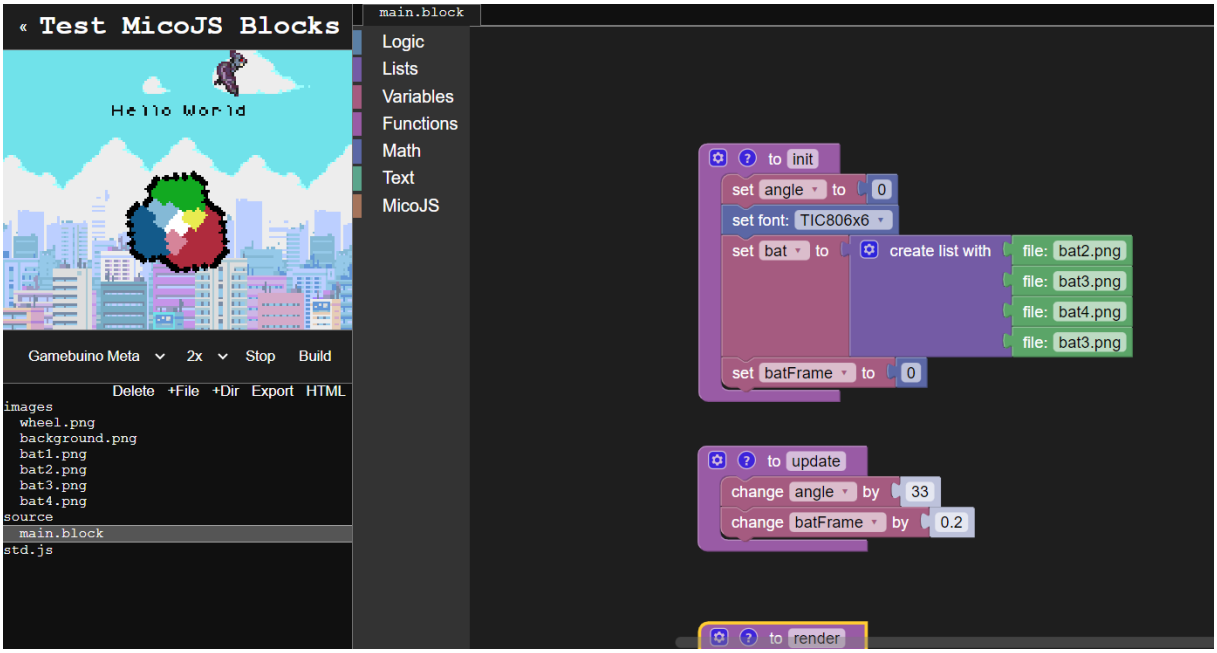
# MicoJS Blocks

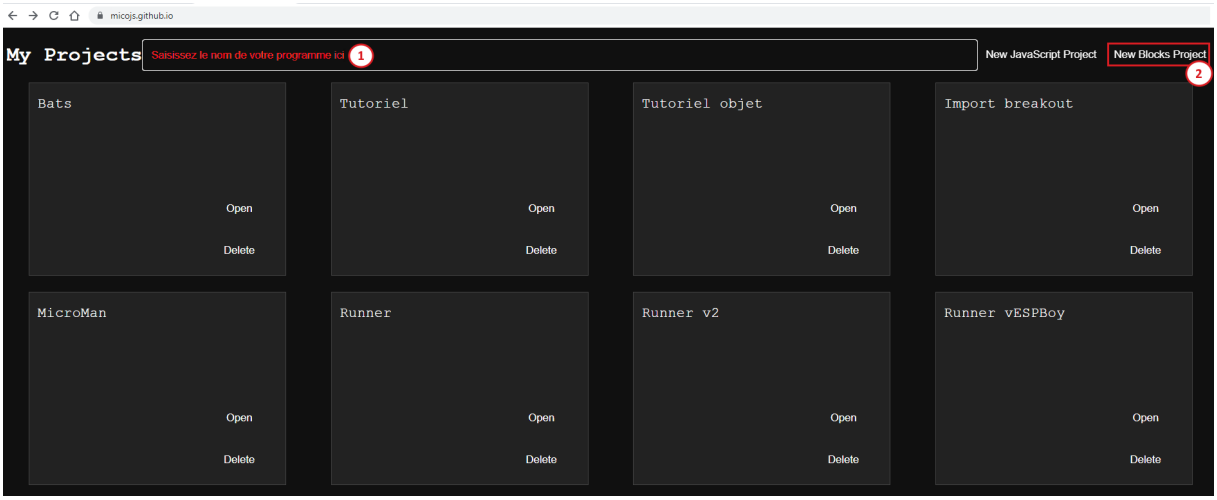
## Introduction et lancement

MicoJS Block est un langage de programmation conçu pour apprendre à coder facilement comme d'autres langage tel que Scratch par exemple qui a été conçu par le MIT. Il fait partie des langage dans lesquels il n'est pas utile de saisir des lignes de code.

