Le crible d'Ératosthène

Le principe

Le crible d'Ératosthène est un procédé qui permet de trouver tous les nombres premiers inférieurs à un certain entier naturel donné N.

L'algorithme procède par élimination : il s'agit de supprimer dans une table des entiers allant de 2 à N, tous les multiples d'un entier.

En supprimant tous les multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun autre entier, et qui sont donc des nombres premiers.

On commence par rayer les multiples de 2, puis les multiples de 3, puis ceux de 5 (4 a été rayé car il est multiple de 2), puis les multiple de 7 (6 a été rayé car il est multiple de 2) et ainsi de suite.

À chaque fois on raye les multiples du plus petit entier restant dans le tableau.

On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur à N, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.

À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

Exemple pour N = 30

Nous rayons tous les multiples de 2

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Nous rayons tous les multiples de 3

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Nous rayons tous les multiples de 5

	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30

Nous arrêtons car $7 \times 7 = 49 > 30$. Tous les nombres qui ne sont pas premiers ont déjà été rayés.

Les nombres premiers inférieurs à 30, sont 2, 3, 5, 7, 11, 13, 17, 19, 23, 29.

L'algorithme Algobox

Nous allons créer une liste de nombre « listeNombre » dont les valeurs iront de 1 à 50 (avec Scratch nous irons jusqu'à 100). Une variable i, permet de parcourir cette liste.

La variable « **num** » mémorise, le nombre dont on cherche les multiples.

La variable « **multiple** » est un des multiples de num.

La variable « **multiplier** » permet de passer d'un multiple à l'autre.

Pour rayer un multiple, nous mettons simplement la valeur qui lui correspond dans la liste à 0.

```
7   DEBUT_ALGORITHME
8     //Remplir la liste de valeurs allant de 1 à 50
9     POUR i ALLANT_DE 1 A 50
10      DEBUT_POUR
11      listeNombre[i] PREND_LA_VALEUR i
12      FIN_POUR
13    //num = nombre dont on cherche tous les multiples
14    num PREND_LA_VALEUR 2
```

num est initialisé à 2 : nous allons rayé au départ tous les multiples de 2.

```
16    //On ne teste que les nombres dont le carré est inférieur à 50
17    //car au delà, tous les non-premiers ont déjà été rayés précédemment
18    TANT_QUE (num * num <50) FAIRE
19      DEBUT_TANT_QUE
20      ////lorsqu'on raye un multiple
21      //on met la case correspondante de la liste à 0
22
23      //Si le nombre num de la liste n'est pas rayé
24      //on va rayer ses multiples
25      SI (listeNombre[num] != 0) ALORS
26        DEBUT_SI
27        //Recherche des multiples de num pour les barrer
28        //multiple est un multiple de num
29        multiple PREND_LA_VALEUR num
30        //multiplier permet de calculer les multiples de num
31        multiplier PREND_LA_VALEUR 2
```

On n'entre dans la boucle « Tant…Que » que si tous les multiples de la liste n'ont pas déjà été tous rayés.

Dans cette boucle, on commence par vérifier si num n'a pas été rayé.

S'il n'a pas été rayé, on va rechercher tous ses multiples pour les rayer.

```
33        ////on n'a pas encore testé tous les multiples possibles de num
34        TANT_QUE (multiple<50) FAIRE
35          DEBUT_TANT_QUE
36          //on calcule la valeur d'un nouveau multiple de num
37          multiple PREND_LA_VALEUR multiplier * num
38
39          //Si ce multiple n'est pas déjà rayé
40          SI (listeNombre[multiple] !=0) ALORS
41            DEBUT_SI
42            ////on raye le multiple de num
43            listeNombre[multiple] PREND_LA_VALEUR 0
44            FIN_SI
45
46          //on passe au multiple suivant de num
47          multiplier PREND_LA_VALEUR multiplier+1
48          FIN_TANT_QUE
49        FIN_SI
```

Une fois que l'on a traité tous les multiples de num, on va passer à l'entier suivant.

```
51        //on passe au nombre num suivant
52        num PREND_LA_VALEUR num + 1
53        FIN_TANT_QUE
```

Lorsqu'on sort de la boucle « Tant .. Que » la plus externe, on a terminé.

