1:=[0,6,1,2,3,4,5,17,18,7,8,9,10,11,12,13,14,15,16];
R(x):=L1[x];
L:=NULL;
L:=L,piece(c+cg1(0,1)[R(0)],r,c9,c8,c3);
L:=L,piece(c+cg1(0,1)[R(2)],r,c9,c6,c4);
L:=L,piece(c+cg1(0,1)[R(9)],r,c9,c7,c2);
L:=L,piece(c+cg1(0,1)[R(5)],r,c9,c6,c2);
L:=L,piece(c+cg1(0,1)[R(15)],r,c9,c7,c4);
```

```
L:=L,piece(c+cg1(0,1)[R(10)],r,c1,c6,c2);
L:=L,piece(c+cg1(0,1)[R(3)],r,c1,c8,c4);
L:=L,piece(c+cg1(0,1)[R(4)],r,c1,c6,c3);
L:=L,piece(c+cg1(0,1)[R(14)],r,c1,c7,c2);
L:=L,piece(c+cg1(0,1)[R(11)],r,c1,c8,c2);
L:=L,piece(c+cg1(0,1)[R(12)],r,c1,c6,c4);
L:=L,piece(c+cg1(0,1)[R(13)],r,c1,c7,c3);
L:=L,piece(c+cg1(0,1)[R(8)],r,c5,c7,c4);
L:=L,piece(c+cg1(0,1)[R(1)],r,c5,c6,c3);
L:=L,piece(c+cg1(0,1)[R(6)],r,c5,c8,c2);
L:=L,piece(c+cg1(0,1)[R(16)],r,c5,c6,c4);
L:=L,piece(c+cg1(0,1)[R(7)],r,c5,c7,c3);
L:=L,piece(c+cg1(0,1)[R(18)],r,c5,c6,c2);
L:=L,piece(c+cg1(0,1)[R(17)],r,c5,c8,c4);
retourne L;
}:;
solpuzzle4(c,r,c1,c2,c3,c4,c5,c6,c7,c8,c9):={
local L,R,L1;
L1:=[0,6,1,2,3,4,5,17,18,7,8,9,10,11,12,13,14,15,16];
R(x):=L1[x];
L:=NULL;
L:=L,piece(c+cg1(0,1)[(R@@5)(0)],r,c8,c3,c9);
L:=L,piece(c+cg1(0,1)[(R@@5)(2)],r,c6,c4,c9);
L:=L,piece(c+cg1(0,1)[(R@@5)(9)],r,c7,c2,c9);
L:=L,piece(c+cg1(0,1)[(R@@5)(5)],r,c6,c2,c9);
L:=L,piece(c+cg1(0,1)[(R@@5)(15)],r,c7,c4,c9);
L:=L,piece(c+cg1(0,1)[(R@@5)(10)],r,c6,c2,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(3)],r,c8,c4,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(4)],r,c6,c3,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(14)],r,c7,c2,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(11)],r,c8,c2,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(12)],r,c6,c4,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(13)],r,c7,c3,c1);
L:=L,piece(c+cg1(0,1)[(R@@5)(8)],r,c7,c4,c5);
L:=L,piece(c+cg1(0,1)[(R@@5)(1)],r,c6,c3,c5);
L:=L,piece(c+cg1(0,1)[(R@@5)(6)],r,c8,c2,c5);
L:=L,piece(c+cg1(0,1)[(R@@5)(16)],r,c6,c4,c5);
L:=L,piece(c+cg1(0,1)[(R@@5)(7)],r,c7,c3,c5);
L:=L,piece(c+cg1(0,1)[(R@@5)(18)],r,c6,c2,c5);
L:=L,piece(c+cg1(0,1)[(R@@5)(17)],r,c8,c4,c5);
retourne L;
}:;
```

`piece(c,r,c1,c2,c3)` dessine un hexagone de centre d'affixe `c` ayant un sommet d'affixe `c+r` et de couleur `c1,c2,c3` : $c2$



```
piece(c,r,c1,c2,c3):={
  local L,h;
  h:=r*sqrt(3)/2;
  L:=isopolygone(c,point(c+r),-6);
  L:=L,segment(c+h*exp(i*pi/6),c+h*exp(7*i*pi/6),
               affichage=c1+epaisseur_ligne_7);
  L:=L,segment(c+h*exp(i*pi/2),c+h*exp(3*i*pi/2),
               affichage=c2+epaisseur_ligne_7);
  L:=L,segment(c+h*exp(-i*pi/6),c+h*exp(5*i*pi/6),
               affichage=c3+epaisseur_ligne_7);
  L:=L,cercle(c,r/10,affichage=7+rempli);
    retourne L;
  }:;
```

`piece1(c,r,c1,c2,c3)` est utilisée pour l'analyse du problème.