Le programme est écrit via une interface visuelle dans laquelle on va assembler des blocs les uns dans les autres et ajuster leur paramètres pour obtenir le résultat escompté.



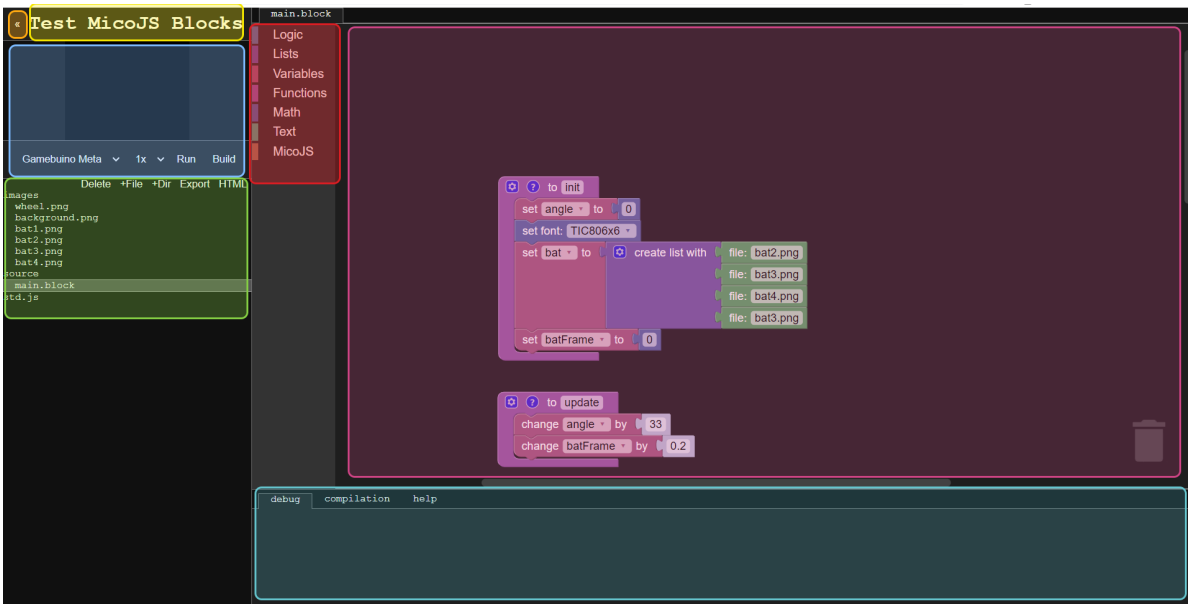

Pour lancer MicoJS, vous pouvez utiliser cette URL : https://micojs.github.io/




Vous pouvez alors saisir un nom pour votre projet (le nom de votre programme) (En 1 sur la capture ci-dessus) puis validez en cliquant sur « New Blocks Project » (Le 2 sur la capture ci-dessus)

# Présentation de l'interface

L'interface est divisées en plusieurs parties, j'ai mis en évidence les blocs en les regroupant dans un bloc de couleur.




**Le bloc orange** :  , cette flèche permet de **retour**ner sur la page de sélection des projets (programmes MicoJS)

**Le bloc jaune** : c'est juste le **titre** de votre programme

**Le bloc bleu** : c'est la partie permettant de **visualiser** l'exécution de votre code. Vous pouvez choisir la plateforme de destination parmi celles proposées : Gamebuino META, Galactic Unicorn, Pokitto, ESPboy, PicoSystem, 32Blit. Vous pouvez choisir un facteur d'agrandissement, lancer l'exécution du programme qui s'affiche alors dans la scène ou le compiler. Certaines options peuvent varier en fonction du matériel sélectionné.

Pour simplifier ce tutoriel, nous partirons du principe que notre code sera à destination d'une Gamebuino META.