Il ne reste plus dans la liste de nombre, que des nombres premiers et des zéros.

```
55      AFFICHER "Les nombres premiers sont"
56      POUR i ALLANT_DE 1 A 50
57        DEBUT_POUR
58        //on affiche que les valeurs non nulles de la liste
59        SI (listeNombre[i]!=0) ALORS
60          DEBUT_SI
61          AFFICHER listeNombre[i]
62          AFFICHER " , "
63          FIN_SI
64        FIN_POUR
65    FIN_ALGORITHME
```

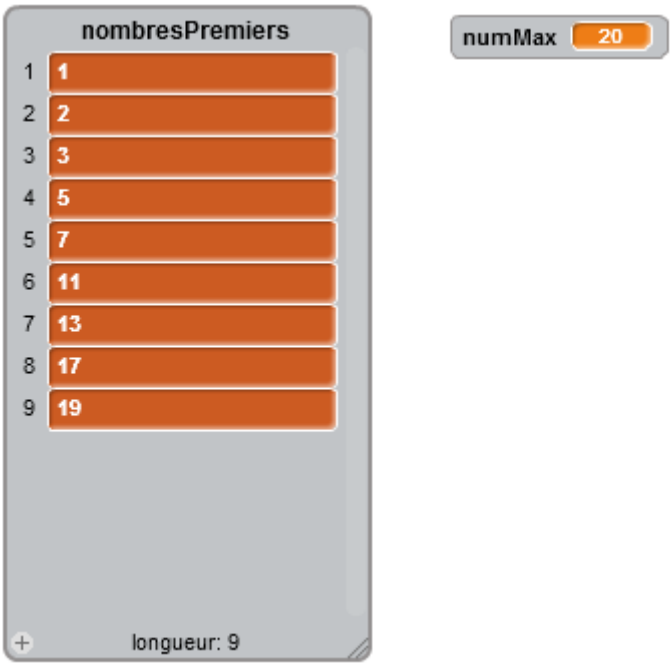
Le projet Scratch simple

Dans ce projet nous avons un lutin, qui n'apparait pas sur la scène.

Nous mettons simplement en œuvre l'algorithme Algobox.

On y ajoutera une variable « **numMax** » qui indiquera la valeur maximum, jusqu'où il faut chercher les nombres premiers et une liste «**nombresPremiers** » visible sur la scène dans laquelle sera recopié les nombres premiers, une fois que le programme les aura tous calculés.

Par exemple pour « numMax » = 20, on verra dans la liste « nombresPremiers » :



Le projet Scratch plus compliqué

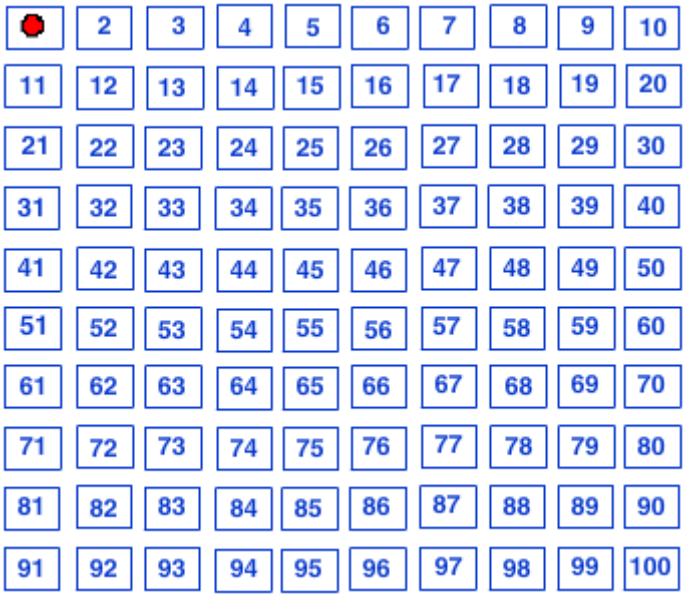
Cette version n'est pas au programme du collège.

