

# Jeu du pendu



## Présentation

Le joueur doit retrouver un mot pris au hasard, en indiquant successivement les lettres qu'il pense contenir afin de le reconstituer. Il n'a droit qu'à 6 erreurs, sans quoi un malheureux cow-boy sera pendu !

### Objectifs pédagogiques

Ce projet aura pour but d'écrire un programme en travaillant :

- les boucles (répétitions)
- les structures conditionnelles (si... alors...sinon...)
- les variables
- les listes
- les chaînes de caractères (\*)

### Organisation et évaluation

Vous travaillerez seul(e) ou en binôme.

Chaque groupe disposera :

- du présent **guide** explicatif,
- d'un **programme Scratch** de démarrage,
- d'un ensemble de fichiers présents dans le dossier « **Compléments élèves** », et qui pourront être utiles pour le réalisation des objectifs complémentaires.

L'évaluation de votre travail tiendra compte :

- du bon fonctionnement du programme
- des éléments de développement personnalisés que vous aurez ajoutés
- de l'implication de chacun (= évaluation individuelle, pas en binôme)

Les meilleurs projets pourront être publiés sur le site du collège.

A chaque fin de séance, enregistrez votre travail sur :

- la zone de partage de données de la classe
- la zone personnelle de chacun (important en cas de problème sur la zone de partage)
- votre clé USB

---


(\*) Chaîne de caractère = mot formé par une succession de lettres, chiffres, ou autres caractères

Pour chaque début de séance, il vous appartient d'apporter votre travail sur clé USB et/ou par mail et/ou en l'ayant déposée préalablement sur votre zone personnelle au collège.

### **Durée**

Ce projet s'étendra sur 6 séances et le travail devra être rendu à l'issue de la dernière séance (compter 1 séance pour chacun des 4 objectifs principaux, et 2 séances pour les objectifs secondaires).

### **Scénario**

Au clic sur , un mot est choisi aléatoirement par le programme (la liste de mots est incluse dans le programme Scratch de démarrage).

**Abby** annonce qu'un mot est à découvrir en moins de 7 erreurs.

Ce mot s'affiche, constitué de « \_ » représentant les lettres à trouver.

On demande au joueur une lettre.

Si la lettre est correcte, elle s'affiche à la place du « \_ » qui la représente, sinon la **Potence** est dressée progressivement.

Puis on redemande une lettre jusqu'à ce que le joueur ait dévoilé toutes les lettres du mot à deviner (partie gagnante) ou que la **Potence** soit complétée (partie perdante).

Si la partie est perdante, **Abby** annonce le mot qu'il fallait trouver.

### **Réalisation**

Vous réaliserez ce programme en menant à bien 4 objectifs principaux successifs présentés dans la suite de ce document.

Pour compléter ce travail une fois les 4 objectifs principaux réalisés, vous choisirez un ou plusieurs objectifs secondaires parmi ceux proposés dans le sous-dossier « Objectifs secondaires » : le niveau de chaque objectif est indiqué (bronze, argent ou or).

## Données

Vous aurez besoin de 5 variables et de 4 listes :