**Le bloc vert** : c'est la partie **gestion des fichiers**. Vous pouvez voir tous les fichiers inclus dans le projet, en ajouter, en supprimer, exporter l'ensemble du projet dans un fichier ZIP ou sous forme d'une page HTML. Dans un premier temps, c'est ici que vous retrouverez par exemple les images de vos sprites (lutins en français mais bon, on dit plus sprites que lutins…)

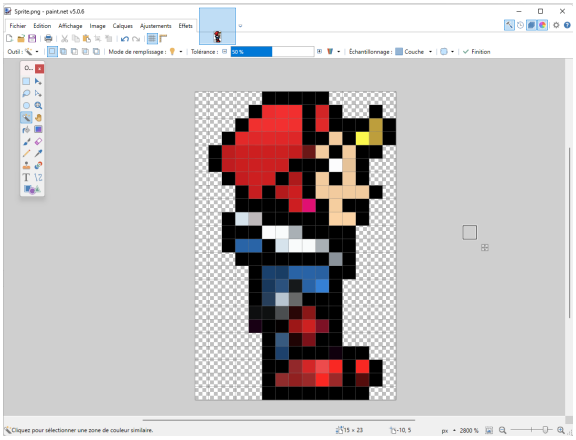
**La partie rouge** : c'est la zone permettant de **sélectionner un bloc** à ajouter à notre programme. Les blocs sont regroupés par thèmes mais nous approfondirons lors de la création de notre premier programme.

**La partie rose** : c'est ici que nous assemblerons nos blocs de code. C'est la partie **espace de travail**.

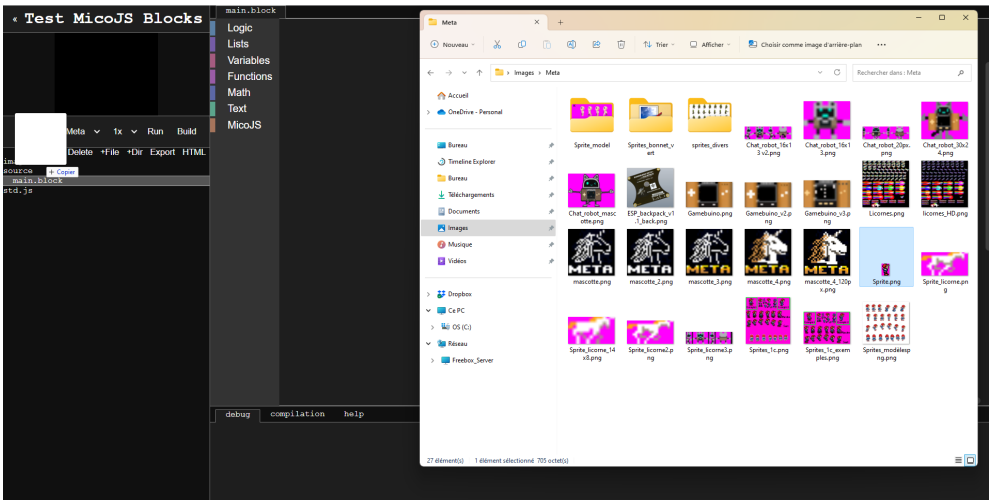
**La partie cyan** : c'est la **console**, c'est ici que s'affiche les messages de debuggage du code, les résultat de compilation. On y retrouve également un onglet permettant de retrouver facilement l'aide sur MicoJS.

# Fiche 1 : Utilisons l'interface pour ajouter un sprite et l'afficher

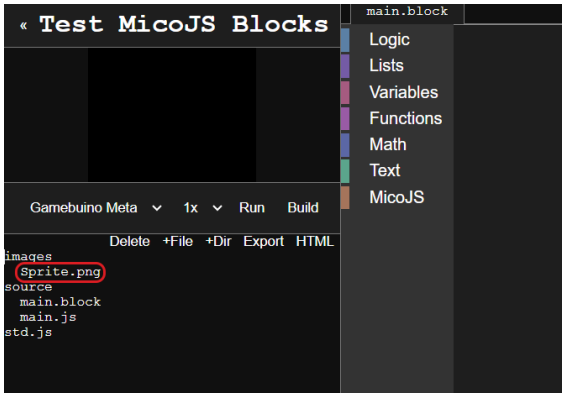
Le sprite (ou lutin) est une ressource graphique essentielle dans notre programme Blocks MicoJS. En effet, c'est un des éléments qui sera affiché et qui sera soit contrôlé par le joueur dans le jeu, soit par le programme pour interagir avec lui ou juste faire joli, mais dans tous les cas, cela contribuera au rendu du code.