Elle montre simplement une technique que nous avons déjà mis partiellement en œuvre dans le projet « Conversion binaire-décimale ».

Il s'agit ici de déplacer un lutin en ligne/colonne sur un damier dont les cases sont numérotées de 1 à 100.

Le lutin doit cacher le nombre affiché dans la case, si ce nombre n'est pas un nombre premier.

L'arrière-plan : j'y ai dessiné 100 cases numérotées, représentant les nombres entiers inférieurs ou égaux à 100.

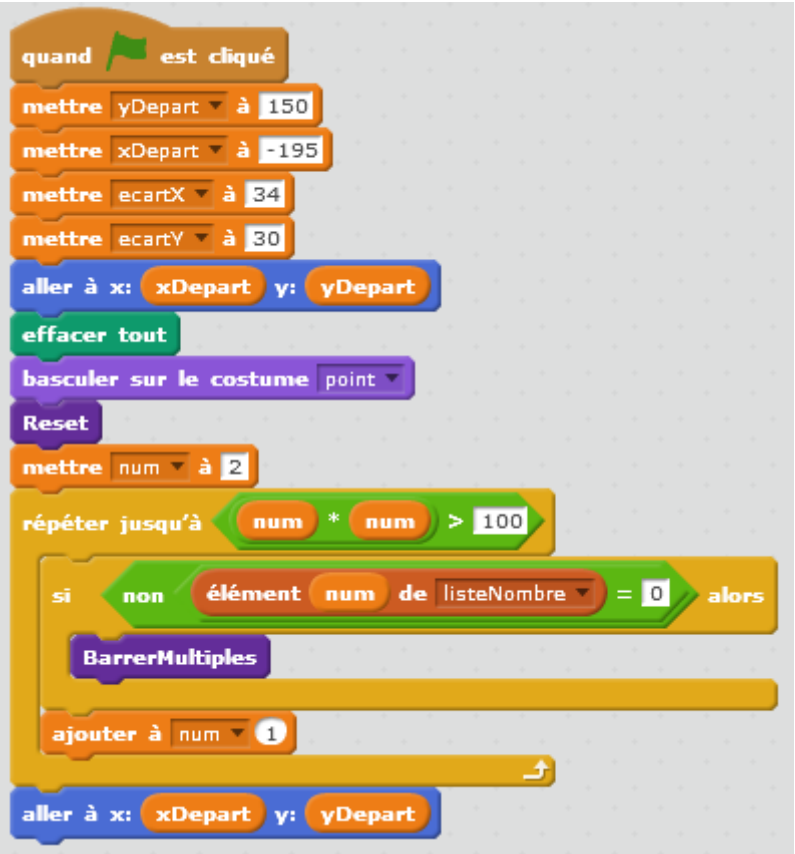


Un lutin « pointeur » possède 2 costumes: l'un représente un point rouge et l'autre une case blanche.

Ce lutin, saute de case en case, et lorsqu'il doit rayer un multiple, il « estampille » son costume « case blanche » sur la case correspondant au nombre à rayer.

Pour savoir si un nombre est rayé ou non, nous utilisons une liste contenant au départ 100 valeurs (de 1 à 100). Lorsqu'on va rayer un multiple, on mettra un 0 à la position correspondant à ce nombre dans cette liste, comme nous l'avons fait dans la version simple.

Le script principal du pointeur



Les variables yDepart, xDepart, ecartX, ecartY sont utilisées pour calculer la position du lutin, de façon à le voir sauter de case en case et estampiller les cases à rayer.

Le lutin se place donc sur la première case en haut à gauche et enfile son costume point.

La procédure Reset initialise la liste « listeNombre » en y mettant des valeurs allant de 1 à 100.