- **Erreurs** *Variable chaîne de caractères*  
Contient toutes les lettres proposées par le joueur et qui ne sont pas présentes dans le mot à deviner. Pour une meilleure lisibilité, les lettres sont séparées par des espaces. Cette variable s'affiche dès la première erreur.
- **Fin du jeu** *Variable « oui » ou « non » (booléen)*  
Indique l'état de la partie. Elle doit être initialisée à « non » au début de la partie. On cesse de demander une lettre au joueur quand sa valeur passe à « oui ».
- **Mot affiché** *Variable chaîne de caractères*  
Comme son nom l'indique, représente le mot à trouver tel qu'il est affiché au cours de la partie. Initialement uniquement composé de « \_ \_ \_ \_ » (alternance de tirets du bas et d'espaces), il se complète avec les lettres du mot à deviner progressivement. Cette variable est visible tout au long de la partie. Pour une meilleure lisibilité, son affichage a été placé en mode « Grande lecture » (mode accessible dans la scène par clic droit sur la variable une fois visible).
- **Mot à deviner** *Variable chaîne de caractères*  
C'est le mot choisi par le programme dans le Dictionnaire.
- **Numéro de caractère** *Variable numérique*  
C'est un nombre qui indique le numéro du caractère dans une chaîne qu'on veut analyser caractère après caractère.
- **Alphabet** *Liste*  
Cette liste contient tous les caractères que le joueur est autorisé à saisir durant le jeu. Le coup n'est pas considéré comme perdant si le joueur saisit autre chose que l'un de ces caractères. Cette liste n'est normalement pas modifiée durant le jeu.
- **Dictionnaire** *Liste*  
Cette liste contient tous les mots que le programme pourra faire deviner, en choisissant l'un d'eux au hasard au début du jeu. Elle n'est normalement pas modifiée durant le jeu, mais peut être remplacée, en important par exemple un autre dictionnaire (dans la scène, clic droit sur **Dictionnaire** une fois visible).
- **Lettres déjà trouvées** *Liste*  
Cette liste contient toutes les lettres gagnantes trouvées par le joueur. Elle doit être vidée en début de partie, puis complétée au fil des bonnes réponses. Elle permet de reconstituer le **Mot affiché** durant la partie en analysant chaque caractère de **Mot à deviner** et en regardant s'il se trouve dans cette liste auquel cas on affiche la lettre et non plus « \_ ».
- **Lettres à découvrir** *Liste*  
Cette liste contient toutes les lettres de mot restant à découvrir.  
En début de partie, elle contient donc toutes les lettres du mot, et se vide au fur et à mesure que le joueur devine les lettres du **Mot à deviner**.  
Ainsi, **Fin du jeu** vaut « oui » quand **longueur de** **Lettres à découvrir** vaut 0.

Deux lutins sont également fournis :

- **Abby** : c'est essentiellement le script d'**Abby** que vous allez programmer. Elle a trois costumes (neutre, souriante, triste) qui pourront être utilisés selon les circonstances du déroulement du jeu.
- **Potence** : contient les images (costumes) qui pourront être affichées successivement au fil des erreurs. Le premier costume est totalement vide et le dernier représente l'ultime étape, quand le joueur a perdu la partie.

## Objectifs principaux

Votre travail va consister essentiellement à compléter successivement les 4 blocs :

<a href="#"><u>Objectif principal 1</u></a>		Dans ce bloc, on initialise listes et variables. Le <b>Mot à deviner</b> est pris aléatoirement dans le dictionnaire.
<a href="#"><u>Objectif principal 2</u></a>		Ce bloc, appelé au démarrage du programme et à chaque fois qu'une lettre du mot est découverte, permet de recalculer l'affichage de <b>Mot affiché</b> en haut de la scène.
<a href="#"><u>Objectif principal 3</u></a>		Ce bloc consiste à demander au joueur de façon répétitive une lettre et à l'analyser (coup gagnant ou coup perdant) avant de recommencer.
<a href="#"><u>Objectif principal 4</u></a>		Lorsque la partie est terminée, <b>Abby</b> affiche au joueur qu'il a gagné ou perdu, auquel cas elle lui annonce le <b>Mot à deviner</b> qu'il fallait découvrir.

⇒ Ne pas commencer l'objectif suivant tant que l'objectif en cours n'est pas terminé.



## Objectif principal 1

### Idee générale :

On initialise les listes et variables avant leur utilisation au cours du jeu.

### Application :

On vide les deux listes **Lettres à découvrir** et **Lettres déjà trouvées** à l'aide de l'instruction **supprimer l'élément tout de la liste** (sur Scratch 3, taper « all » à la place de « tout »).

On initialise également **Erreurs** (chaîne vide), et **Fin du jeu** (à « non ») à l'aide de **mettre à**.

On appelle **cacher la variable** de **Mot affiché** ainsi que pour **Erreurs** qui ne seront rendues visibles que lorsque la partie aura commencé.

On prend un mot au hasard dans **Dictionnaire** et on le mémorise dans la variable **Mot à deviner**. Pour cela, on pourra utiliser les instructions : **élément de Dictionnaire** (possibilité de choisir « au hasard » sur Scratch 2, taper « random » sur Scratch 3) et **mettre Mot à deviner à**.

⇒ Vérifier, en l'affichant temporairement pour tester, que **Mot à deviner** prend bien des valeurs différentes en cliquant plusieurs fois sur

Enfin, on va remplir la liste **Lettres à découvrir** en lui ajoutant toutes les lettres de **Mot à deviner** qu'on va parcourir caractère après caractère dans une répétition.

Pour ce faire, on commence par initialiser **Numéro de caractère** à 1 pour signifier qu'on commence au premier caractère.

