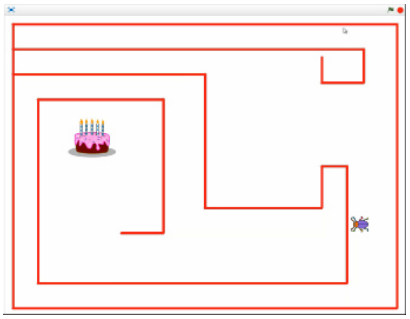
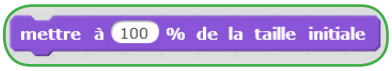
Le labyrinthe

→ Ouvrez le fichier "Labyrinthe.sb2"

L'objectif est ici de contrôler un lutin au clavier et de le faire évoluer dans le labyrinthe que vous avez construit lors d'une précédente activité. Vous aviez dessiné un labyrinthe, et vous aviez sauvegardé le lutin sous le nom "Laby". Nous l'utiliserons dans l'exercice 2, comme le montre la vidéo ci-dessous.



« Mais comment choisir une taille précise pour un lutin ?»



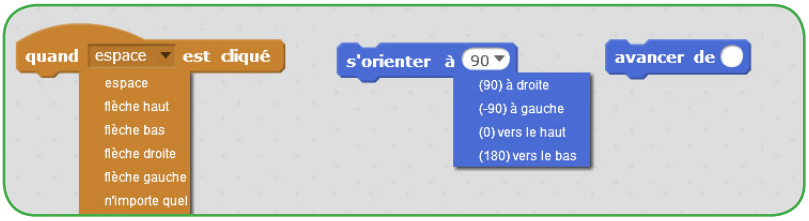
Ce bloc permet de réinitialiser la taille d'un lutin.
Mais à quelle taille correspond ce 100% ? Pour le savoir, il faut aller chercher l'information dans l'onglet "Costumes" du lutin. Vous pouvez y lire les dimensions du lutin: 85x76, c'est à dire que le lutin est inclus dans un rectangle de 85 unités de longueur sur 76 unités de largeur.
Si 100% correspond à 85 unités, il est alors facile de calculer grâce à la proportionnalité le pourcentage nécessaire pour obtenir un lutin de 20 unités de longueur.

Exercice 1

Les couloirs du labyrinthe ont une largeur minimale de 30. Il faut donc que le lutin soit d'une taille inférieure à 30 pour qu'il puisse à coup sûr emprunter ces couloirs. On choisira donc **une taille de 20**. Vous connaissez déjà une méthode pour réduire la taille d'un lutin, mais elle ne vous permet malheureusement pas de connaître la nouvelle taille obtenue.

→ Calculez ce pourcentage (arrondi à 1%) et appliquez le au lutin au lancement du programme.

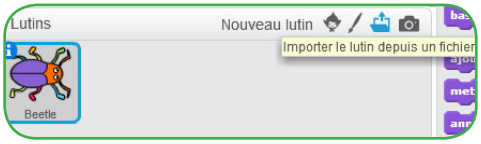
→ Utilisez les blocs ci-dessous (plusieurs fois) pour que le lutin se déplace dans les quatre directions (haut,bas,droite,gauche), lorsque l'on clique sur les quatre flèches. Vous programmerez un déplacement de 5 unités à chaque appui de touche.



Exercice 2:

→ Importez votre lutin "Laby" , puis exécutez le programme.

→ Ajoutez un bloc au début du programme pour placer le lutin insecte à un endroit de votre choix où il ne touche pas les parois.



→ Cherchez dans la bibliothèque le lutin nommé "Cake", et placez le dans le labyrinthe, en le redimensionnant grossièrement pour qu'il ne touche pas les parois. Placez le évidemment à un endroit que le joueur pourra rejoindre.

Vous allez à présent vous occuper des collisions avec les parois. Vous utiliserez l'algorithme suivant:
« A chaque fois que le lutin avance de 5, on teste s'il touche la couleur du labyrinthe. Si c'est le cas, le lutin recule de 5 et crie "Aïe !" pendant une seconde ».
→ En utilisant les blocs ci-dessous, modifiez vos quatre scripts de déplacement pour empêcher le lutin de traverser les parois.

→ En suivant une logique identique, modifiez vos quatre scripts de déplacement pour que le lutin réagisse au contact du gâteau en disant "Miam ! Miam !" pendant deux secondes.

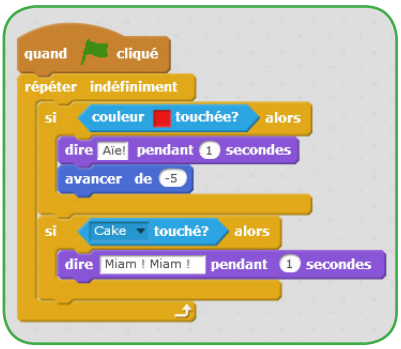
« Mais comme le gâteau a plusieurs couleurs, comment faire ?»

Effectivement, le gâteau ne peut être détecté grâce à sa couleur.
Il faut ici utiliser un autre bloc:

Ce bloc permet de tester le contact avec un autre lutin, avec le bord de la scène ou avec le pointeur de la souris.

Exercice 3:

Vous pouvez remarquer que l'on effectue plusieurs fois les mêmes tests dans ce programme.
Il est possible de simplifier le code en utilisant **une boucle de contrôle.**
→ Supprimez tous les tests de vos quatre scripts de déplacement.
→ Ajoutez le script ci-contre.



Ce script se lance au début du programme.
Cette fois, on va tester le contact du lutin avec les parois ou le gâteau de manière continue, même si l'utilisateur ne fait rien.
C'est pour cela qu'il y a une "boucle infinie": on répète les tests jusqu'à ce que l'utilisateur arrête le programme, et non pas uniquement quand celui-ci appuie sur une touche du clavier.

« Mais finalement, les deux méthodes sont identiques dans leur résultat...»

Oui... et non !
La boucle de contrôle a l'avantage de rendre plus lisible le code: moins il y a de blocs, plus clair sera le programme. Cependant, cette méthode a un défaut: les test sont effectués tout le temps, même quand ce n'est pas nécessaire. Cela a pour conséquence d'utiliser de façon inutile le processeur de l'ordinateur.
En utilisant le gestionnaire de tâche de Windows (ctr+alt+sup), on a accès aux performances de la machine:
-Sans boucle: Scratch 2.exe (32 bit) 1,2%
-Avec boucle: Scratch 2.exe (32 bit) 28,5%

En fonction de la complexité du programme ou du processeur, cela pourrait avoir pour conséquence de faire « ramer » l'ordinateur.