Scratch : Jeu de « cache-cache »

Le jeu que nous allons créer ici est une sorte de « cache-cache » : un objet va disparaitre, et en cliquant sur l'écran nous allons savoir à quel point nous sommes proches de l'objet en question, le but étant, par essais erreur, de cliquer suffisamment près de l'objet.

Ce programme fait appel à beaucoup de notions récurrente en programmation : variables, conditions, boucles, etc. Pour bien prendre en main le code, il est recommandé de s'être familiarisé au préalable à ces notions. Il est également possible de simplement recopier le code mais l'apport en terme d'apprentissage et de réflexion est plus limité.
Nous ferons notamment aussi appel au théorème de Pythagore, encore une fois il est possible de juste donner les instructions, mais c'est mieux si les participants connaissent déjà cette notion.
Il s'agit d'un guide étape par étape, avec quelques commentaires pour illustrer les avantages (ou inconvénients d'ailleurs) de chaque choix fait lors de la création de ce code. Le programme obtenu à la fin fonctionne, mais n'est pas la seule solution pour arriver à ce résultat.

Si vous souhaitez faire ce programme avec un groupe de jeune, dépendant du temps que vous voulez y consacrez, il convient peut-être mieux de leur présenter chaque étape, de les laisser réfléchir puis de reprendre ensemble les différentes solutions trouvées et d'en choisir une.

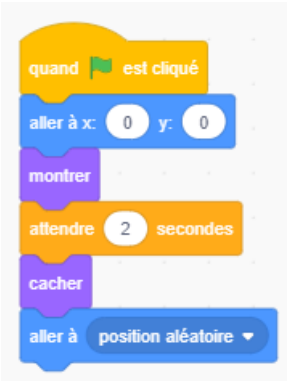
N'hésitez pas non plus à changer l'ordre des étape si cela vous semble pertinent (par exemple si vous voulez calculer la distance entre l'objet et la souris avant de faire disparaitre l'objet), ou à temporairement sortir des blocs du code pour tester une fonction à la fois.

1. Le jeu de base

1. Cacher le sprite

La première chose à faire pour retrouver un objet disparu dans ce programme est de faire disparaitre le dit objet. Pour rappel un « sprite », ou « lutin », est un objet programmable dans scratch. Vous pouvez les modifier ou en ajouter dans la zone dédiée en bas à droite de l'écran.

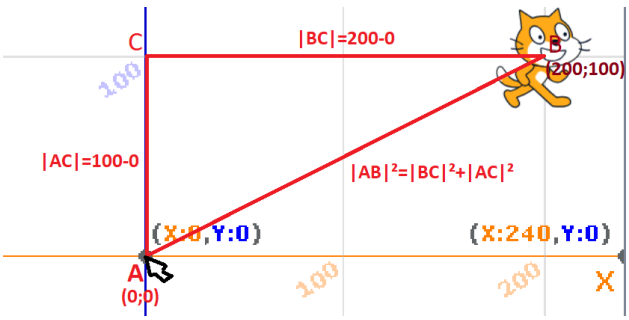
Vous pouvez déjà modifier le sprite du chat si vous souhaitez tout de suite travailler avec le sprite « earth » (inclus de base dans scratch).



Pour démarrer le programme, insérer n bloc évènement, par exemple « quand le drapeau verts est cliqué », puis initier la séquence. C'est optionnel, mais dans le code de droite on commence par envoyer le sprite au centre de l'écran (le bloc « aller à x :0 y :0 ») afin de le montrer (bloc « montrer »), juste pour savoir ce qu'on cherche.
On veut ensuite cacher le sprite avant de le déplacer (« aller à position aléatoire »). Si on intervertit ces blocs, on voit en fait où le sprite se déplace avant qu'il disparaisse.

Pour rappel les blocs sont organisés en catégorie. La couleur des blocs dépend de la catégorie à laquelle ils appartiennent. Si vous ne trouvez pas un bloc, regardez sa couleur, et chercher celle qui correspond dans la bibliothèque (les catégories ont également des noms assez parlant, par exemple pour déplacer sprite, il semble logique d'aller chercher dans la catégorie « mouvement »).