Puis, on effectue une répétition dont le nombre d'itérations est égal au nombre de lettres de **Mot à deviner** :



⇒ On distinguera bien **longueur de** qui donne le nombre de caractères d'une chaîne de caractères, et **longueur de** qui donne le nombre d'éléments d'une liste.

Dans la répétition :

- On demande d'ajouter dans **Lettres à découvrir** le caractère **Numéro de caractère** de **Mot à deviner** : on aura besoin de **lettre de** et **ajouter à Lettres à trouver**.
- On passe au caractère suivant en augmentant **Numéro de caractère** de 1 (utilisation de **ajouter à**).

### Test :

Vérifier plusieurs fois en cliquant sur , que **Mot à deviner** change à chaque clic, et que **Lettres à découvrir** prend les bonnes valeurs.

définir Actualiser mot à afficher

## Objectif principal 2

### Idée générale :

En partant de **Mot affiché** initialisé à une chaîne vide, on parcourt chaque lettre de **Mot à deviner** dans une répétition, et on regarde si elle est contenue dans **Lettres déjà trouvées** ou non. Si c'est le cas, on ajoute la lettre à **Mot affiché**, sinon « \_ ».

### Application :

On commence par initialiser **Mot affiché** (chaîne vide), et mettre **Numéro de caractère** à 1 avant de commencer une répétition :

répéter longueur de **Mot à deviner** fois

Dans cette répétition, on pourra employer **Lettres déjà trouvées** contient  ? pour tester si une lettre est déjà contenue dans la liste, en s'inspirant de ceci :

si **Lettres déjà trouvées** contient  ? alors  
sinon

- Si la lettre est contenue dans **Lettres déjà trouvées**, on la rajoute dans **Mot affiché** suivie d'une espace (\*) (utilisation de **mettre** **Mot affiché** à  et de plusieurs **regroupe**  ).
- Sinon, on rajoute dans **Mot affiché** le tiret du bas (« \_ ») suivi d'une espace également.

Avant la fin de la répétition, ne pas oublier de passer au caractère suivant.

Après la répétition, montrer **Mot affiché** (utilisation de **montrer la variable** ).

### Test :

Pour tester le bon fonctionnement de ce bloc :

définir Initialisation

1. Double-cliquer sur **Initialisation** pour qu'un mot soit choisi dans le dictionnaire.
2. Montrer **Mot à deviner** (cliquer sur **montrer la variable** ).
3. Ajouter manuellement des lettres du mot choisi dans **Lettres déjà trouvées**.

définir Actualiser mot à afficher

4. Double-cliquer sur **Actualiser mot à afficher** pour vérifier que **Mot affiché** s'affiche correctement en fonction des lettres saisies dans **Lettres déjà trouvées**.

\* L'espace (caractère typographique) est un nom féminin.

### Objectif principal 3

#### Idée générale :

Dans une répétition qui aura lieu jusqu'à ce que **Fin du jeu** vaille « oui », on demande au joueur une lettre. Si cette lettre est dans **Mot à deviner** alors le coup est gagnant, sinon la **Potence** se dresse progressivement.

#### Application :

On crée une répétition jusqu'à ce que **Fin du jeu** soit égale à « oui », à l'intérieur de laquelle :

- On commence par **demander**  **et attendre**.
  - ⇒ Le fait de laisser libre le champ après « demander » permet d'éviter qu'une ligne de texte comportant la question n'empiète sur l'image d'arrière-plan, sans gêner la compréhension.
- On teste si la saisie est acceptable. Pour cela, il suffit de vérifier que **Alphabet** contient **réponse** en utilisant :




- Si la saisie est acceptable, on teste si **réponse** est contenue dans **Lettres à découvrir**, en utilisant :



- Si c'est le cas (lettre gagnante) :
  - i. on ajoute **réponse** à **Lettres déjà trouvées**,
  - ii. on appelle **Actualiser mot à afficher** pour redessiner **Mot affiché**,
  - iii. on retire **réponse** de **Lettres à découvrir** : voir comment faire dans [l'encadré de la page suivante](#),
  - iv. si **Lettres à découvrir** n'a plus d'éléments (**longueur de**  égale à 0) alors on met **Fin du jeu** à « oui ».
- Sinon (lettre perdante) :
  - i. on rajoute la **réponse** (+ espace) à **Erreurs** à l'aide de **regroupe**   (comme dans [l'objectif précédent](#)),
  - ii. on affiche **Erreurs** (utilisation de **montrer la variable** ),
  - iii. on appelle **envoyer à tous** **Lettre incorrecte !**  **et attendre** pour actualiser le dessin de la **Potence**,
  - iv. dans le lutin **Potence**, après changement de costume, si on atteint le dernier **costume #**, alors on met **Fin du jeu** à « oui ».