```
piece1(c,r,c1,c2,c3):={
  local L,h;
  h:=r*sqrt(3)/2;
      L:=segment(c+7*h*exp(i*pi/6),c+7*h*exp(7*i*pi/6),
               affichage=c1+epaisseur_ligne_7);
    L:=L,segment(c+7*h*exp(i*pi/2),c+7*h*exp(3*i*pi/2),
               affichage=c2+epaisseur_ligne_7);
    L:=L,segment(c+7*h*exp(-i*pi/6),c+7*h*exp(5*i*pi/6),
               affichage=c3+epaisseur_ligne_7);
    L:=L,isopolygone(c,point(c+r),-6,affichage=epaisseur_ligne_7);
    retourne L;
  }:;
```

Les piéces des variantes :

```
piecec(c,r,c1,c2,c3):={
  local L,j;
   L:=isopolygone(c,point(c+r),-6);
   L:=L,segment(c+r,c+r*exp(i*pi/3),
                affichage=c1+epaisseur_ligne_7);
   L:=L,segment(c+r*exp(i*pi/3),c+r*exp(2*i*pi/3),
                affichage=c2+epaisseur_ligne_7);
   L:=L,segment(c+r*exp(2*i*pi/3),c+r*exp(3*i*pi/3),
                affichage=c3+epaisseur_ligne_7);
   L:=L,segment(c+r*exp(3*i*pi/3),c+r*exp(4*i*pi/3),
                affichage=c1+epaisseur_ligne_7);
   L:=L,segment(c+r*exp(4*i*pi/3),c+r*exp(5*i*pi/3),
                affichage=c2+epaisseur_ligne_7);
   L:=L,segment(c+r*exp(5*i*pi/3),c+r*exp(6*i*pi/3),
                affichage=c3+epaisseur_ligne_7);
   retourne L;
   }:;
piecep(c,r,c1,c2,c3):={
  local L;
```

```
   //L:=isopolygone(c,point(c+r),-6);
   L:=L,affichage(triangle_equilateral(c+r,c+r*exp(i*pi/3)),
                  c1+rempli);
   L:=L,affichage(triangle_equilateral(c+r*exp(i*pi/3),
                  c+r*exp(2*i*pi/3)),c2+rempli);
   L:=L,affichage(triangle_equilateral(c+r*exp(2*i*pi/3),
                  c+r*exp(3*i*pi/3)),c3+rempli);
   L:=L,affichage(triangle_equilateral(c+r*exp(3*i*pi/3),
                  c+r*exp(4*i*pi/3)),c1+rempli);
   L:=L,affichage(triangle_equilateral(c+r*exp(4*i*pi/3),
                  c+r*exp(5*i*pi/3)),c2+rempli);
   L:=L,affichage(triangle_equilateral(c+r*exp(5*i*pi/3),
                  c+r*exp(6*i*pi/3)),c3+rempli);
   retourne L;
}:;
```

Dans un niveau de géométrie, on tape pour avoir les 27 piéces :

```
piece(-6,1,2,9,6), piece(-6+2i,1,2,5,6), piece(-6+4i,1,2,1,6);
piece(-4,1,3,9,6), piece(-4+2i,1,3,5,6), piece(-4+4i,1,3,1,6);
piece(-2,1,4,9,6), piece(-2+2i,1,4,5,6), piece(-2+4i,1,4,1,6);
piece(0,1,2,9,77), piece(2i,1,2,5,77), piece(4i,1,2,1,77);
piece(2,1,3,9,77), piece(2+2i,1,3,5,77), piece(2+4i,1,3,1,77);
piece(4,1,4,9,77), piece(4+2i,1,4,5,77), piece(4+4i,1,4,1,77);
piece(6,1,2,9,8), piece(6+2i,1,2,5,8), piece(6+4i,1,2,1,8);
piece(8,1,3,9,8), piece(8+2i,1,3,5,8), piece(8+4i,1,3,1,8);
piece(10,1,4,9,8), piece(10+2i,1,4,5,8), piece(10+4i,1,4,1,8);
```

On obtient les 27 piéces.

Dans un niveau de géométrie, on tape pour avoir une solution :

```
//grille(0,1);lg(0,1)[0];
piece(lg(0,1)[1,0],1,3,9,8);
piece(lg(0,1)[1,2],1,4,9,6);
piece(lg(0,1)[1,9],1,2,9,77);
piece(lg(0,1)[1,5],1,2,9,6);
piece(lg(0,1)[1,15],1,4,9,77);
piece(lg(0,1)[1,10],1,2,1,6);
piece(lg(0,1)[1,3],1,4,1,8);
piece(lg(0,1)[1,4],1,3,1,6);
piece(lg(0,1)[1,14],1,2,1,77);
piece(lg(0,1)[1,11],1,2,1,8);
piece(lg(0,1)[1,12],1,4,1,6);
piece(lg(0,1)[1,13],1,3,1,77);
piece(lg(0,1)[1,8],1,4,5,77);
piece(lg(0,1)[1,1],1,3,5,6);
piece(lg(0,1)[1,6],1,2,5,8);
piece(lg(0,1)[1,16],1,4,5,6);
```

