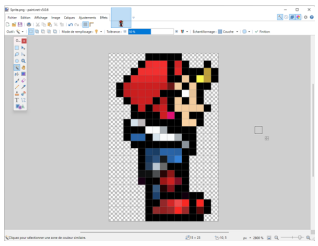
MicoJS Blocks

Fiche 2 : Animons un sprite et contrôlons le avec les boutons

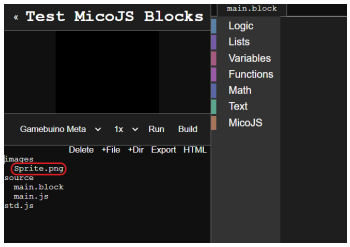
Animons notre sprite

Commençons par reprendre ce que nous avons vu à la fin de la fiche n°1 : 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.



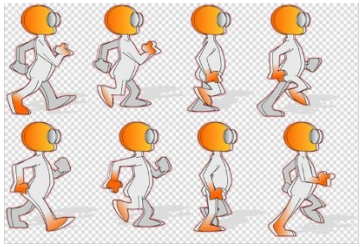
Le sprite est donc à la base une image. On la créé à partir d'un logiciel de dessin (Paint dans Windows, Aesprite, Gimp, Paint.net, etc…), puis on fait glisser le fichier depuis l'explorateur de fichier vers la zone « images »

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



Bien nous avons également vu dans la fiche n°1 comment l'afficher à l'écran et comment le déplacer à travers un programme très simple. En fait le sprite dans notre cas correspond à une 'frame' de notre personnage. Une frame, je dirais que cela correspond à l'image d'un objet à un moment donné pour une situation donnée. Il a une pose qui correspond à son état dans cette situation.

Prenons un exemple :

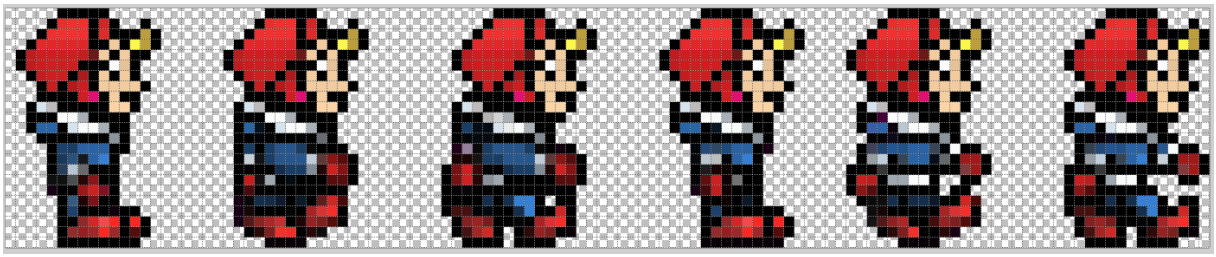


Cette image comprend les 8 frames d'un personnage en train de marcher. Chaque image correspond à l'état du personnage lorsque nous voulons l'animer pour donner l'impression qu'il marche.

En règle général, il y a aussi une frame qui correspond au moment ou il est au repos et il y aurait une autre série de frame pour chacune des animations que nous voudrions lui ajouter pour illustrer son état lorsqu'il accomplit d'autres actions.

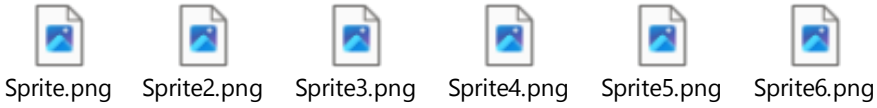
En effet, une animation est la succession plus ou moins rapide d'image qui va donner l'impression que l'action se déroule. Elle a donc un nombre d'étapes (frames) variable en fonction de l'effet visuel souhaité et on évite d'en créer de trop si cela n'est pas nécessaire afin de ne pas trop ralentir le jeu ni saturer la mémoire de notre console. Ce sera à vous de voir ce que vous voulez mais vous pourrez vous baser sur la large gamme de jeu déjà existant et globalement sur ce constat : plus un objet est petit moins il est utile d'avoir un nombre important de frame.