2. Calculer la distance



Pour retrouver la terre, nous avons dit que nous allions cliquer sur l'écran. En fait nous allons calculer la distance entre l'endroit où nous cliquons et la position du sprite.
Pour rappel, les Sprite se déplacent dans un repère orthonormé. On va donc calculer la différence de position sur les axes x et y. Par exemple sur l'image de droite, le sprite se trouve en en (200 ;100), et on clique avec la souris en (0 ; 0). Le sprite est donc 200 unités plus à droite, et 100 unités plus haut. Il serait possible de simplement comparer les distances des deux objets (souris et sprite), mais ce serait alors vraiment simple de retrouver le sprite.
Grâce à Pythagore on va pouvoir calculer la distance « réelle »on va en fait calculer l'hypoténuse d'un triangle, dont les 2 autres côtes ont une longueur égale à la différence de coordonnées entre la souris et du milieu du sprite.
Nous allons donc soustraire els coordonnées du sprite et de l'endroit cliqué par la souris (attention de bien soustraire les x et les y correctement).

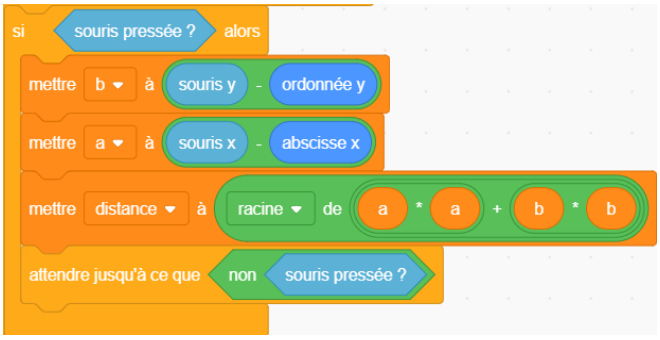
La création de deux variables comem ci-dessus n'est pas nécessaire mais facilite la lisibité du programme.
Une fois a (x de la souris – x du sprite) et b (y de la souris – y du sprite) on peut faire la formule complète pour calculer une nouvelle variable : « distance ».

Les blocs ci-dessus traduisent bien la formule :

$$\text{Distance} = \sqrt{((a*a)+(b*b))}$$

Remarque : vous pouvez faire la soustraction pour définir a et b comme vous le souhaitez. En effet ce n'est pas grave si la valeur obtenue est négative, car on en prendra le carré et un carré (quand on travaille avec des nombres réels) est nécessairement positif.

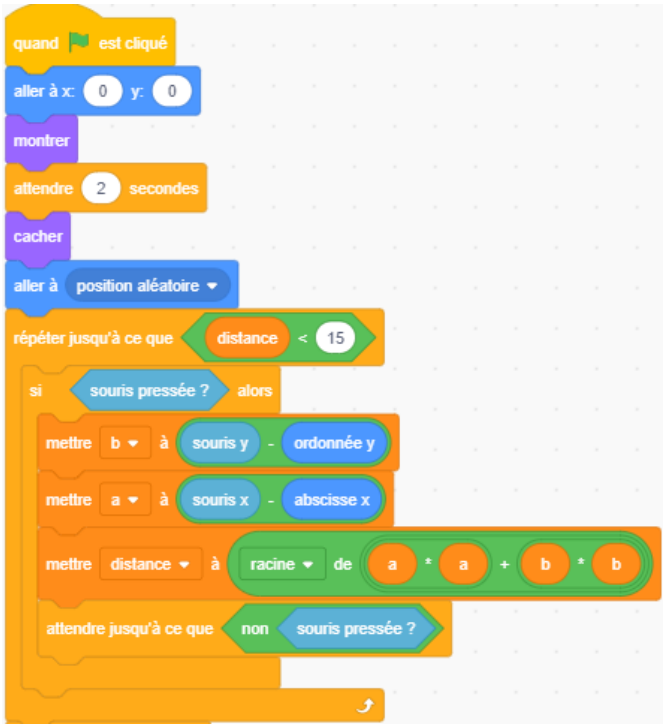
Si on imbriquait simplemnt cette série de bloc dans le code, la dstance serait définie une fois au lancement du programme. Nous allons donc dans un premier temps définir quand la calculer (quand la souri est pressée) et surtout faire en sorte qu'on puisse recommencer.
Première chose : insérez la série de bloc pour calculer la distance dans un condition et choisissez « souris pressée ? » comme condition.