Le sprite est donc à la base une image. On la créé à partir d'un logiciel de dessin (Paint dans Windows, Aesprite, Gimp, Paint.net, etc…), puis on fait glisser le fichier depuis l'explorateur de fichier vers la zone « images »




Le sprite s'affiche alors dans la liste des ressources disponibles pour notre projet

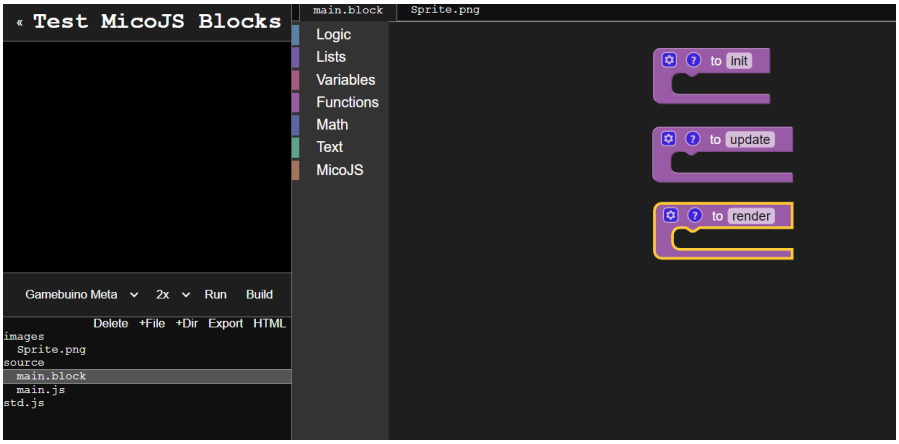


Il est maintenant temps d'afficher notre sprite.

Pour cela, partons de nos 3 blocs de base :

- init
- update
- render

C'est 3 blocs sont essentiels dans un programme MicoJS Blocks. Ils correspondent à 3 fonctions fondamentales de MicoJS :



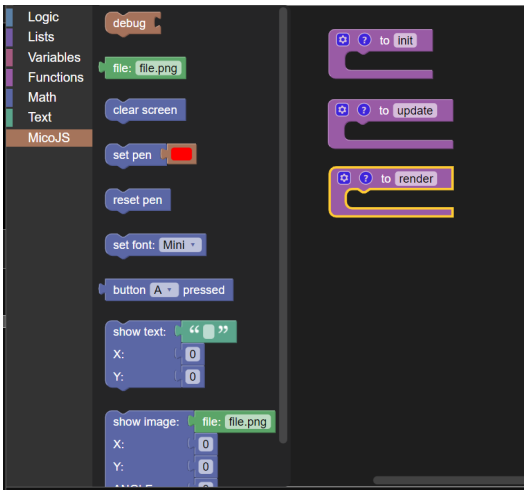

- `init` : Le bloc init est appelée automatiquement au démarrage du jeu une fois que tout est chargé et prêt à être utilisé. Les ressources et les informations sur le mode écran peuvent ne pas être disponibles avant ce point, c'est pourquoi init est exécuté une seule fois mais toujours en premier, avant les 2 autres blocs ne soient appelées en boucle indéfiniment (enfin jusqu'à ce que l'on arrête le programme d'une façon ou d'une autre...) ...
  Init sert à initialiser le programme: à définir tout ce qui doit être définit avant de l'exécuter.

- `update` : Les blocs situées dans le bloc update sont exécutées en boucle. C'est dans cette partie que l'on vérifie les changements des entrées (l'état des boutons par exemple, le passage du temps et ses conséquences sur l'état des différents objets de notre programme) et que l'on effectue les mises à jour du programme. Ce bloc peut être appelé plusieurs fois pendant l'affichage d'une image, il ne faut donc rien afficher dans ce bloc (donc ne pas afficher d'image, ou de texte par exemple). Ici, on calcule la position des éléments, on fait les vérifications (collisions, appuie sur les boutons, changements d'état du jeu: en pause, fin de partie, etc...), mais nous rentrerons plus en détail sur ces choses au fil de notre progression.

