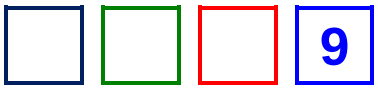
Projet Binaire → Décimal

Le système décimal

Dans ce système, système base 10, il existe 10 chiffres : 0, 1, 2, 3, 4, 5, 6, 7, 8 et 9.

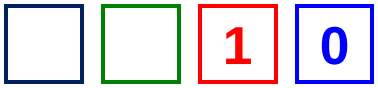
Avec ces chiffres on peut compter jusqu'à 9.

Lorsqu'on compte on commence par les unités : colonne la plus à droite dans un nombre.

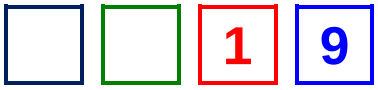


Pour aller au-delà de 9 il faut changer de colonne.

C'est à dire que si la colonne des unités est pleine, on commence la colonne des dizaines et on remet les unités à zéro.



Ensuite, on complète de nouveau la colonne des unités jusqu'à ce qu'elle soit pleine.



On ajoute ensuite, une dizaine dans la colonne des dizaines et les unités sont de nouveau remises à 0, et ainsi de suite.

Par exemple, arrivé à 19, la colonne des unités est pleine.

On ajoute donc une dizaine et on remet à zéro la colonne des unités : on arrive à 20.

		2	0

En complétant ainsi chaque colonne, nous voyons que « une centaine » vaut 10 dizaines et que « une dizaine » vaut 10 unités.

Dans un nombre décimal, le « poids » d'une colonne est égal au poids de la colonne précédente multipliée par 10. Le poids de la colonne des unités vaut 1, celui de la colonne des dizaines vaut 10, celui de la colonne des centaines vaut 100 (10×10), celui de la colonne des milliers vaut 1000 (10×10×10)

On peut dire que dans un nombre décimal, chaque colonne est une puissance de 10 supérieure à la précédente.

De cette manière, le nombre 56 = 50 + 6 peut aussi écrire

$56 = 5\times10^1 + 6\times10^0$.

		10^1	10^0
		5	6

On peut décomposer chaque nombre en puissances de 10 successives.

Par exemple, $3506 = 3\times10^3 + 5\times10^2 + 0\times10^1 + 6\times10^0$.

10^3	10^2	10^1	10^0
3	5	0	6

En base 10, pour écrire un nombre :

On change de colonne dès que la précédente est à 9.

On peut décomposer tous les nombres en puissance de 10.

Le système binaire

Le binaire est le mode de comptage en base 2.

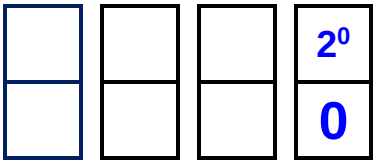
Un bit (contraction de « binary-digit ») ne peut prendre que deux valeurs : 0 ou 1.

Un nombre binaire s'écrit avec une suite de bits : 10011011

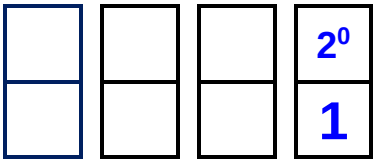
Dans un nombre binaire, le poids d'une colonne est égal au poids de la colonne précédente multipliée par 2.

Chaque colonne est une puissance de 2 supérieure à la précédente.

Valeur décimale 0 :



Valeur décimale 1 :



Valeur décimale 2 : $1\times2^1+0\times2^0$

		2^1	2^0
		1	0

Valeur décimale 3 : $1\times2^1+1\times2^0$

		2^1	2^0
		1	1

Valeur décimale 4 : $1\times2^2+0\times2^1+0\times2^0$

	2^2	2^1	2^0
	1	0	0

Valeur décimale 5 : $1\times2^2+0\times2^1+1\times2^0$

	2^2	2^1	2^0
	1	0	1

Valeur décimale 6 : $1\times2^2+1\times2^1+0\times2^0$

	2^2	2^1	2^0
	1	1	0

Valeur décimale 7 : $1\times2^2+1\times2^1+1\times2^0$

	2^2	2^1	2^0
	1	1	1

En base 2 pour écrire un nombre :

On change de colonne dès que la précédente est à 1.