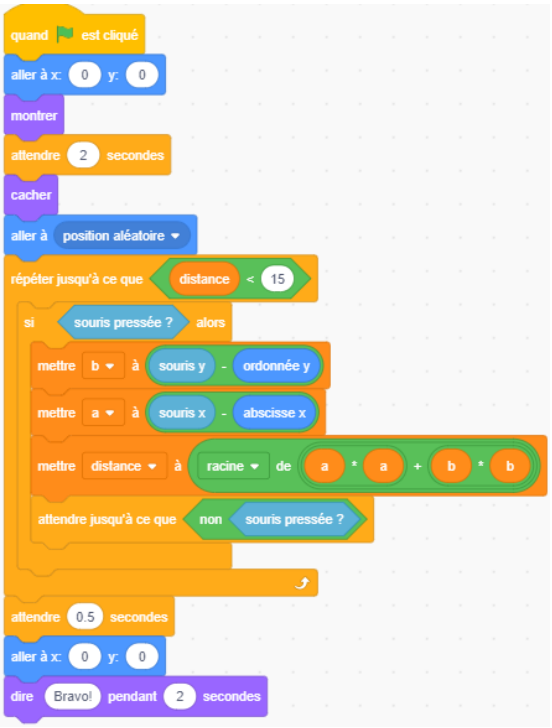
Insérez à la fin de cette séquence un assemblage de bloc « non « souris pressée » », cela servira a ne pas permettre de juste garder sa souris enfoncée. En effet sans ce dernier, le code sera définit en permanence tant que le clic de la souris est gardé enfoncé.

Enfin, insérer le tout dans une boucle « répétez jusqu'à ce que ». Là, en tant que condition, insérez un bloc comparatif, pour établir une condtition sur la variable distance. Dans notre code nous choisissons « distance < 15 », mais il y a plusieurs façons différentes de procéder. Veillez juste à prévoir une marge. Par exemple dans notre cas nous avons choisis 15 car si on mettait « distance = 0 » il faudrait cliquer

exactement sur le pixel auquel correspond le centre du sprite recherché.
Vous devriez avoir quelque chose qui ressemble à ceci :



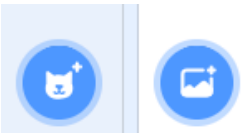
3. Gagner



Le code ci-dessus s'arrête une fois le sprite trouvé.

Il faut maintenant voir ce qu'il se passe quand le sprite est trouvé. Nous allons commencer simplement en faisant apparaitre le sprite une fois celui-ci trouvé. Vous pouvez aussi par exemple le faire retourner au centre d l'écran et lui faire dire quelque chose pour différencier cette animation de celle du début.
Vous pouvez ajouter autant