Reprenons l'exemple de notre personnage. Pour le faire marcher, nous allons créer 6 étapes :



Nous allons sauvegarder chaque étape sous la forme d'une image et les importer dans notre programme. Le premier correspond à notre Sprite.png, nous allons donc ajouter Sprite2.png, …Sprite6.png

Je vous mets les images ici pour que ce soit plus simple :

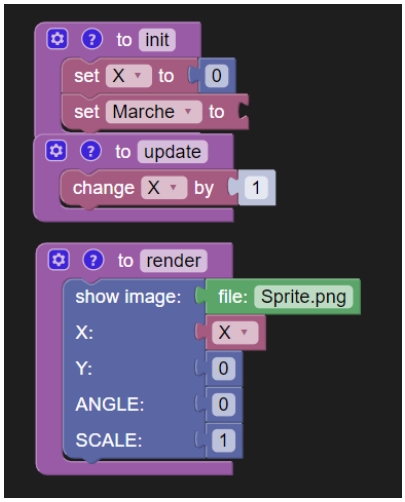


Après leur intégration dans votre code, vous devriez avoir un écran similaire à celui-ci :

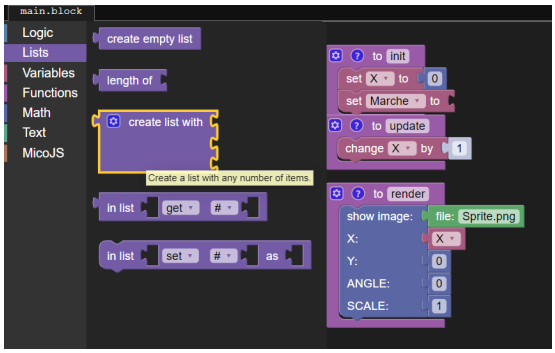
Il y aurait plusieurs façons de créer notre animation. Nous pourrions afficher les images manuellement à l'aide d'un compteur de frames par exemple mais avec MicoJS, il y a un type de données particulièrement utile pour ce type d'animations : les listes.

Nous allons donc créer une liste « Marche » en y ajoutant chacune des images de notre animation.

Pour cela, créons une variable « Marche » puis prenons un bloc « set » et ajoutons le dans le bloc Init



Mais cette fois, nous n'allons pas lui donner une valeur fixe. Ouvrons la catégorie « Lists » et prenons le bloc « create list with » et ajoutons le à droite du bloc « set Marche to »

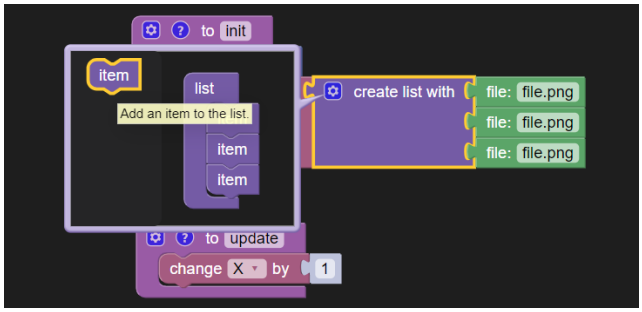


Ensuite comme nous avions ajouté le fichier Sprite.png dans notre bloc « show image » dans la première fiche, nous allons ajouter les 6 fichiers d'image dans notre liste Marche.

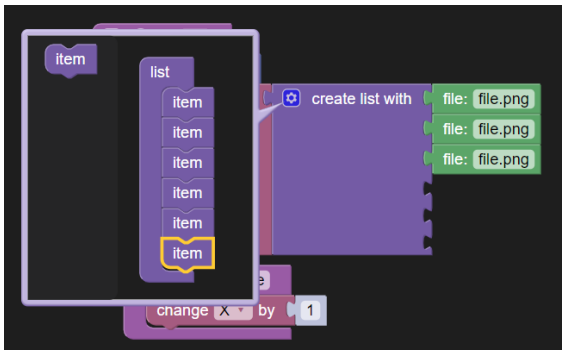
Attention, par défaut, vous ne pouvez ajouter que 3 éléments dans votre liste. Il va falloir ajouter 3 autres emplacement dans votre liste, mais vous allez voir, c'est très simple :

Lorsque le bloc est sélectionné, cliquez sur la roue crantée

Prenez « Item » et ajoutez le 3 fois en bas de votre liste



Vous devriez obtenir un bloc ressemblant à celui-ci :

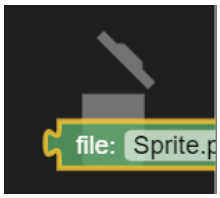


Recliquez sur la roue crantée pour fermer le paramétrage de l'objet, ajoutez les autres blocs files et modifiez les avec les noms de vos images en en respectant bien l'ordre pour que l'animation soit logique.