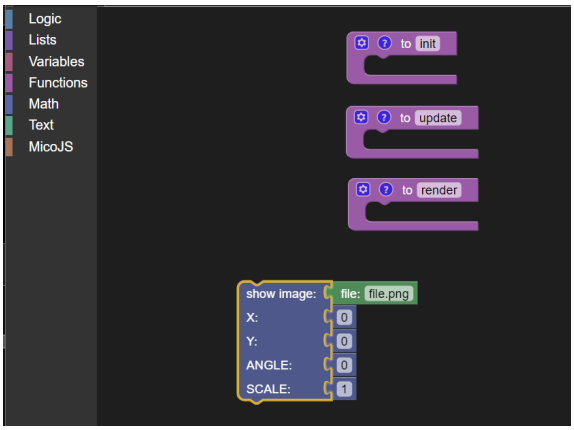
- `render` : Ce bloc sert à dessiner à l'écran. Il ne faut pas mettre à jour les variables du programme ici. Comme tous les calculs sont déjà fait dans le bloc update, l'exécution de ce bloc est normalement simple et rapide.

  C'est ici que nous allons rajouter le bloc qui va afficher notre sprite

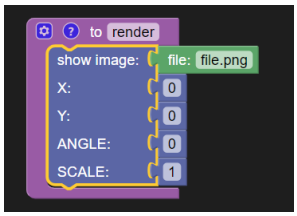
Pour cela, dans la partie de **sélection d'un bloc**, nous allons cliquer sur « **MicoJS** »




Les blocs de cette rubrique s'affiche. On va cliquer sur le bloc show image et tout en maintenant le bouton gauche de la souris enfoncé, on va le faire glisser dans le bloc **render**.

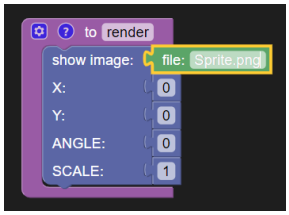



Quand l'ancre du bloc passe en jaune, lâché le bouton gauche de la souris, vous entendez un clic et le bloc est ajouté.

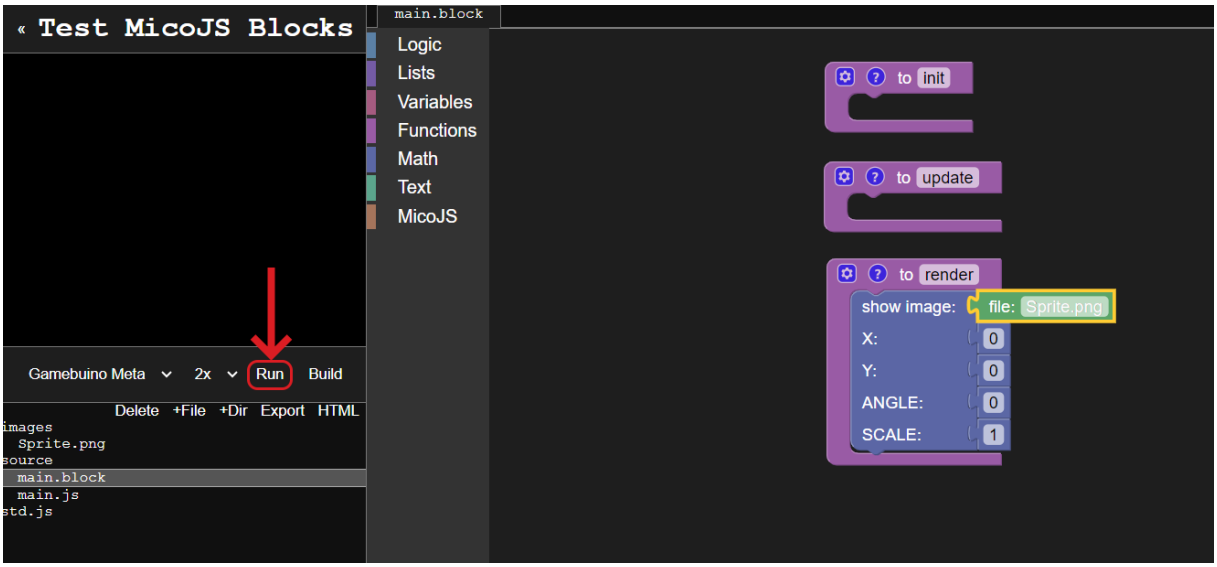



Nous avons ajouté un bloc « show image » mais il n'est pas encore paramétré. Pour le moment, il a des valeurs par défaut qu'il va falloir que l'on ajuste pour qu'elles correspondent à nos besoins.