```
piece(lg(0,1)[1,7],1,3,5,77);
piece(lg(0,1)[1,18],1,2,5,6);
piece(lg(0,1)[1,17],1,4,5,8);
```

On obtient une solution.
On tape pour connaitre les 8 pièces non utilisées :

```
piece(-6,1,3,1,8);piece(-4,1,4,1,77);
piece(-2,1,3,5,8);piece(0,1,2,5,77);
piece(2,1,3,9,6);piece(4,1,3,9,77);
piece(6,1,2,9,8);piece(8,1,4,9,8);
```

8.12.4 Les programmes des 2 variantes

En remplacant `piece` par `piecec` on obtient les piéces de la 1ière variante :

```
piecec(-6,1,2,9,6), piecec(-6+2i,1,2,5,6), piecec(-6+4i,1,2,1,6);
piecec(-4,1,3,9,6), piecec(-4+2i,1,3,5,6), piecec(-4+4i,1,3,1,6);
piecec(-2,1,4,9,6), piecec(-2+2i,1,4,5,6), piecec(-2+4i,1,4,1,6);
piecec(0,1,2,9,77), piecec(2i,1,2,5,77), piecec(4i,1,2,1,77);
piecec(2,1,3,9,77), piecec(2+2i,1,3,5,77), piecec(2+4i,1,3,1,77);
piecec(4,1,4,9,77), piecec(4+2i,1,4,5,77), piecec(4+4i,1,4,1,77);
piecec(6,1,2,9,8), piecec(6+2i,1,2,5,8), piecec(6+4i,1,2,1,8);
piecec(8,1,3,9,8), piecec(8+2i,1,3,5,8), piecec(8+4i,1,3,1,8);
piecec(10,1,4,9,8), piecec(10+2i,1,4,5,8), piecec(10+4i,1,4,1,8);
```

Une solution de la 1ière variante :

```
//grille(0,1);lg(0,1)[0];
piecec(lg(0,1)[1,0],1,3,9,8);
piecec(lg(0,1)[1,2],1,4,9,6);
piecec(lg(0,1)[1,9],1,2,9,77);
piecec(lg(0,1)[1,5],1,2,9,6);
piecec(lg(0,1)[1,15],1,4,9,77);
piecec(lg(0,1)[1,10],1,2,1,6);
piecec(lg(0,1)[1,3],1,4,1,8);
piecec(lg(0,1)[1,4],1,3,1,6);
piecec(lg(0,1)[1,14],1,2,1,77);
piecec(lg(0,1)[1,11],1,2,1,8);
piecec(lg(0,1)[1,12],1,4,1,6);
piecec(lg(0,1)[1,13],1,3,1,77);
piecec(lg(0,1)[1,8],1,4,5,77);
piecec(lg(0,1)[1,1],1,3,5,6);
piecec(lg(0,1)[1,6],1,2,5,8);
piecec(lg(0,1)[1,16],1,4,5,6);
piecec(lg(0,1)[1,7],1,3,5,77);
piecec(lg(0,1)[1,18],1,2,5,6);
piecec(lg(0,1)[1,17],1,4,5,8);
```

En remplacant `piece` par `piecep` on obtient les piéces de la 2ième variante :

```
piecep(-6,1,2,9,6), piecep(-6+2i,1,2,5,6), piecep(-6+4i,1,2,1,6);
piecep(-4,1,3,9,6), piecep(-4+2i,1,3,5,6), piecep(-4+4i,1,3,1,6);
piecep(-2,1,4,9,6), piecep(-2+2i,1,4,5,6), piecep(-2+4i,1,4,1,6);
piecep(0,1,2,9,77), piecep(2i,1,2,5,77), piecep(4i,1,2,1,77);
piecep(2,1,3,9,77), piecep(2+2i,1,3,5,77), piecep(2+4i,1,3,1,77);
piecep(4,1,4,9,77), piecep(4+2i,1,4,5,77), piecep(4+4i,1,4,1,77);
piecep(6,1,2,9,8), piecep(6+2i,1,2,5,8), piecep(6+4i,1,2,1,8);
piecep(8,1,3,9,8), piecep(8+2i,1,3,5,8), piecep(8+4i,1,3,1,8);
piecep(10,1,4,9,8), piecep(10+2i,1,4,5,8), piecep(10+4i,1,4,1,8);
```

Une solution de la 2ième variante :