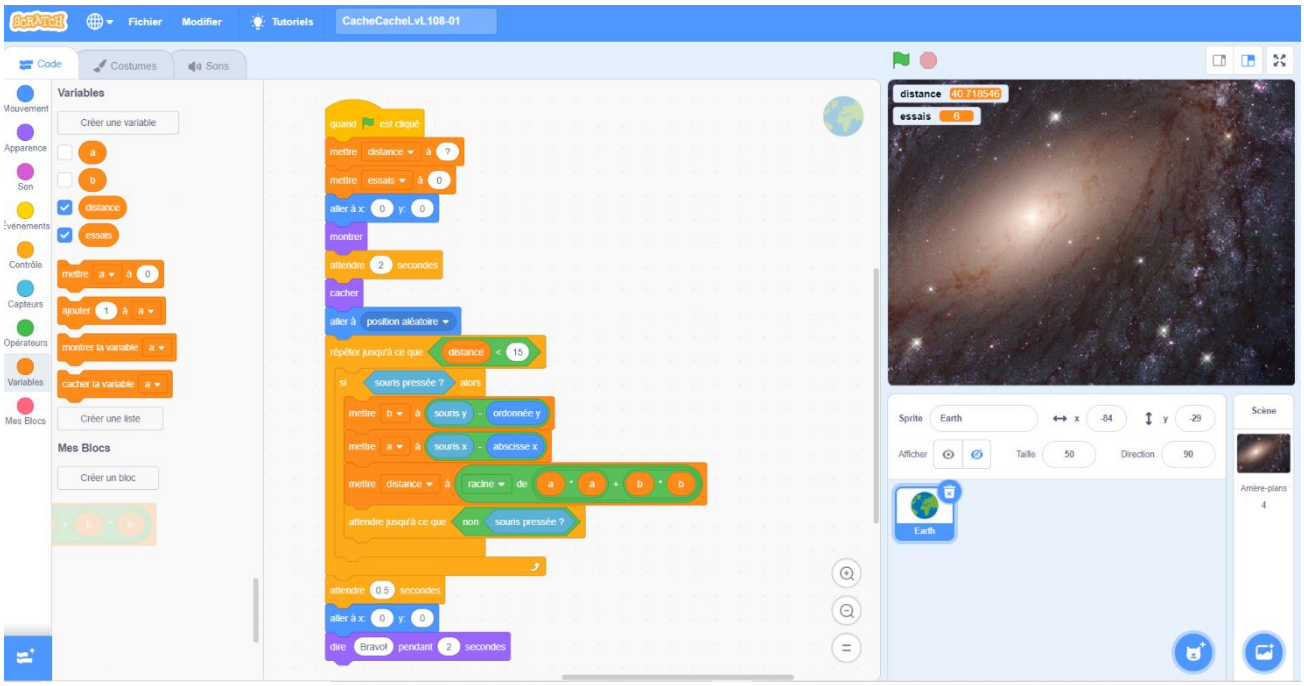
4. Détails (attention certains sont importants !)



A priori votre programme fonctionne. Vous pouvez changer le sprite et le fond si ce n'est déjà fait, ceux de la photo en début de document sont inclus de base dans le programme. Pour rappel il faut utiliser les puces bleues en bas des zones de sprite et de fond.

Il y a également quelques ajustements à faire pour éviter tout problème. Par exemple si vous lancez le code alors que votre variable distance était déjà inferieure à la condition que vous aviez mise dans la boucle, on en sort directement et on passe dans l'animation de victoire.
Insérez un bloc « mettre ma variable à et mettez « distance » à une valeur haute ou à « ? » (mettre un espace pour la laisser vide ne marche pas, il considère la valeur comme étant zéro, et si distance=0, dépendant de ce que vous avez mis dans la boucle cela pourrait causer des problèmes).
Vous pouvez créer une variable « essais » et également insérer un bloc « mettre essais à 0 » avec le bloc « mettre distance à ? ».
Ainsi à chaque fois que le programme est lancé, ces 2 variables se réinitialisent.

Si vous décider d'inclure une variable essais, incorporez également un bloc « ajouter 1 à essais » dans la boucle « si souris pressée » pour augmenter le compteur à chaque clic de souris. Le but du jeu serait alors de trouver la terre en le moins de clics possibles.


Fichier
Modifier
Tutoriels
CacheCacheLvL108-01
Code
Costumes
Sons
Mouvement
Apparence
Son
Événements
Contrôle
Capteurs
Opérateurs
Variables
Mes Blocs
Variables
Créer une variable
a
b
distance
essais
mettre a à 0
ajouter 1 à a
montrer la variable a
Cacher la variable a
Créer une liste
Mes Blocs
Créer un bloc
quand est cliqué
mettre distance à ?
mettre essais à 0
aller à x: 0 y: 0
montrer
attendre 2 secondes
cacher
aller à position aléatoire
répéter jusqu'à ce que distance < 15
si souris pressée ? alors
mettre b à souris y - ordonnée y
mettre a à souris x - abscisse x
mettre distance à racine de a * a + b * b
attendre jusqu'à ce que non souris pressée ?
attendre 0.5 secondes
aller à x: 0 y: 0
dire Bravo! pendant 2 secondes
distance 40.718546
essais 6
Sprite Earth
x -84
y -29
Afficher
Taille 50
Direction 90
Earth
Scène
Arrière-plans 4