On peut décomposer tous les nombres en puissance de 2.

Conversion binaire → décimale

Il suffit de prendre chaque bit un par un et de le multiplier par le poids de sa colonne, en partant de la droite.

Le poids de la première colonne vaut 2^0, le poids de la seconde colonne vaut 2^1, le poids de la troisième colonne vaut 2^2 etc.

Par exemple : 10101101 vaut

$2^7 + 2^5 + 2^3 + 2^2 + 2^0 = 128 + 32 + 8 + 4 + 1 = 173$

×128	×64	×32	×16	×8	×4	×2	×1
1	0	1	0	1	1	0	1
↓	↓	↓	↓	↓	↓	↓	↓
128	+ 0	+ 32	+ 0	+ 8	+ 4	+ 0	+ 1

=173

L'algorithme de conversion

Nous avons besoin d'une variable :

nbBinaire qui contiendra le nombre binaire à convertir en décimale.

Ce nombre sera fourni par l'utilisateur, sous forme d'une chaine de caratères.

valeurDecimale qui contiendra la valeur décimale du nombre binaire nbBinaire. Cette variable doit être initialisée à 0.

pos qui contiendra la position du bit que nous sommes en train d'examiner.

Pour examiner les bits un par un, nous devons commencer par la droite.

Soit L, le nombre total de bits du nombre binaire.

Algobox numérotant les caractères d'une chaine à partir de 0, nous devrons positionner « pos » à L-1, pour examiner le premier bit, puis à L – 2 pour examiner le second bit, …jusqu'à L – L pour examiner le bit de poids le plus fort (bit le plus à gauche, c'est à dire premier caractère de nbBinaire)

poids qui contiendra le poids (puissance de 2) correspondant à la position du bit dans le nombre. Sa valeur sera de 1, puis 2, puis 4, puis 8 etc. Cette variable sera initialisée à 1.

i variable de boucle, dont la valeur variera de 1 à L, longueur de « nbBinaire »

```
VARIABLES
  nbBinaire EST_DU_TYPE CHAINE
  valeurDecimale EST_DU_TYPE NOMBRE
  poids EST_DU_TYPE NOMBRE
  pos EST_DU_TYPE NOMBRE
  i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  LIRE nbBinaire
  valeurDecimale PREND_LA_VALEUR 0
  poids PREND_LA_VALEUR 1
  POUR i ALLANT_DE 1 A nbBinaire.length
    DEBUT_POUR
    pos PREND_LA_VALEUR nbBinaire.length - i
    SI (nbBinaire.substr(pos,1)==1) ALORS
      DEBUT_SI
      valeurDecimale PREND_LA_VALEUR valeurDecimale + poids
      FIN_SI
    poids PREND_LA_VALEUR 2*poids
    FIN_POUR
  AFFICHER nbBinaire
  AFFICHER " = "
  AFFICHER valeurDecimale
FIN_ALGORITHME
```

- Examinons ces deux lignes :

```
POUR i ALLANT_DE 1 A nbBinaire.length
  DEBUT_POUR
  pos PREND_LA_VALEUR nbBinaire.length - i
```

À chaque fois que l'on démarre un tour dans la boucle, pos prend la valeur « longueur du nombre binaire » - i.

Au départ i = 1, donc « pos » pointera bien vers le bit le plus à droite dans le nombre binaire.

Au tour suivant i = 2, « pos » pointera alors vers le deuxième bit en partant de la droite dans le nombre binaire.

Au dernier tour, i = « longueur du nombre binaire », « pos » = 0 et pointera vers le bit le plus à gauche dans le nombre binaire.

- Examinons ces lignes

```
SI (nbBinaire.substr(pos,1)==1) ALORS
  DEBUT_SI
  valeurDecimale PREND_LA_VALEUR valeurDecimale + poids
  FIN_SI
poids PREND_LA_VALEUR 2*poids
```

Nous extrayons du nombre binaire, le bit situé à la position « pos » et nous testons s'il vaut 1.