#### Test :

Cliquer sur  et vérifier que le jeu fonctionne bien jusqu'à s'arrêter de demander une lettre quand le mot est découvert ou bien que la partie est perdue.



Pour retirer **réponse** de **Lettres à découvrir**, remettre dans l'ordre les instructions suivantes :





#### Objectif principal 4

##### Idée générale :

En fin de partie, **Abby** réagit au fait que le joueur a gagné ou perdu. S'il a perdu, elle annonce le mot qu'il fallait trouver.

##### Application :

**Abby** se place vers la droite de la scène en regardant vers la gauche (changement de costume).

Si le joueur a gagné (cas où **longueur de** **Lettres à découvrir** ▼ vaut 0) alors elle sourit et réagit en conséquence.






Sinon, **Abby** se montre triste et annonce le mot qu'il fallait trouver.

Puis, après un court délai, une nouvelle partie peut commencer (utilisation de **envoyer à tous** **Démarrage du programme** ▼).

# Jeu du pendu

## Objectif secondaire « Alpha et Omega »


- ⇒ Pour aller plus loin, une fois les objectifs principaux réalisés.
- ⇒ Les objectifs secondaires peuvent être réalisés dans n'importe quel ordre.
  - ⇒ Les objectifs secondaires sont de niveaux différents
- ⇒ Tous les objectifs secondaires peuvent être réalisés dans un même projet.

<p>Dès le début de la partie, afficher la première lettre du mot, et la dernière.</p>	 <p>Niveau bronze</p>
<p style="text-align: center;"><b>Réalisation</b></p> <p>Modifier les blocs  et  pour ajouter l'exception de traitement lorsque  vaut 1 ou .</p>	

# Jeu du pendu

## Objectif secondaire « Chasse au doublon »






- ⇒ Pour aller plus loin, une fois les objectifs principaux réalisés.
- ⇒ Les objectifs secondaires peuvent être réalisés dans n'importe quel ordre.
  - ⇒ Les objectifs secondaires sont de niveaux différents
- ⇒ Tous les objectifs secondaires peuvent être réalisés dans un même projet.

<p>Durant la partie, si une lettre gagnante comme perdante a déjà été proposée, le faire annoncer par Abby sans que cela ne soit considéré comme une erreur.</p>	 <p>Niveau bronze</p>
<p style="text-align: center;"><b>Réalisation</b></p> <p>Créer une liste <b>Lettres erreur</b> qu'on videra dans <b>définir Initialisation</b> et dans laquelle on insérera chaque lettre incorrecte dans le bloc <b>définir Déroulement du jeu</b>.</p> <p>Modifier ensuite ce même bloc <b>définir Déroulement du jeu</b> pour ajouter, après avoir vérifié que la lettre saisie par l'utilisateur est dans <b>Alphabet</b>, un <b>si</b> <b>alors</b> <b>contient</b> <b>?</b> ou bien si <b>sinon</b> <b>Lettres erreur</b> <b>contient</b> <b>?</b> (en utilisant bien sûr la <b>réponse</b> donnée) : si tel est le cas, faire réagir <b>Abby</b> par un message, sinon poursuivre le jeu normalement.</p>	

# Jeu du pendu

## Objectif secondaire « Le bon caractère »

- ⇒ Pour aller plus loin, une fois les objectifs principaux réalisés.
- ⇒ Les objectifs secondaires peuvent être réalisés dans n'importe quel ordre.
  - ⇒ Les objectifs secondaires sont de niveaux différents
- ⇒ Tous les objectifs secondaires peuvent être réalisés dans un même projet.