2. Améliorer le programme

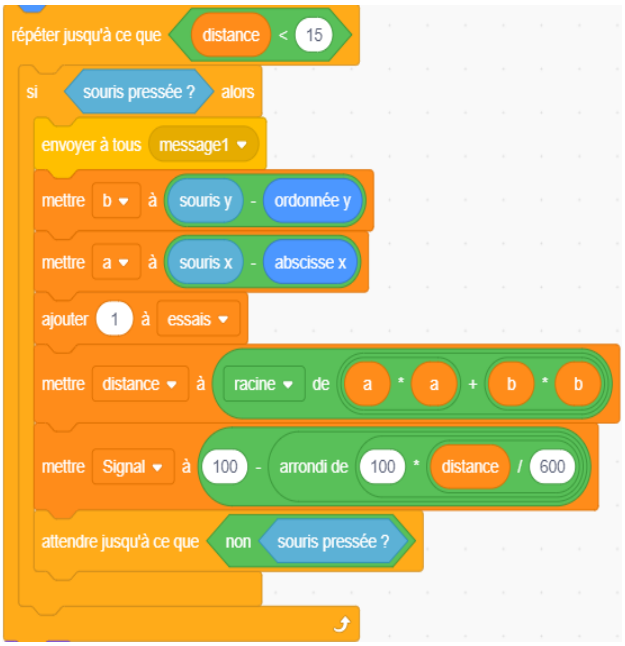
Si le temps le permet il y a beaucoup de choses qui peuvent être modifier pour améliorer le programme. Il est possible de rajouter des éléments de décors, des effets visuels, ou encore des fonctions supplémentaires. Il est aussi possible de combiner le tout.

1. Convertir la distance en « signal ».

Plutôt que de calculer une distance, nous allons simuler un signal. C'est plus sensé dans un scenario comme le nôtre.

On peut soit renommer et changer la formule de la variable « distance », soit créer une nouvelle variable « signal » et la calculer sur base de « distance ».
La seconde option est plus lisible mais rajoute une variable. A vous de voir.

Pour l'exprimer en pourcents il faut diviser « distance » par sa valeur maximum possible (c'est en la diagonale de l'écran, qui fait 480 sur 360 pixels, la distance maximale est donc la racine carrée de 480^2+360^2, soit 600), et multiplier par 100. Il faut ensuite soustraire le résultat à 100. Ainsi si on clique pile sur sprite, on aura une distance de 0, ce qui fera 100-0=100.



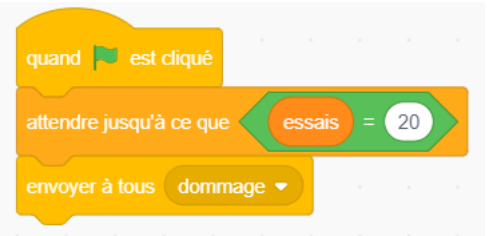
Le bloc « arrondi » n'est pas obligatoire, mais rend l'interface plus lisible (par défaut le programme affiche 6 décimales, ce qui allonge la case).