Si oui, nous ajoutons à valeurDecimale, le poids du bit.

Le poids est multiplié par deux en prévision d'un autre tour dans la boucle.

Au premier passage dans la boucle, le poids vaut 1 (bit le plus à droite)

Au deuxième passage dans la boucle, le poids vaut 2 (deuxième bit en partant de la droite)

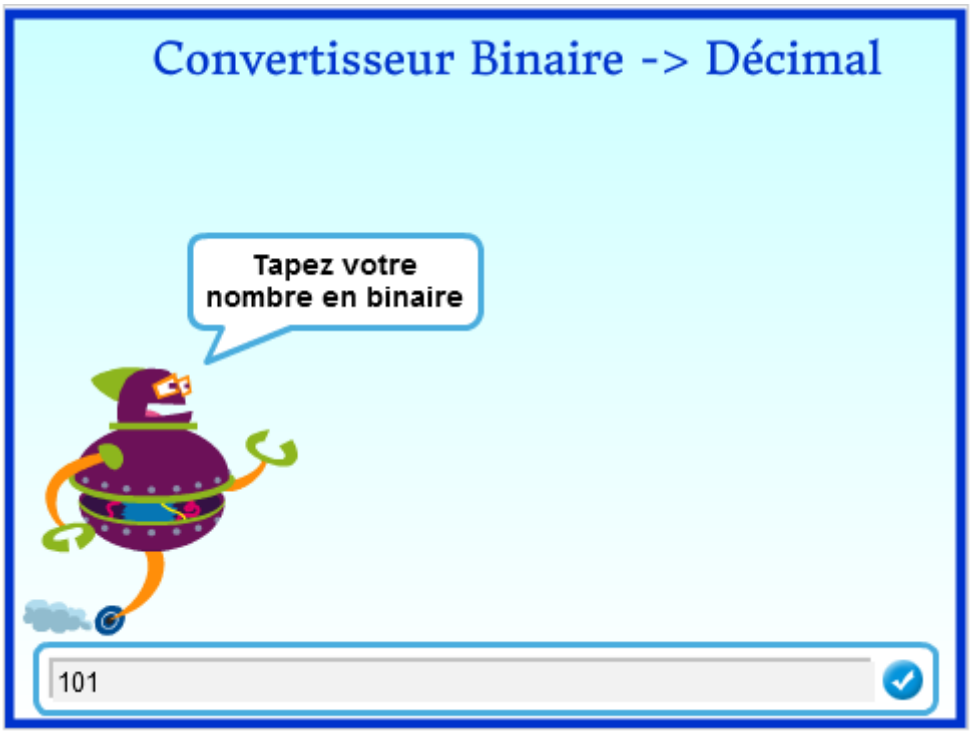
Au troisième passage dans la boucle, le poids vaut 4 (troisième bit en partant de la droite) etc.