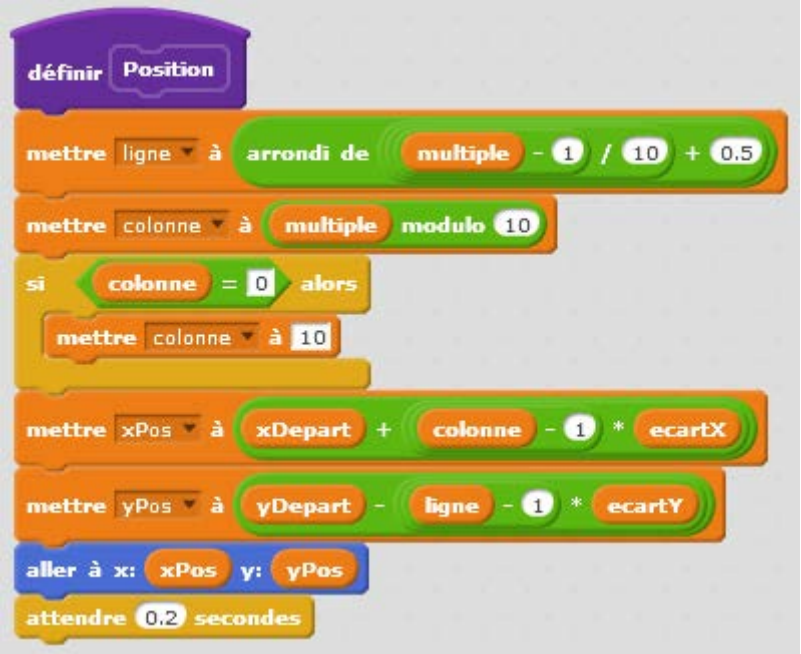
La boucle « Répéter jusqu'à » correspond à la boucle « Tant.. que » la plus externe de l'algorithme Algobox. Ici nous allons jusqu'à 100, au lieu de 50.

Dans cette boucle nous vérifions que le nombre « num », n'a pas été rayé.
S'il n'a pas été rayé, nous rayons ses multiples dans la procédure « BarrerMultiples » et nous passons ensuite au nombre suivant.

La procédure « Position »

Cette procédure est utilisée par la procédure « **BarrerMultiples** » pour placer le lutin sur la case correspondant au nombre ou au multiple en cours de traitement.

À partir de la valeur de « **multiple** », elle calcule le numéro de la ligne et de la colonne où se trouve la case correspondant à ce multiple.



Le numéro de la ligne :

ligne = arrondi de ((multiple – 1)/10 + 0.5)

Si multiple = 2 → ligne = arrondi de ((2 – 1)/10 + 0.5)