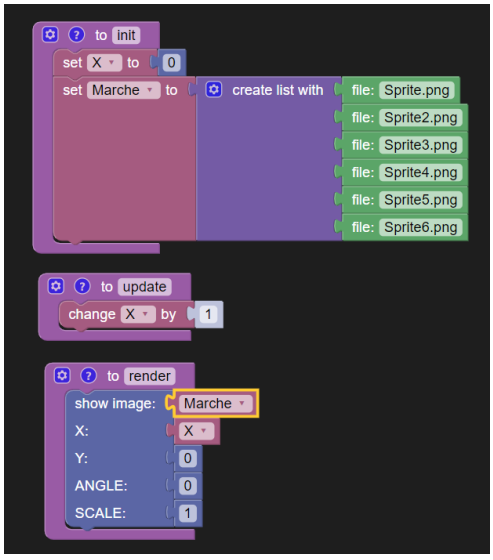
Note : Quand vous modifiez un nom de fichier, après l'avoir modifié, cliquez sur « Tabulation » pour passer directement à l'élément suivant (le nom suivant du fichier dans la liste).

Voilà, notre liste est prête. Nous allons modifier le bloc « show image » pour qu'il n'affiche plus le sprite : Sprite.png de manière fixe mais qu'il affiche notre animation « Marche »

Pour cela, supprimez le bloc « file :Sprite.png » en le faisant glisser sur la poubelle qui se trouve en bas à droite de la zone d'édition de votre programme :



Puis ouvre la catégorie « Variables », prenez le bloc « Marche » et mettez le à la place du bloc que nous venons de supprimer.



Lancez votre programme et regardez ce qu'il se passe…

Il ne se passe rien…

C’est normal.

Pour utiliser la liste, nous devons lui indiquer que nous voulons qu’il affiche une image particulière de cette liste.

Pour cela nous allons créer un index. Un index, c’est une variable qui va parcourir notre liste de son début jusqu’à sa fin.

A savoir : une liste commence à 0. La première image correspond donc à un index de 0 et la dernière image à 5 dans notre cas (nombre d’éléments – 1).

Vous savez faire, alors je ne vous le réexplique pas mais vous allez :

- créer une variable Index
- l’initialiser à 0 dans le bloc init
- ajouter 1 à Index dans le bloc update

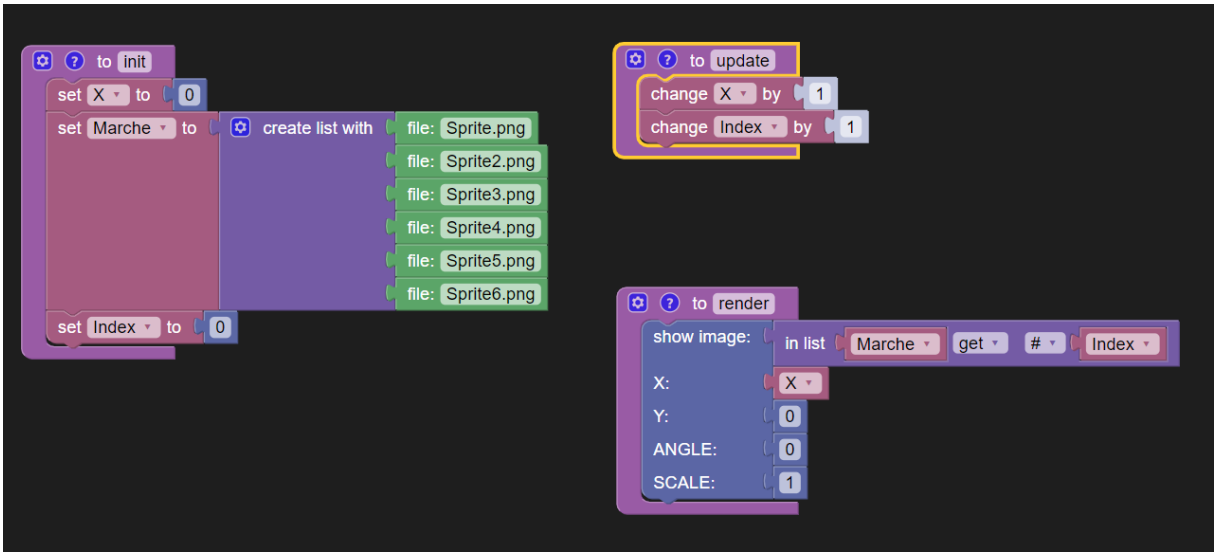
Pour le reste, nous continuerons ensemble car ce sera nouveau.