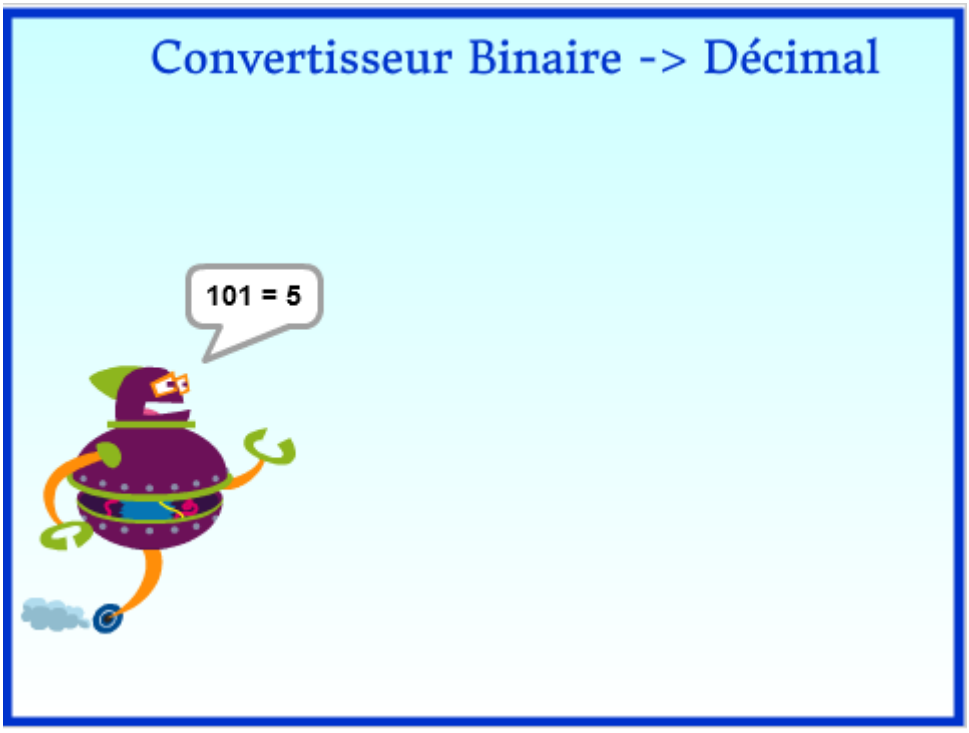
Lorsque tous les bits du nombre binaire ont été examinés, nous avons le résultat de la conversion dans valeurDecimale.

Avec Scratch

La version simple

L'interface du projet





Le lutin demande le nombre décimal à convertir, il effectue la conversion et donne le résultat.

Nous utilisons 4 variables : « **réponse** » contenant le nombre binaire tapé par l'utilisateur, « **valeurDecimale** » qui contiendra la valeur décimale correspondant au nombre binaire tapé, « **poids** » qui contient le poids de la colonne du bit en cours d'évaluation et « **pos** » qui contient la position du bit en cours d'évaluation.

Aucun contrôle n'est fait sur la validité du nombre fourni par l'utilisateur.

Dans une version 2, de ce projet, nous pouvons rajouter ce contrôle.

Tous les chiffres tapés par l'utilisateur doivent avoir pour valeur 0 ou 1.