2. Limiter le nombre d'essais

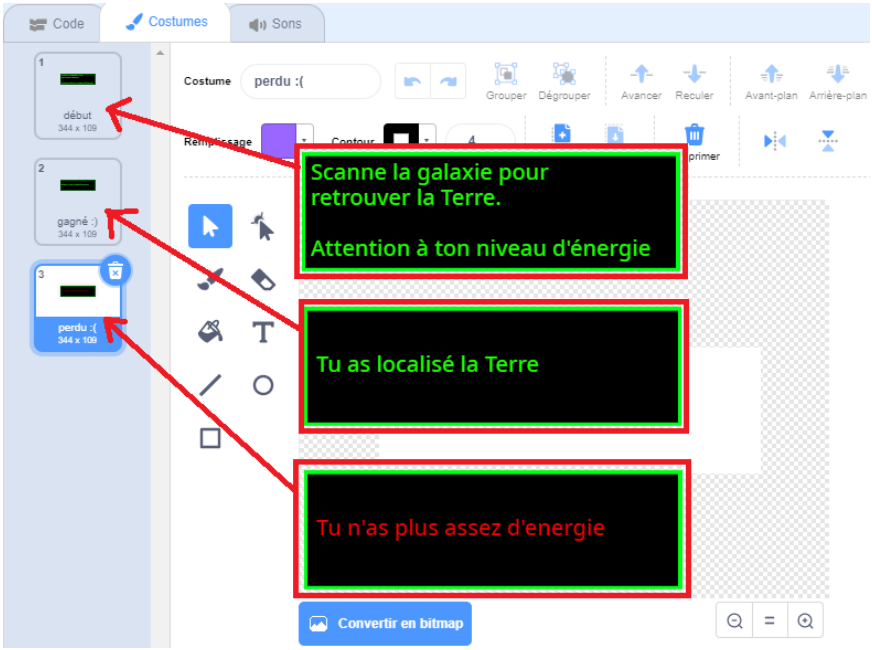
Si vous voulez introduire une notion de vie, il suffit de se baser sur la variable essais que nous avions déjà préalablement établie.

Ajoutez simplement un bloc « ajouter 1 à essais » dans la boucle « si souris pressée ? alors », et nous allons ensuite définir une action lorsque cette variable arrive à un certain nombre.

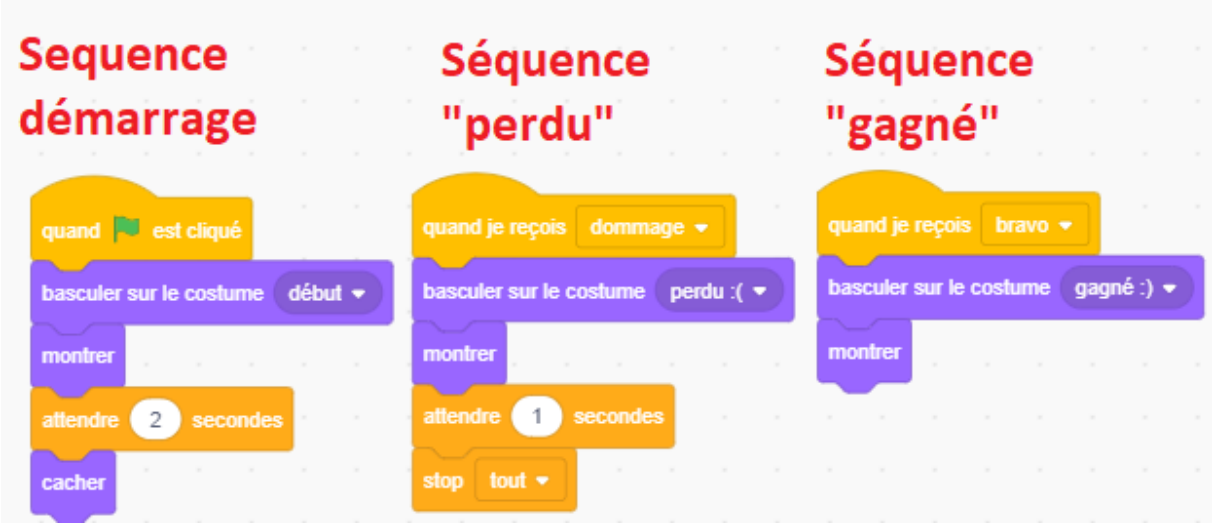
Ajoutez un nouveau bloc « quand le drapeau vert est cliqué » et imbriquez-y un bloc « attendre jusqu'à … ». La condition sera une comparaison avec la variable essais, et sous ce bloc ajoutez un « envoyer message 1 » (vous pouvez le nom du message du moment que les autres blocs que vous utiliserez fassent appel au même message), comme sur l'image de droite par exemple.