```
//grille(0,1);lg(0,1)[0];
piecep(lg(0,1)[1,0],1,3,9,8);
piecep(lg(0,1)[1,2],1,4,9,6);
piecep(lg(0,1)[1,9],1,2,9,77);
piecep(lg(0,1)[1,5],1,2,9,6);
piecep(lg(0,1)[1,15],1,4,9,77);
piecep(lg(0,1)[1,10],1,2,1,6);
piecep(lg(0,1)[1,3],1,4,1,8);
piecep(lg(0,1)[1,4],1,3,1,6);
piecep(lg(0,1)[1,14],1,2,1,77);
piecep(lg(0,1)[1,11],1,2,1,8);
piecep(lg(0,1)[1,12],1,4,1,6);
piecep(lg(0,1)[1,13],1,3,1,77);
piecep(lg(0,1)[1,8],1,4,5,77);
piecep(lg(0,1)[1,1],1,3,5,6);
piecep(lg(0,1)[1,6],1,2,5,8);
piecep(lg(0,1)[1,16],1,4,5,6);
piecep(lg(0,1)[1,7],1,3,5,77);
piecep(lg(0,1)[1,18],1,2,5,6);
piecep(lg(0,1)[1,17],1,4,5,8);
```

Les différents exemples de solutions sont obtenues avec :

```
solpuzzle1(0,1,1,2,3,4,5,6,77,8,9);solpuzzle2(8,1,1,2,3,4,5,6,77,8,9)
solpuzzle1(16,1,5,2,3,4,1,6,77,8,9);solpuzzle2(24,1,5,2,3,4,1,6,77,8,
solpuzzle1(0,1,2,6,8,77,4,1,5,9,3);solpuzzle2(8,1,2,6,8,77,4,1,5,9,3)
solpuzzle1(16,1,4,6,8,77,2,1,5,9,3);solpuzzle2(24,1,4,6,8,77,2,1,5,9,
solpuzzle1(0,1,6,1,9,5,77,2,4,3,8);solpuzzle2(8,1,6,1,9,5,77,2,4,3,8)
solpuzzle1(16,1,77,1,9,5,6,2,4,3,8);solpuzzle2(24,1,77,1,9,5,6,2,4,3,
solpuzzle3(-8,1,2,6,8,77,4,1,5,9,3),solpuzzle3(0,1,2,6,8,77,4,5,1,9,3
solpuzzle4(8,1,6,1,9,5,77,2,4,3,8),solpuzzle4(16,1,6,5,9,1,77,2,4,3,8
```

8.12.5 Les programmes en Logo de la 2ième variante

```
piece(c3,c2,c1):={
  repete(2,crayon c3,triangle_plein(20,20,60),tourne_gauche 60,crayon
      triangle_plein(20,20,60),tourne_gauche 60,crayon c1,
      triangle_plein(20,20,60),tourne_gauche 60);
}:;
```