<p><b>Adapter le jeu pour prendre en compte les mots comportant apostrophes ou tirets.</b></p>	 <p>Niveau bronze</p>
<p style="text-align: center;"><b>Fichier requis</b></p> <p><i>Dictionnaire (avec - et ').txt</i> : inclut les mots comportant ces caractères.</p> <p>Pour remplacer le dictionnaire par défaut par le dictionnaire complet, montrer <b>Dictionnaire</b>, puis, dans la scène, importer par clic droit la nouvelle liste de mots.</p>	
<p style="text-align: center;"><b>Réalisation</b></p> <p>Modifier les blocs  et  pour ajouter l'exception de traitement lorsque  vaut « - » ou « ' ».</p> <p>Durant la réalisation de cette mission, il pourra être utile d'utiliser un <b>Mot à deviner</b> tel que « CHEF-D'OEUVRE » afin de réaliser facilement des tests probants sans avoir à cliquer plusieurs fois sur  pour espérer obtenir un mot pertinent.</p>	

# Jeu du pendu

## Objectif secondaire « Au galop »

- ⇒ Pour aller plus loin, une fois les objectifs principaux réalisés.
- ⇒ Les objectifs secondaires peuvent être réalisés dans n'importe quel ordre.
  - ⇒ Les objectifs secondaires sont de niveaux différents
- ⇒ Tous les objectifs secondaires peuvent être réalisés dans un même projet.

Durant le jeu, un cheval se déplace de gauche à droite (puis recommence indéfiniment) de plus en plus vite au fur et à mesure que la Potence se dessine. Le cheval s'enfuit en hennissant si la partie est perdue.





Niveau argent

### Fichiers requis


[Cheval.sprite2](#) : inclut un lutin représentant un cheval comportant costumes (cheval en mouvement) et sons.



[Arbre.sprite2](#) : inclut un lutin représentant l'arbre de droite.



En cliquant sur  (Scratch 2) ou sur  (Scratch 3), importer le lutin **Cheval** et le lutin **Arbre**.

### Réalisation

Pour le lutin **Arbre**, insérer  pour s'assurer que l'arbre soit toujours en avant-plan et que le cheval donne l'illusion de passer derrière lui.

Pour que la vitesse du **Cheval** s'adapte à l'avancement de la **Potence** et coure de plus en plus vite au fil des erreurs, on pourra faire avancer le cheval de  combiné à  dans une même répétition.



A chaque nouvelle erreur, on joue le son du galop, sauf si la **Potence** est terminée : dans ce cas, le cheval hennit avant de disparaître définitivement derrière l'arbre.

Pour un plus bel enchaînement, faire en sorte que le cheval ait totalement disparu avant qu'**Abby** n'apparaisse pour dire si la partie est gagnée ou perdue.

# Jeu du pendu

## Objectif secondaire « Le bon mot »

- ⇒ Pour aller plus loin, une fois les objectifs principaux réalisés.
- ⇒ Les objectifs secondaires peuvent être réalisés dans n'importe quel ordre.
  - ⇒ Les objectifs secondaires sont de niveaux différents
- ⇒ Tous les objectifs secondaires peuvent être réalisés dans un même projet.

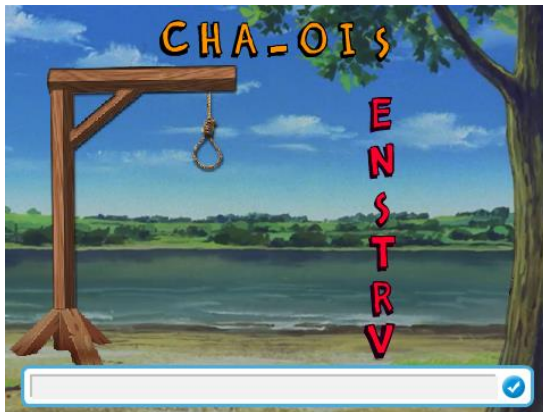
<p>Avant de commencer la partie, il est demandé au joueur s'il veut jouer contre l'ordinateur ou contre un autre joueur qui choisira lui-même un mot.</p>	 <p>Niveau argent</p>
<p style="text-align: center;"><b>Réalisation</b></p> <p>Pour demander au joueur le mode de jeu souhaité, il suffit de créer une liste nommée par exemple <b>Mode de jeu</b> et qui peut apparaître ainsi au clic sur  :</p> <div data-bbox="644 967 949 1120" style="border: 1px solid gray; padding: 5px; width: fit-content; margin: 10px auto;"><p style="text-align: center;">Mode de jeu</p><p>1 Avec l'ordinateur</p><p>2 Avec un(e) ami(e)</p><p style="text-align: center;">+ longueur: 2</p></div> <p>On interagit avec le joueur avec <b>demander</b> Choisissez votre mode de jeu (1 ou 2) et attendre puis, si la réponse est 2, on demande au joueur un mot.</p> <p>La difficulté consiste à vérifier que le mot saisi est valide (ne contient pas d'accents ou de caractères particuliers).</p> <p>Le plus simple peut consister à vérifier que le mot saisi est présent dans <b>Dictionnaire</b> mais cela exclut un grand nombre de mots que l'ami(e) pourrait souhaiter entrer.</p> <p>Pour vérifier que le mot saisi est correct, la meilleure façon est d'effectuer une répétition lettre après lettre, en utilisant une variable <b>Mot correct</b> de type booléen (« oui » / « non ») initialisée à « oui » avant la répétition. Si l'un des caractères n'est pas dans <b>Alphabet</b>, alors on bascule <b>Mot correct</b> sur « non », autrement on passe à la lettre suivante.</p> <p>A la fin de la répétition, si le mot est incorrect, alors on reprend du début, sinon la partie peut commencer.</p>	