Une fois le nombre d'essais maximum définit comme ci-dessus, vous pouvez créer un nouveau sprite, dont vous pouvez modifier les costumes pour en faire un message de fin. Vous pouvez d'ailleurs en profiter pour créer plusieurs costumes si vous voulez par exemple faire un message au lancement du programme ou quand vous trouvez la terre.



Le code du sprite est assez simple, imbriquez les blocs que vous souhaitez exécuter sous un bloc événement « quand je reçois un message » (avec le bon message). Profitez-en pour créer la séquence du début et de « victoire ». Elles iront toutes dans la zone de code du même sprite.



3. Personnaliser l'interface

Les éléments sur la gauche de l'écran ne sont que des éléments visuels. LA plupart sont simplement directement dessinés sur la scène.
La fenêtre indiquant le « signal » est indiquée par défaut sur l'écran, il suffit de cliquer dessus pour en modifier l'apparence.
Il y a également la jauge de vie qui nécessite plus d'attention. Il existe peut-être des sprites tout faits sur internet, mais il a fallu dessiner des costumes pour chaque vie. Dans notre cas par exemple nous avons 20 essais, il y a donc 20 costumes différents. Notez qu'il est possible de faire appel à des variables pour choisir un costume, nous avons par exemple dit au sprite de la jauge de vie de se mettre sur les costume « essais ». En ayant renommé les costumes avec des chiffres de 1 à 20, si on a 15 essais par exemple, le spirte va automatiquement se mettre sur le costume « 15 ».

Le reste n'est globalement que du dessin. Veillez juste à adapter le code, si vous dessinez une sorte d'interface comme sur l'image, la terre ne devra pas aller dans cette zone : plutôt que d'employer un bloc a »aller à position aléatoire, il faudra définir un x et un y aléatoire, mais compris dans la zone accessible de l'écran.

Vous trouverez ci-après les sprites associés à leurs code respectif. Il y a moyen d'en ajouter d'autres pour faire joli mais leur code sera alors relativement simple.

```
quand [drapeau] est cliqué
attendre jusqu'à ce que (essais = 20)
envoyer à tous dommage

quand [drapeau] est cliqué
mettre Signal à 0
mettre essais à 0
aller à x: 30 y: 0
mettre la taille à 50 % de la taille initiale
montrer
répéter 2 fois
    attendre 1 secondes
    tourner ↻ de 15 degrés
    attendre 1 secondes
    tourner ↺ de 15 degrés
répéter 5 fois
    ajouter -10 à la taille
    attendre 0.1 secondes
cacher
mettre distance à 500
mettre x à nombre aléatoire entre -180 et 240
mettre y à nombre aléatoire entre -180 et 180
```

```
répéter jusqu'à ce que (distance < 15)
    si (souris pressée ?) alors
        envoyer à tous message1
        mettre b à (souris y - ordonnée y)
        mettre a à (souris x - abscisse x)
        ajouter 1 à essais
        mettre distance à (racine de (a * a + b * b))
        mettre Signal à (100 - (arrondi de (100 * distance / 552.18)))
        attendre jusqu'à ce que (non (souris pressée ?))
montrer
répéter 5 fois
    ajouter 10 à la taille
    attendre 0.2 secondes
attendre 0.5 secondes
glisser en 1 secondes à x: 30 y: 0
envoyer à tous bravo
```

Ce lutin a 3 costumes differents. Il s'agit en gros de zone de texte, il y en a un pour le lancement du programme (le costume « début »), un quand on gagne (costume « gagné ») et un en cas de défaite quand le nombre d'essais est dépassé (costume « perdu »)

Ce lutin a 21 costumes, 1 pour chaque nombre d'essais. Plus le nombre d'essais est élevé, plus la jauge se vide. Nous avons mis 20 essais, le 21eme étant « 0 ».