```
puzzle():={
 efface;
 piece(3,9,8);tourne_gauche 30;saute 17.32*2;tourne_droite 30;
 piece(3,5,6);tourne_gauche 30;saute 17.32*2;tourne_droite 30;
 piece(3,5,77);saute -60;
 piece(4,9,6);tourne_gauche 30;saute 17.32*2;tourne_droite 30;
 piece(4,5,77);saute -60;
 piece(2,1,6);tourne_gauche 30;saute 17.32*2;tourne_droite 30;
 piece(2,9,77);tourne_gauche 30;saute -17.32*4;tourne_droite 30;
 piece(2,1,8);pas_de_cote -17.32*2;
 piece(4,1,6);pas_de_cote -17.32*2;
 piece(3,1,77);tourne_droite 30;saute 17.32*2;tourne_gauche 30;
 piece(2,1,77);tourne_droite 30;saute 17.32*2;tourne_gauche 30;
 piece(4,9,77);tourne_gauche 30;saute 17.32*2;tourne_droite 30;
 piece(4,5,6);tourne_gauche  30;saute 17.32*2;tourne_droite 30;
 piece(4,5,8);pas_de_cote 17.32*2;
 piece(2,5,6);tourne_gauche 30;saute -17.32*2;tourne_droite 30;
 piece(2,5,8);tourne_gauche 30;saute -17.32*2;tourne_droite 30;
 piece(2,9,6);tourne_droite 30;saute -17.32*2;tourne_gauche 30;
 piece(3,1,6);pas_de_cote 17.32*2;
 piece(4,1,8);tourne_droite 30;saute 17.32*2;tourne_gauche 30;
}:;
```

Puis dans un niveau de dessin, on tape :
`efface`
`puzzle()`
Pour obtenir une solution de la 2ième variante.

8.13 Le nombre de voyageurs et les suites

7 *Un train pouvant contenir au plus 1000 personnes part de A avec x voyageurs jusqu'à son terminus B. Il effectue 6 arrets intermédiaires et à chaque arrêt un tiers des voyageurs descendent et 15 personnes montent. À l'arrivée il reste y voyageurs avec y différent de x. Calculer x et y.*
Soit u_n le nombre de voyageurs au départ de l'arret n (n=0..6).
On tape : `seqsolve(x*2/3+15,[x,n],a)`
On obtient : `45+(a-45)*(2/3)^n`
Ou on tape : `rsolve(u(n+1)=2*u(n)/3+15,u(n),u(0)=a)`
On obtient : `[a*(2/3)^n-45*(2/3)^n+45]`
On en déduit que 3^6 divise $a-45$ i.e. $a-45=3^6*k\ k\in\mathbb{N}^*$
On tape : `solve(a-45=3^6,a)`
On obtient la valeur de a : `[774]`
Donc si 3^6 divise $a-45$ et si $a<1000$ on a $a=774$ On tape : `u(n):=45+(774-45)*(2/3)^n`
On tape : `u(n)$(n=0..6)`
On obtient le nombre de voyageurs selon les arrêts :
`774,531,369,261,189,141,109`
Le nombre de voyageurs à l'arrivée est égal au nombre de voyageurs au départ du

6ième arret.
Donc il y avait 774 voyageurs au départ et 109 voyageurs à l'arrivée.

8.14 La date de péremption

8 *Un épicier achète des produits dans un pays où la date de péremption est notée par un nombre entier ayant un grand nombre de chiffres. La consigne pour avoir l'année est de découper ce nombre en tranches de 2 chiffres à partir de la droite et d'en faire la somme. La consigne pour avoir le mois est de faire la somme alternée des chiffres en partant de la droite (le chiffre des unités -le chiffre des dizaines+ le chiffre des centaines....).L'épicier a trouver 2014 pour l'année mais se trompe en faisant le deuxième calcul. Pouvez-vous l'aider à trouver le mois ? Pouvez-vous l'aider à trouver l'année et le mois plus facilement sachant que la date de péremption d'un produit est égale à sa date de fabrication + k avec $k < 120$ mois ?*
On a :
$N = a_0 + a_1 * 10 + a_2 * 10^2 + .. + a_n 10^n$ avec $0 \leq a_k < 10$ pour $k = 1..n$
Soit $p = iquo(n+1, 2) + irem(n+1, 2) - 1$ $N = b0 + b1 * 100 + b_2 * 100^2 + ...b_p * 100^p$ avec $0 \leq b_k < 100$ pour $k = 1..p$
par exemple si $N = 876543210$ alors $a_k = k$ et $n = 8$ et
$b_0 = 10$, $b_1 = 32$, $b_2 = 54$, $b_3 = 76$, $b_4 = 8$ et $p = 4 + 1 - 1 = 4$ On a :
$10^k \% 11 = (-1)^k \% 11$ et
$100^k \% 11 = 1 \% 11$
Soit $A = \sum_{k=0}^{n} a_k$ et $M = \sum_{k=0}^{p} (-1)^k b_k$
On a donc :
$A = N \% 11$ et $M = N \% 11$ donc $M = A \% 11$
Donc si $A = 2014$ on a $M = 4-1+0-2 = 1$ c'est donc le mois de janvier ... Pour calculer A et M il suffit de calculer $M = N \% 11$ et $A = ((182+k)*11+M)\$(k = 0..3) = (2003, 2014, 2025, 2036)$

8.15 Trouver le $n$ième terme d'une suite récurrente

8.15.1 Un exemple

Soit f une fonction, on définit la suite u_n par u_0 et par $u_n = f(u_{n-1})$ pour $n > 0$. Par exemple $f(x) = -7x/10 + 2$ et $u_0 = 3$.
On veut calculer les premiers termes de cette suite.
On tape :

```
f(x):=-7x/10+2
```

puis on tape :