# Jeu du pendu

## Objectif secondaire « L'attaque des clones »

- ⇒ Pour aller plus loin, une fois les objectifs principaux réalisés.
- ⇒ Les objectifs secondaires peuvent être réalisés dans n'importe quel ordre.
  - ⇒ Les objectifs secondaires sont de niveaux différents
- ⇒ Tous les objectifs secondaires peuvent être réalisés dans un même projet.

Utiliser des lutins représentant des caractères alphabétiques pour représenter le mot à chercher et les erreurs commises.





Niveau or

### Fichier requis

[Alphabet.sprite2](#) : inclut 29 costumes (26 lettres, tiret, apostrophe et tiret du bas).



En cliquant sur  (Scratch 2) ou sur  (Scratch 3), importer le lutin et le nommer **Mot**, puis le réimporter et nommer le nouveau lutin **Erreurs**.

### Réalisation

#### Idée générale :

Alors qu'on pourrait penser qu'il suffirait de créer autant de lutins que de lettres de l'alphabet, on se rend compte que cette hypothèse n'est pas réalisable dès lors que le mot à trouver contient plusieurs fois la même lettre. Il faut envisager une autre stratégie : l'utilisation des clones.

En effet, chaque lutin peut être dupliqué (cloné) durant l'exécution du programme, et chaque clone va se comporter exactement de la même façon que le lutin d'origine.

Ainsi, en dupliquant le lutin **Mot**, on peut représenter sur la scène n'importe quel mot souhaité (une lettre = un clone), le tout étant de faire apparaître le bon costume (= la bonne lettre) au bon endroit.

Durant le fonctionnement du programme, on a besoin de modifier certains clones spécifiques et pas d'autres, lorsqu'on veut transformer un « \_ » en « S » par exemple (cas où le joueur a découvert une lettre) : pour ce faire, l'envoi de messages ne suffit pas car tous les clones reçoivent le même message. Il


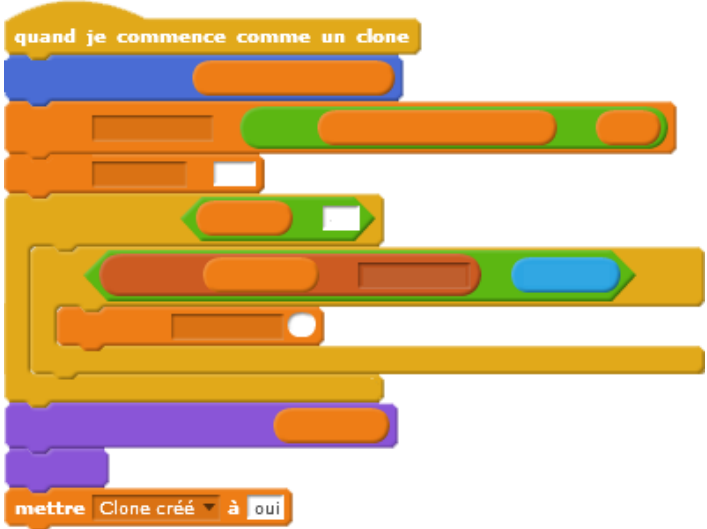


faut donc pouvoir identifier chaque clone pour savoir la lettre qu'il contient et ainsi réagir convenablement à la réception du message : c'est au moment de la création du clone qu'on va devoir définir ses propriétés.