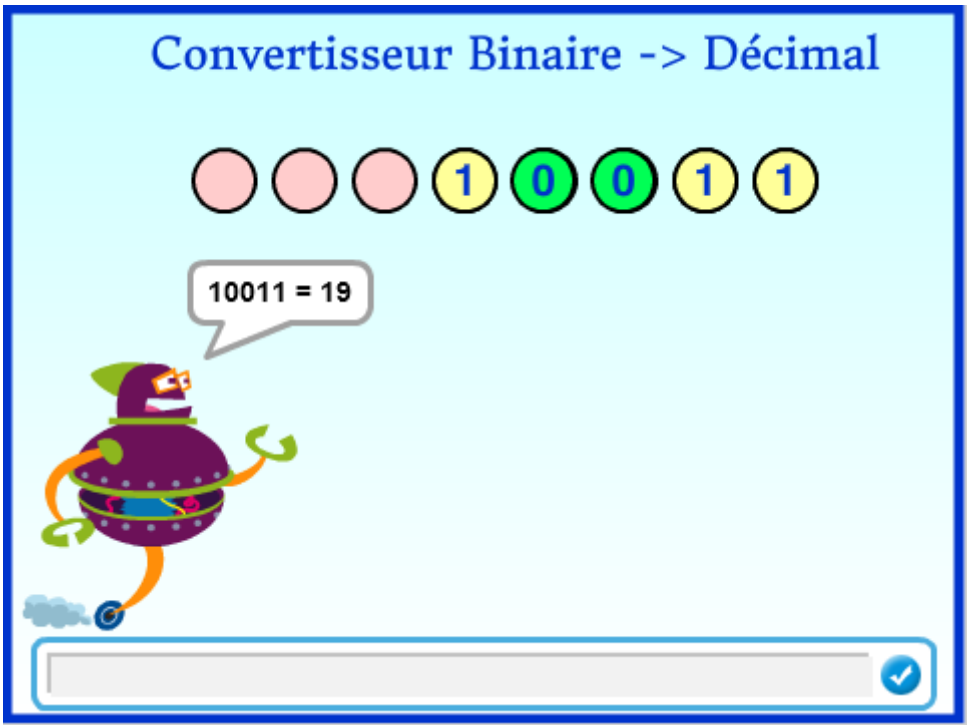
Une version plus sophistiquée

Nous rajoutons à la version précédente du projet, une visualisation du nombre binaire. Nous n'irons pas plus loin que 8 bits.

Cette version du projet n'est pas au programme du collège.

Cependant elle montre une technique de placement du lutin en des points précis de la scène, que l'on retrouve dans tous les jeux, où un lutin parcourt une grille.

L’interface de notre projet



Nous ne traiterons que des nombres binaires, ne comportant pas plus de 8 bits. La valeur décimale variera donc entre 0 et 255.

Nous avons 2 lutins.

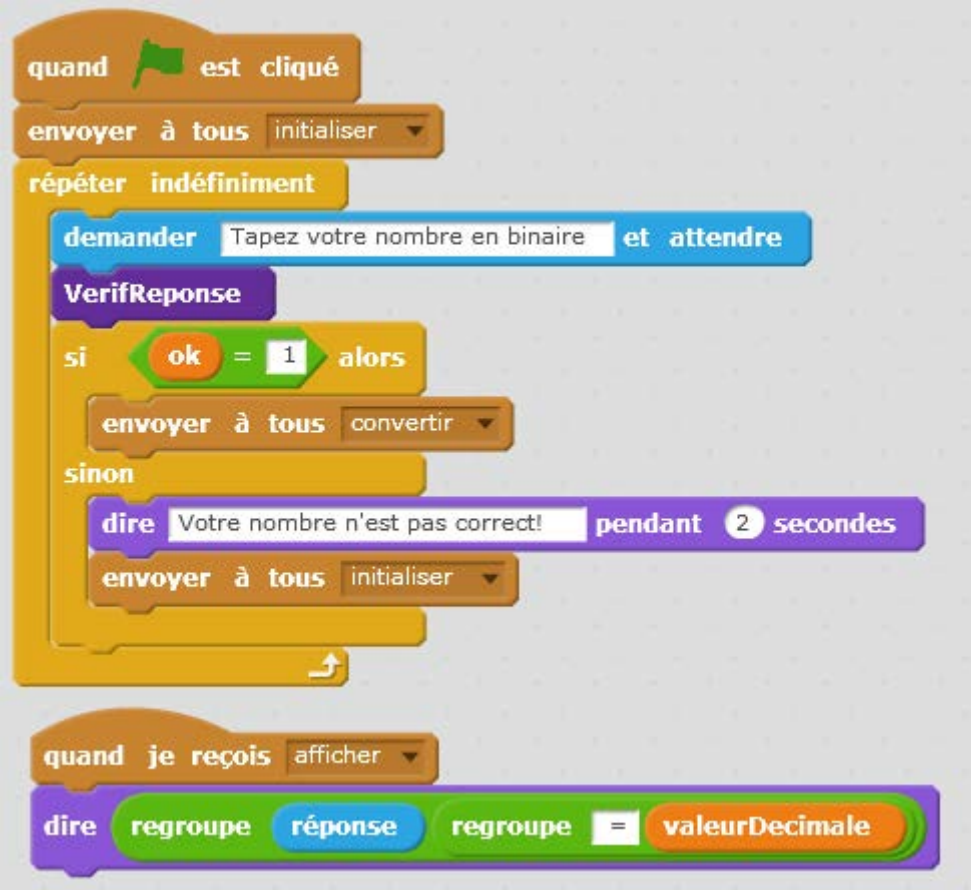
Le lutin « robot » demande le nombre binaire à convertir, vérifie que le nombre fourni est conforme (pas plus de 8 bits et chaque bit ne peut prendre que la valeur 0 ou 1), lance la conversion puis affiche le résultat.