```
3
```

puis on tape

```
f(ans())
```

et on obtient `41/10`, puis on tape

```
f(ans())
```

et on obtient `487/100`, etc....

8.15.2 Interêts d'un placement

On place 10 000 euros à un taux annuel de 4%. Quelle et la somme que l'on obtient au bout de 5 ans ? au bout de 10 ans ?

Une somme S devient $S*(1+4/100)$ au bout de 1 an,
Une somme S devient $S*(1+4/100)^2$ au bout de 2 ans,
...
Une somme S devient $S*(1+4/100)^5$ au bout de 5 ans,
Une somme S devient $S*(1+4/100)^{10}$ au bout de 10 ans,
On tape : `evalf(10000*(1+4/100)^5)`
On obtient `12166.529024`
On tape : `evalf(10000*(1+4/100)^10)`
On obtient `14802.4428492`

8.15.3 Mensualités d'un emprunt

On veut emprunter $S = 10000$ euros sur 5 ans (respectivement 60 mois) à un taux annuel de 4% (respectivement à un mensuel de 0.33 %). Combien doit-on rembourser annuellement (mensuellement) ?
Sur 5 ans, on fait 5 remboursements annuels égaux à r.
Au bout de 1 an, la somme S devient :
$S*(1+4/100)-r$,
Au bout de 2 ans, la somme S devient :
$(S*(1+4/100)-r)*(1+4/100)-r = S*(1+4/100)^2-r*(1+4/100)-r$,
Au bout de 3 ans, la somme S devient :
$(S*(1+4/100)^2-r*(1+4/100)-r)*(1+4/100)-r =$
$S*(1+4/100)^3-r*(1+4/100)^2-r*(1+4/100)-r$,
...
Au bout de 5 ans, la somme S devient :
$S*(1+4/100)^5-r*((1+4/100)^4+(1+4/100)^3+(1+4/100)^2+(1+4/100)+1)$
i.e.
$S*(1+4/100)^5-r*((1+4/100)^5-1)/(4/100)$,
Si on veut avoir tout remboursé au bout de 5 ans, il faut résoudre :
$S*(1+4/100)^5-r*((1+4/100)^5-1)/4*100=0$.
On tape :
`evalf(solve(10000*(1+4/100)^5-r*((1+4/100)^5-1)/4*100,r))`
On obtient `[2246.27113493]`
On aura remboursé en tout : `2246.27*5=11231.35` euros.

On vous propose de rembourser mensuellement 187 euros par mois.
Est-ce correct ?

On cherche le taux mensuel t de cet emprunt.
Au bout de 1 mois, la somme S devient $S*(1+t)-r$,
Au bout de 2 mois, la somme S devient $(S*(1+t)^2-r*(1+t)-r$,
.....
Au bout de 60 mois, la somme S devient :
$(S(1+t)^{60}-r*((1+t)^{59}...(1+t)+1)$ soit $S(1+t)^{60}-r*((1+t)^{60}-1)/t$,
Si on veut avoir tout remboursé au bout de 60 mois, il faut résoudre :
$10000(1+t)^{60}-187*((1+t)^{60}-1)/t=0$.

On tape :
`fsolve(10000*(1+t)^60-187*((1+t)^60-1)/t,t=0.0033)`
On obtient : `0.00385432769081`
Si on fait 60 versements de $r = 187$ euros, on aura payer en tout `187*60=11220` euros donc un peu moins que précédemment...mais le taux annuel est plus élevé ($0.00385432769081 * 12 \simeq 4.62/100$). On est donc perdant....

8.16 Solution approchée de $f(x) = 0$ sur $[a; b]$

On suppose que f est continue et que $f(a) * f(b) < 0$et donc f s'annule sur $[a; b]$. On va procéder par dichotomie : on partage l'intervalle $[a; b]$ en deux :

$$[a, b] = [a; m = (a + b)/2] \cup [m = (a + b)/2; b]$$

et on regarde si $f(a) * f(m) < 0$ si c'est le cas on cherche la solution dans $[a; m]$ sinon on cherche la solution dans $[m; b]$. On recommence autant de fois qu'il faut pour avoir la précision eps désirée.
On tape :

```
Zero(f,a,b,eps):={
  local m;
  si f(a)==0 alors retourne a fsi;
  si f(b)==0 alors retourne b fsi;
  si f(a)*f(b)>0 alors retourne "erreur" fsi;
  tantque (abs(a-b)>eps) faire
    m:=(a+b)/2;
    si (f(a)*f(m)<0) alors
      b:=m;
    sinon
      a:=m;
    fsi;
  ftantque;
  si (a<b) alors
    retourne evalf(a,-expr(mid(""+eps,2))),evalf(b,-expr(mid(""+eps,2)
  fsi;
  retourne evalf(b,-round(log10(eps))),evalf(a,-round(log10(eps)));

}
:;
```

Trouver un encadrement à $1e-9$ prés puis a $1e-20$ près d'une solution de $x^2 - 2 = 0$ sur $[0; 2]$.
On tape :
`f(x):=x^2-2;`
`Zero(f,0,2,1e-9)`
On obtient l'encadrement : `[1.41421356145,1.41421356238]` On tape :
`Zero(f,0,2,1e-20)`
On obtient l'encadrement : `[1.41421356237309504879,1.41421356237309504880]`
On tape :

`evalf(sqrt(2),20)`
On obtient : `1.41421356237309504880`

8.17 Codage de messages

8.17.1 Les caractères utilisables

Les caractères utilisables dans `Xcas` ont un code ASCII qui va de 32 (l'espace) à 126 (le tilde), puis il y a des caractères spéciaux et les lettres accentuées qui ont un code ASCII qui va de 161 (le point d'exclamation à l'envers) à 255 (le bécarre). Avec `Xcas`, on utilise les commandes :

- `asc` qui a comme argument une chaîne de caractères renvoie la liste des codes ASCII. Par exemple `asc("AH@AH")=[65,72,64,65,72]`
- `char` qui a une liste de nombres renvoie la chaîne de caractères de code ASCII les nombres modulo 256. Par exemple :
  `char([65+256,72,64,65,328+256])= "AH@AH"`

8.17.2 Codage par addition

On peut convenir dans un premier temps de coder le message avec 27 caractères qui sont @, A,B,...Z et qui ont un code ASCII entre 64 et 90. On utilisera le caractère @ pour séparer les mots. Pour coder par addition, on choisit une clé secréte qui est ici un nombre c entre 0 et (126-90=36). Pour coder un message on remplace la lettre de code n par la lettre dont le code est $n+c$. Ainsi si la clé vaut 7, le "A" qui a comme code 65 sera remplacé par la lettre de code 65+7=72 c'est à dire par le "H" et si la clé vaut 27, le "A" qui a comme code 65 sera remplacé par la lettre de code 65+27=92 c'est à dire par le "\". Dans ce cas le message codé utilise les caractères dont les codes vont de $64+c$ à $90+c<127$.
Pour décoder il suffit de remplacer c par $-c$.
On tape :