Je vous laisse faire ces premiers changements avant de continuer. Cela doit être de plus en plus intuitif pour vous, non ?

Bien alors continuons.

Déplacez le dernier bloc Marche que nous avons ajouté et posé le quelque part dans l’espace de travail.

Dans la catégorie « Lists » allez chercher le bloc « in list » et ajouté le à la place du bloc Marche que nous avons déplacé puis dans le premier emplacement de ce bloc, remettez le bloc Marche que vous aviez déposé sur le bureau et dans le second emplacement ajoutez la variable Index.



Oui, j’ai réorganisé mes blocs pour que mon code soit plus lisible. Organisez votre espace de travail comme vous le souhaitez afin de vous y retrouver le plus facilement possible.

Lancez votre programme pour voir ce qu’il se passe

Le petit bonhomme avance et disparait…

Pourquoi ? Je vous laisse réfléchir.

Et bien nous utilisons Index. Il est initialisé à 0 puis s'incrémente de 1 à chaque boucle. Hors notre animation n'a que 6 emplacement. Lorsqu'Index arrive à 6, le programme ne sait plus quoi afficher… Il n'affiche donc rien… logique… un programme ne fait que ce que vous lui dite de faire…

On va donc rajouter un autre élément fondamental en programmation : un test.

On va demander à notre programme de tester si la valeur d'Index vaut plus que 5, de la remettre à 0

Pour cela, ouvrons la catégorie « Logic » et faisons glisser un bloc « if » à la fin du bloc « update »

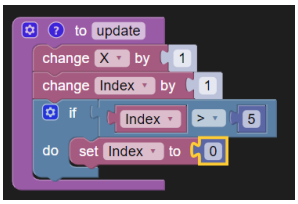
Pour le paramètre du haut, retournons dans « Logic » et prenons ce bloc :

Paramétrez le pour faire le test « Index > 5 » (Pour changer le = en > cliquez juste dessus et choisissez le test désiré)

Pour le paramètre en dessous, nous utiliserons le bloc set pour remettre Index à 0.

Note pour obtenir le bloc avec la valeur 0, vous pouvez aller le chercher dans la catégorie et le paramétrer ou tout simplement cliquer avec le bouton droit sur le bloc que nous avons déjà ajouté et choisir « Dupliquer » dans la liste déroulante. Cette option est très pratique et nous aurions déjà pu l'utiliser plusieurs fois. Mais je ne vais pas vous donner toutes les astuces en une fois… 😊

Une fois que vous l'aurez fait, vous devriez obtenir ce bloc :



Lancez votre programme.

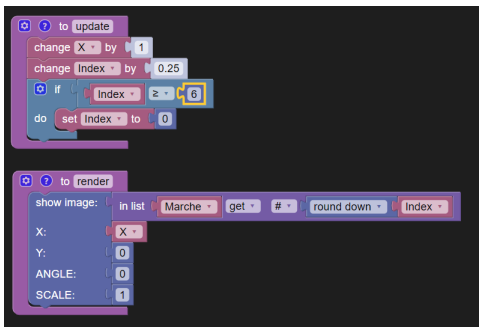
C'est mieux non ?

Bien, bien mais l'animation est trop rapide pour moi… que puis-je faire ?