= arrondi de (0.1 + 0.5) = arrondi de 0.6 = 1

Si multiple = 3 → ligne = arrondi de ((3-1)/10 + 0.5

= arrondi de (0.2 + 0.5) = arrondi de 0.7 = 1

Si multiple = 10 → ligne = arrondi de ((10-1)/10 + 0.5

= arrondi de (0.9 + 0.5) = arrondi de 1.4 = 1

Le numéro de colonne

colonne = multiple modulo 10

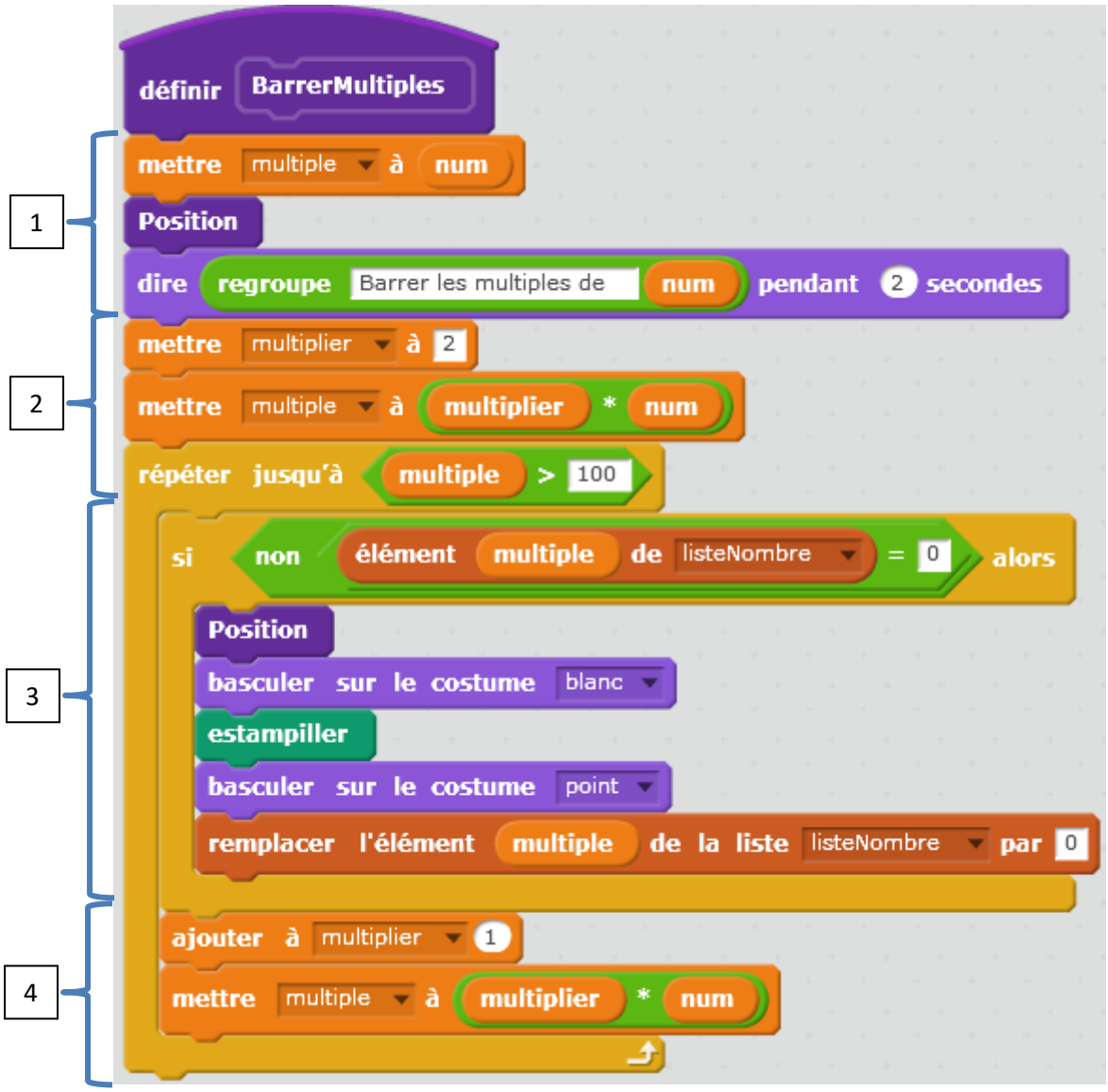
Si multiple = 2 → colonne = 2 modulo 10 = 2

Si multiple = 15 → colonne = 15 modulo 10 = 5

Si **colonne = multiple modulo 10** donne 0, c'est donc que nous sommes sur la colonne 10.

Connaissant le numéro de ligne et de colonne, on peut alors calculer les coordonnées x et y du lutin.

La procédure « BarrerMultiples »



1. Les trois premières instructions permettent de positionner le « point rouge » sur la case du nombre dont on va barrer les multiples.
2. nous calculons le premier multiple de num et nous entrons dans la boucle si ce multiple n'est pas supérieur à 100
3. Si ce multiple n'a pas déjà été barré, nous positionnons le lutin sur la case qui correspond au multiple, nous basculons le lutin sur son

costume blanc et nous estampillons ce costume : le nombre inscrit dans la case est alors caché.

Nous rebasculons le lutin sur son costume « point rouge » et nous indiquons dans la liste des nombres que ce nombre a été barré.

4. Nous passons au multiple suivant de num, et nous repartons au début de la boucle.