Pour cela, cliquez sur les valeurs à modifier. Par exemple pour moi, mon sprite ne s'appelle pas file.png mais Sprite.png (je me demande où j'ai pu aller chercher un nom aussi compliqué 😊 )



Je modifie donc la valeur de ce paramètre dans le bloc. Vous pouvez également jouer avec les autres valeurs si vous le souhaitez afin de voir le résultat en cliquant sur « Run »

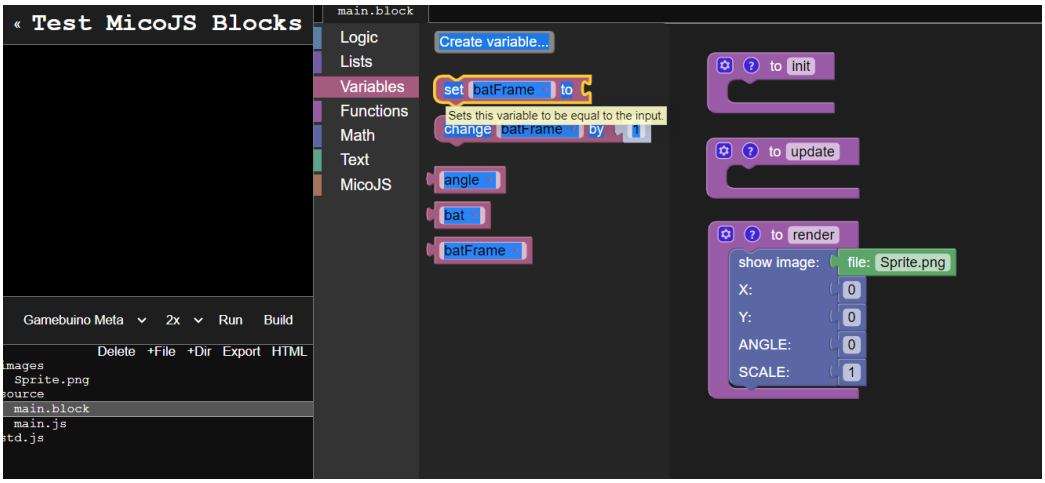



Et normalement, votre sprite s'affiche. Vous pouvez modifier les valeurs de X et de Y. Pour tester, cliquez alors sur Stop, puis relancez le programme après avoir fait les modifications que vous souhaitiez.

C'est sympa, non ? Bon, après, ça manque vite d'intérêts s'il ne se passe rien, alors nous allons ajouter quelques instructions dans le bloc update pour créer du mouvement. Pour cela nous allons ajouter un autre élément fondamental : une variable.

Une variable est un espace dans la mémoire permettant de stocker temporairement une donnée. Il peut s'agir du score actuel, de la position d'un personnage, de texte, … Lorsque l'on arrête le programme ou que l'on éteint la console, la valeur est perdue.

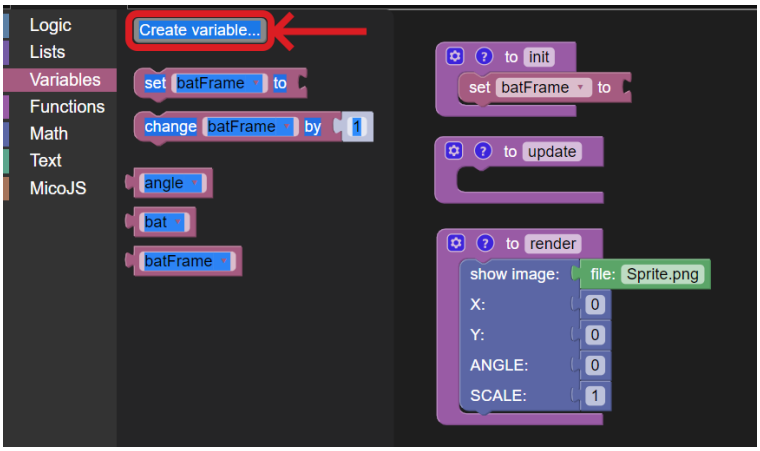
On la déclare et lui donne une **valeur initiale**, à l'aide du bloc set qui se trouve dans la section Variables des blocs.