Le lutin « Binaire » possède 3 costumes :

vide  , à 1  , à 

C’est ce lutin qui effectue la conversion et affiche le nombre binaire à convertir.

Le robot



Ce lutin utilise 4 variables :

- deux variables globales (visibles par tous les lutins) : « **réponse** » contenant le nombre binaire fourni par l’utilisateur, et **« valeurDecimale** » contenant la valeur décimale correspondant au nombre binaire fourni.

- deux variables locales (visibles uniquement du robot) : « **indice** » une variable qui nous servira à parcourir le contenu de « réponse » dans la

procédure de vérification du nombre binaire fourni et « **ok** » qui indique après vérification du nombre binaire, si celui-ci est correcte ou non.

- Lorsqu'on démarre en cliquant sur le drapeau vert, le robot demande au lutin « Binaire » de s'initialiser c'est-à-dire d'afficher 8 bits vides (roses).
- Il entre ensuite dans une boucle infinie (on ne peut l'arrêter qu'en cliquant sur le bouton rouge).
    - Dans cette boucle, il demande à l'utilisateur de fournir son nombre binaire. Ce nombre est mémorisé dans la variable « réponse »
    - Il lance ensuite la vérification de ce nombre. Cette procédure positionne une variable « ok » à 1 si le nombre fourni est correcte, et 0 s'il est erroné.
    - Si ok = 1, le robot demande au lutin « Binaire » d'effectuer la conversion.
    - Si ok = 0, le robot signale que le nombre n'est pas correcte, demande au lutin « Binaire » de se réinitialiser et on repart au début de la boucle infini.

Le robot affiche le résultat de la conversion, lorsque le lutin « Binaire » lui dit de l'afficher :

La procédure **VérifReponse** est expliquée dans la vidéo.

Le lutin « Binaire»

Ce lutin utilise 8 variables :

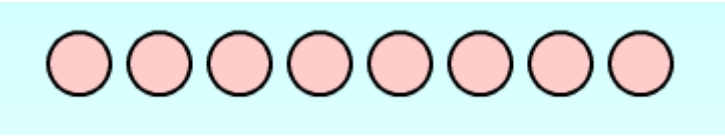
- deux variables globales : « **réponse** » et « **valeurDecimale** »

- 6 variables locales : « **x** », « **y** », « **écart** », « **position** » servant à positionner et dessiner les bits sur la scène.

« **poids** » poids du bit que l'on va traiter

« **pos** » position du bit à traiter dans le nombre binaire

La phase d'initialisation



Dans cette phase, le lutin affiche 8 bits vides.

Ces bits sont dessinés sur une ligne d'ordonnée y, à partir d'une position d'abscisse x, emplacement du bit le plus à droite.

L'écart entre le centre de chaque rond est de 40 pixels.

J’ai choisi d’utiliser 3 variables locales à ce lutin (le robot ne les voit pas), pour mémoriser x, y et l’écart de façon à ce qu’il soit plus facile de mettre au point l’interface du projet.

En modifiant les valeurs de ces variables, on écarte plus ou moins les ronds, on les positionne plus ou moins à droite, plus ou moins haut.

- Lorsqu’il reçoit l’ordre de s’initialiser, le lutin efface ce qui est dessiné sur la scène et enfile son costume « vide » (rond rose).
- La variable position est initialisée à la position en x où il faut dessiner le costume que le lutin a enfilé.
- Nous devons dessiner 8 bits, alors nous entrons dans une boucle « répéter 8 fois ».

- Dans cette boucle, le lutin se place à la position où il faut dessiner.
- Il décalque sur la scène son costume. L'instruction « estampiller » fonctionne comme un tampon. Elle dessine sur la scène le costume du lutin là où se trouve le lutin.
- La variable position est modifiée pour pointer vers l'emplacement du prochain bit à dessiner et on remonte au début de la boucle.

La conversion est expliquée dans la vidéo.