```
Codadd(L,cle):={
 local s,j;
 s:=size(L)
 retourne char(asc(L)+ makelist(cle,1,s));
}:;
```

On tape : `M1:=Codadd("MESSAGE@SECRET@POUR@ZORRO",7)`
On obtient :
`"TLZZHNLGZLJYL[GWV\YGaVYYV"`
On tape : `Codadd(M1,-7)`
On obtient : `="MESSAGE@SECRET@POUR@ZORRO"`
On tape : `M2:=Codadd("MESSAGE@SECRET@POUR@ZORRO",27)`
On obtient :
`""h`nn\b`[n`^m`o[kjpm[ujmmj""`
On tape : `Codadd(M2,-27)`
On obtient : `="MESSAGE@SECRET@POUR@ZORRO"`
On peut dans un deuxième temps vouloir coder le message uniquement avec des lettres de code compris entre 64 et 90 (pour que ce soit lisible). Pour cela on ramène

les codes entre 0 et 26 (en enlevant 64) puis on ajoute la clé, puis on ramène les nombres obtenus entre 0 et 26 en les remplacant par le reste de la division par 27 et enfin ou rajoute 64 pour avoir un code entre 64 et 90.
On tape :

```
Codaddi(L,cle):={
 local s,j,M;
 s:=size(L)
 M:=asc(L)+ makelist(-64,1,s);
 pour j de 0 jusque s-1 faire
  M[j]:=irem(M[j]+cle,27)+64;
 fpour
 retourne char(M);
}:;
```

On tape : `M2:=Codaddi(="MESSAGE@SECRET@POUR@ZORRO",7)`
On obtient :
`"TLZZHNLGZLJYL@GWVAYGFVYYV"`
On tape : `Codadd(M3,-7)`
On obtient : `="MESSAGE@SECRET@POUR@ZORRO"`

8.17.3 Codage par multiplication

On suppose que l'on n'utilise que les 27 caractères "A".."Z" et le caractère @ comme espace.
On tape : `asc("@ABZ")`
On obtient ; `[64,65,66,90]` Ainsi en enlevant 64 à ces codes on a des nombres entre 0 et 26.
On multiplie ces codes par le clé qui doit être un nombre inversible de $Z/27Z$ par exemple si la clé vaut 7, on a $7 * 4 = 28 = 1 \mod 27$.
Les nombres inversibles de $Z/27Z$ sont les nombres qui ne sont pas divisible par 3. Pour avoir l'inverse d'un tel nombre il faut faire faire à `Xcas` les calculs dans $Z/27Z$.
On tape : `1/7 % 27`
On obtient `4 % 27` ($4 * 7 = 27 + 1 = 1 \mod 27$)
On tape :`1/17 % 27`
On obtient `8 % 27` ($8 * 17 = 5 * 27 + 1 = 1 \mod 27$)
C'est l'inverse de la clé que l'on utilisera pour le décodage.
On tape :

```
Codmult(L,cle):={
 local s,M,j;
 s:=size(L);
 M:=asc(L)+ makelist(-64,1,s);
 pour j de 0 jusque s-1 faire
  M[j]:=irem(M[j]*cle,27)+64;
 fpour;
 retourne char(M);
}:;
```

On tape : `M4:=Codmult("MESSAGE@SECRET@POUR@ZORRO", 7)`
On obtient : `"JHYYGVH@YHURHE@DXLR@TXRRX"`
On tape : `Codmult(M4,4)`
On obtient : `"MESSAGE@SECRET@POUR@ZORRO"`
On remarque que le caractère @ reste inchangé car son code est ramené à 0 puis reste 0 apres la multiplication..... Cela n'est pas forcément un probleme puisqu'il code l'espace mais on peut connaitre la longueur des mots.
On peut alors modifier le programme en enlevant 63 (pour éviter d'avoir 0) et en prenant le reste de la division par 28. Dans ce cas, c'est " ?" de code 63 qui ne sera pas changé par le codage.
On tape :