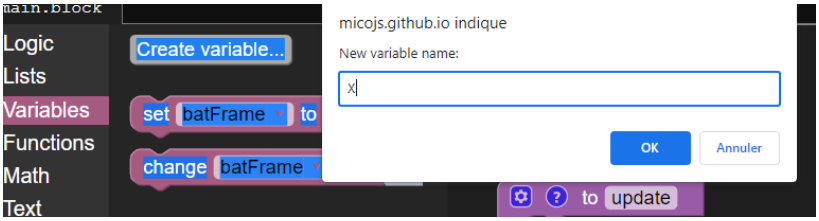

On prend donc ce bloc et on va le faire glisser dans le bloc init pour initialiser notre variable lorsque l'on lance le programme.

Une variable a un **nom**, ici on a choisi d'appeler notre variable X car elle va nous servir à déplacer le sprite horizontalement, donc à modifier son abscisse (sa valeur sur l'axe X). Il faut un nom qui décrit bien ce à quoi elle sert pour que le code reste lisible. En utilisant des variables qui s'appellent a, b et c, on se mélangerait vite les pinceaux lorsque le programme deviendrait plus complexe. Nous aurons l'occasion d'en reparler plus tard. Pour le moment paramétrons notre bloc set comme nous le souhaitons.

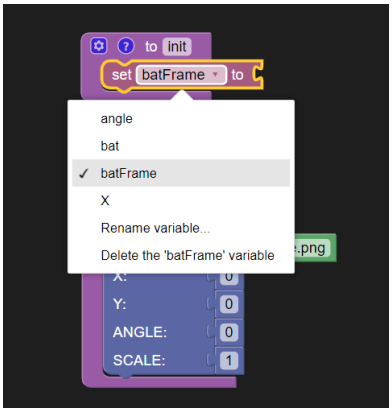
Pour cela, il nous faut créer notre variable X. Pour cela, toujours dans la catégorie Variables, cliquons sur Create variable




Saisissons son nom : X et validons par « **OK** »



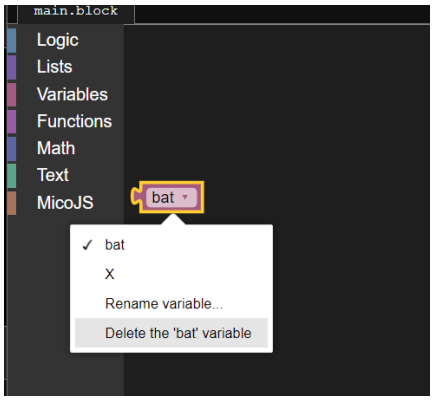

Maintenant que la variable existe, elle est disponible dans le bloc set que nous avons ajouté. Si on clique sur l'emplacement du nom de la variable, elle est disponible dans la liste :



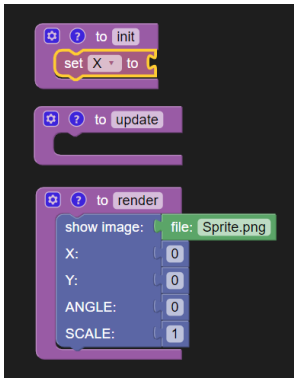

Note : Vous pouvez en profiter pour supprimer des variables non utilisés. Par exemple dans ma liste, je n'utilise pas les variable angle, bat et batFrame. Pour les supprimer, je clique dessus pour les sélectionner puis je clique sur Delete.
Notez qu'au lieu de créer une nouvelle variable, j'aurais pu aussi renommer une des anciennes

variables pour la réutiliser en fonction de mes besoins. Ca peut être utile si vous avez des évolutions de vos besoins en cours d'élaboration du programme mais nous ne nous y attarderons pas pour le moment.