Non, non, je ne vous donne pas la solution tout de suite, essayez…

Oui, il faut changer la valeur de l'incrément d'Index dans le bloc update. Mais ce n'est pas tout, il faut également changer notre test pour tester si Index >= 6 afin d'afficher la 5ème image tout le temps où elle doit l'être et utiliser la fonction mathématique round. Je vous laisse faire les modifications.

Vous devriez obtenir ceci :



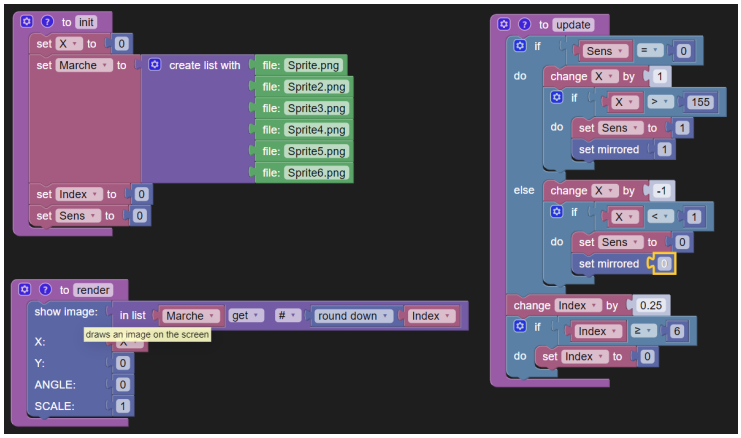
Bien maintenant nous voudrions que notre personnage change de sens et revienne en arrière lorsqu'il atteint le bord droit de l'écran…

Il nous faudra donc tester la valeur de X et nous allons ajouter une variable qui nous indique dans quel sens l'on va. Quand le sens vaudra 0, on ira vers la droite jusqu'à ce que l'on atteigne le bord droit. A ce moment-là, on changera le sens (il passera à un) et on utilisera le bloc set mirrored qui permet d'inverser horizontalement le sprite.

Essayez de le faire tout seul en cherchant un peu dans les possibilités de l'interface pour y arriver.

C'est toujours bon de chercher un peu, on apprend toujours des choses et en plus, on retient mieux.

Que vous y soyez arrivé ou non, vous devriez obtenir un programme a peu près semblable à celui-ci :



Et voilà, ne personnage reste à l'écran et vous savez maintenant utiliser les tests.

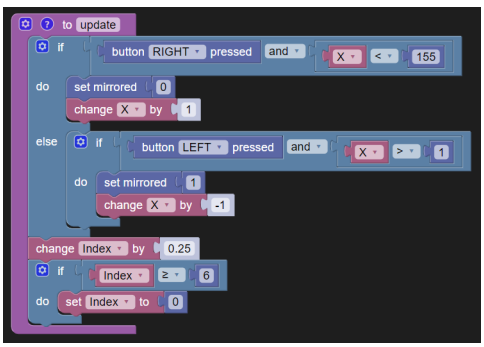
Mais il nous reste une chose importante à savoir utiliser : la gestion des touches

Contrôlons notre personnage à l'aide des touches.

Notre but : modifier notre programme pour que le personnage aille à droite si l'on appuie sur la touche droite et que l'on n'a pas encore atteint le bord droit et bien entendu on ira à gauche si on appuie sur la touche gauche et que l'on n'a pas déjà atteint le bord droit.

Nous allons utiliser le bloc « button » qui se trouve dans la catégorie « MicoJS » et modifier un peu nos conditions pour ne permettre le déplacement que lorsque l'on appuie sur la touche et que le déplacement est autorisé.

Essayez de le faire par vous-même avant de regarder la solution ci-dessous :



C’est déjà la fin de cette fiche mais avec ce que nous avons vu ensemble, vous pouvez déjà faire pas mal de tests.

Vous pouvez par exemple essayer de créer une animation qui se produise lorsque l’on appuie sur le bouton A, une autre lorsque l’on appuie sur le bouton B. Vous pouvez vous ajouter des contraintes, telles que l’on ne puisse rien faire d’autre par exemple que d’attendre la fin de l’action lorsque l’on a appuyé sur le bouton A. Manipulez, testez, fixez vous des objectifs et amusez-vous, vous avez déjà pas mal de choses. Dans la prochaine fiche, on abordera la gestion de ‘tirs’ et de collisions…