```
Codmulti(L,cle):={
 local s,M,j;
 s:=size(L);
 M:=asc(L)+ makelist(-63,1,s);
 pour j de 0 jusque s-1 faire
  M[j]:=irem(M[j]*cle,28)+63;
 fpour;
 retourne char(M);
}:;
```

On tape : `1/9 % 28`
On obtient `-3 % 28`
On tape : `M5:=Codmulti("MESSAGE@SECRET@POUR@ZORRO",9)`
On obtient : `""MYKKQOYHKYGBYTHLCABHRCBBC"`
On tape : `Codmulti(M5,-3)`
On obtient : `"MESSAGE@SECRET@POUR@ZORRO"`
Mais le plus simple est de travailler dans $Z/29Z$ car 29 est premier et dans $Z/29Z$ tous les nombres non nuls sont inversibles. On pourra avoir dans le message codé le caractère " ?" de code 63.
On tape : `1/9 % 28`
On obtient `13 % 28`.
On tape :

```
Codmultip(L,cle):={
 local s,M,j;
 s:=size(L);
 M:=asc(L)+ makelist(-62,1,s);
 pour j de 0 jusque s-1 faire
  M[j]:=irem(M[j]*cle,29)+62;
 fpour;
 retourne char(M);
}:;
```

On tape `1/9 % 29`
On obtient `13 % 29`
On tape : `M6:=Codmultip("MESSAGE@SECRET@POUR@ZORRO@KKKK",9)`
On obtient : `"QCMMYUCPMCNDCVPOFBDPRFDDFP????"`
On tape : `Codmultip(M6,13)`
On obtient : `"MESSAGE@SECRET@POUR@ZORRO@KKKK"`

8.17.4 Codage sans utiliser`asc` et `char`

On peut aussi sans avoir besoin des fonctions `asc` et `char` écrire un fonction `Code` qui codera 29 caractères par exemple ‘">" par 0," ?" par 1, l'espace par 2, "A" par 3...."Z" par 28 et une fonction `Caract` qui renverra le caractère de code donné en argument.
On tape :

```
Code(a):={
local alphab,code,j;
alphab:=">? ABCDEFGHIJKLMNOPQRSTUVWXYZ";
code:=table(seq(alphab[j]=j,j=0..28));
retourne code[a];
}
:;
```

On remarquera que l'on utilise ici un tableau `code` qui est indicé par des caractères : c'est ce que l'on appelle dans `Xcas` une table.

```
Caract(n):={
local alphab,code;
alphab:=">? ABCDEFGHIJKLMNOPQRSTUVWXYZ";
retourne alphab[n];
}
:;
```

On écrit le codage par addition en utilisant les fonctions `Code` et `Caract` :

```
Codadditi(L,cle):={
local s,j,M;
s:=size(L)
M:="";
pour j de 0 jusque s-1 faire
M:=M+Caract(alphab[irem(Code(L[j])+cle,29));
fpour
retourne M;
}
:;
```

Ou on écrit directement :

```
Codaddition(L,cle):={
local s,j,M,code,alphab;
s:=size(L)
alphab:=">? ABCDEFGHIJKLMNOPQRSTUVWXYZ";
code:=table(seq(alphab[j]=j,j=0..28));
M:="";
pour j de 0 jusque s-1 faire
M:=M+alphab[irem(code[L[j]]+cle,29)];

fpour
retourne M;
}
```

On tape : `M7:=Codaddition("MESSAGE SECRET POUR ZORRO",7)`
On obtient :
`"TLZZHNLGZLJYL>GWV?YGDVYYV"`
On tape : `Codaddition(M7,-7)`
On obtient : `"MESSAGE SECRET POUR ZORRO"`

On écrit le codage par muliplication en utilisant les fonctions `Code` et `Caract` :

```
Codmultipli(L,cle):={
local s,j,M;
s:=size(L)
M:="";
pour j de 0 jusque s-1 faire
M:=M+Caract(irem(Code(L[j])*cle,29));
fpour
retourne M;
}
```

Ou on écrit directement :

```
Codmultiplica(L,cle):={
local s,j,M,alphab,code;
s:=size(L)
alphab:=">? ABCDEFGHIJKLMNOPQRSTUVWXYZ";
code:=table(seq(alphab[j]=j,j=0..28));
M:="";
pour j de 0 jusque s-1 faire
M:=M+alphab[irem(code[L[j]]*cle,29)];
fpour
retourne M;
}
```

On tape : `M8:=Codmultiplica("MESSAGE SECRET POUR ZORRO KKKK",9)`
On obtient :
`"QCMMYUCPMCNDCVPOFBDPRFDDFP????"`
On tape : `Codmultiplica(M8,13)`
On obtient :
`"MESSAGE SECRET POUR ZORRO KKKK"`