La solution la plus simple pour supprimer une variable et de la sélectionner dans la liste des variables puis de cliquer sur Delete dans la liste déroulante

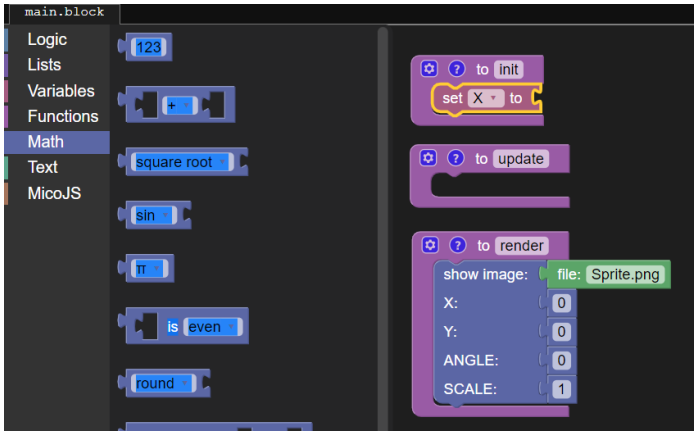



Mais reprenons. Actuellement nous avons donc notre bloc set ainsi :

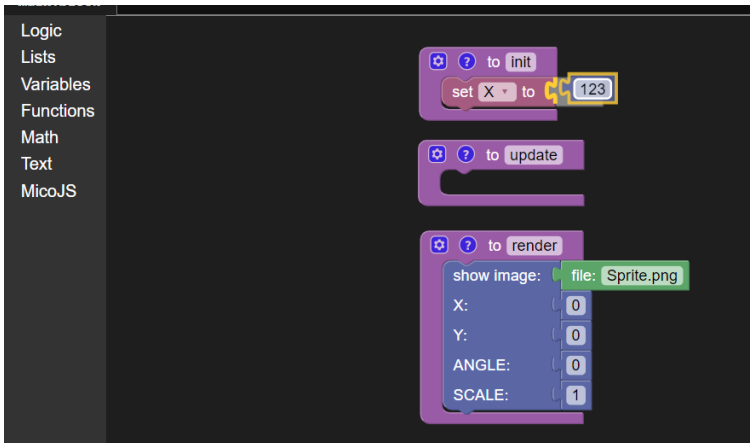



Nous lui disons donc que nous voulons initialiser la variable à une valeur que nous n'avons pas encore ajouté. On voit qu'il est possible d'accrocher un bloc à droite du bloc set. On peut y mettre par exemple un autre bloc correspond à une variable pour que X prenne la valeur de cette variable ou y mettre une valeur fixe, ce qui sera notre cas pour l'initialisation de X.

Pour ajouter une valeur fixe, ouvre la section Math



Faites glisser le premier bloc à droite de notre bloc set.

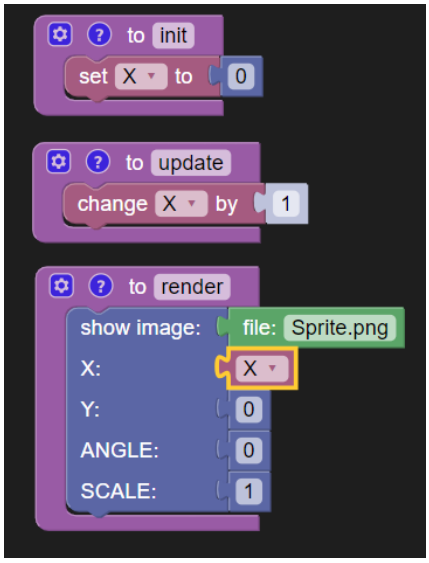



Modifiez ensuite la valeur qui s'y trouve par 0

Ouvrez la catégorie Variables et ajoutez le bloc change pour qu'il augmente de 1 à chaque exécution de la boucle.

Enfin, remplacez la valeur fixe 0 du bloc show image situé dans le bloc render par la variable X.

Je ne vous explique pas comment faire, vous avez déjà tout vu, mais à la fin vous devriez obtenir ce programme :




Utilisez « Run » pour le lancer.

Votre personnage part du côté gauche et avance… puis sort de l'écran et disparait à tout jamais ou du moins jusqu'à ce que vous cliquiez sur « Stop » puis de nouveau sur « Run ».

C'est normal, le programme ne fait que ce que l'on lui demande. On lui demande d'augmenter X à chaque occurrence de la boucle, il le fait. A aucun moment on ne contrôle le fait que X soit supérieur à la taille de l'écran. Il continue donc jusqu'à atteindre ses limites.

Il va falloir que l'on rajoute des choses mais ce sera dans une prochaine étape. Pour le moment, bravo, vous avez appris à lancer MicoJS Blocks, à utiliser son interface et à créer votre premier programme. Amusez vous un peu avec ce que vous avez vu et faites vos propres tests et expérimentations avant de passer à la fiche suivante.