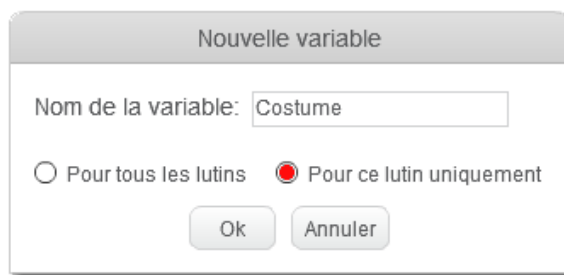
Enfin, il faut être conscient que lorsque, durant l'exécution du programme, on crée par exemple 2 clones du même lutin, on a non pas 2, mais 3 lutins identiques : celui d'origine, et ses deux clones. Dans ce qui suit, on considère que le lutin d'origine reste masqué tout au long de la partie et ne sert qu'à être cloné : on fait le choix de ne se servir que des clones.

Application :

La création d'un clone se fait au moyen de l'instruction **créer un clone de Mot**. Ceci lance la création du clone, mais il faut savoir que l'instruction placée juste après est susceptible d'être exécutée avant la fin de la création du clone en question, ce qui peut créer des problèmes de synchronisation. Pour éviter ce problème, il suffit de créer une variable **Clone créé** de type booléen (« oui » / « non ») et de l'utiliser ainsi :

Bloc appelant la création d'un clone	Création du clone
 <p>La variable <b>Clone créé</b> est initialisée à « non » et l'on attend qu'elle devienne « oui » avant de placer d'autres instructions.</p> <p>C'est à la toute dernière instruction de la procédure de création du clone que <b>Clone créé</b> est passée « oui » comme montré dans l'exemple ci-contre.</p>	

Pour créer une propriété de chaque clone, on sélectionne le lutin **Mot**, puis on crée une nouvelle variable pour ce lutin uniquement :



La variable **Costume** ainsi créée pourra prendre une valeur différente pour chaque clone de **Mot**.

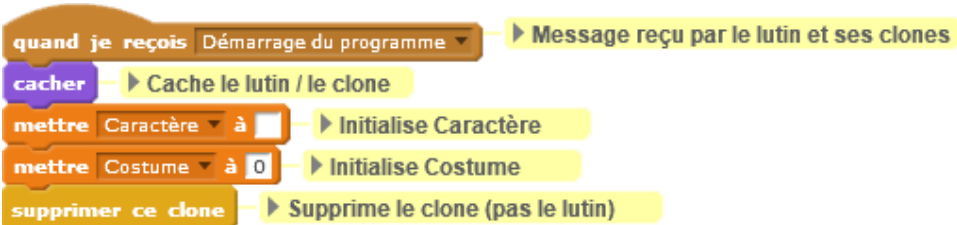
On aura besoin de deux propriétés : **Caractère** et **Costume**.

- **Caractère** Variable chaîne de caractère  
Contient le caractère (lettre) que le clone représente.
- **Costume** Variable numérique

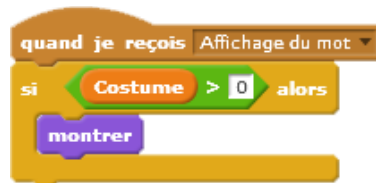
Contient le numéro de costume du clone cachant la lettre à deviner (au début de la partie, le costume est celui représentant le « \_ », puis change lorsque la lettre est devinée). Ainsi, par exemple, si **Mot à deviner** est « GLAIVE », et que l'on considère le clone représentant le « I » :

- **Caractère** vaut « I »,
- **Costume** vaut 9.

Le script de **Mot** se décompose en 4 parties :



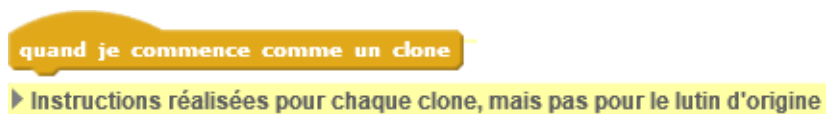
La suppression du clone permet de s'assurer, en début de partie, qu'aucun clone d'une partie précédente n'est plus présent.



Cette partie doit s'exécuter au moment où **Abby** envoie un message juste avant que le jeu ne débute. **Costume** vaut 0 uniquement pour le lutin initial : il ne doit pas se montrer.



Cette partie s'exécute après que le joueur a saisi une lettre.



Dans cette dernière partie, appelée par **définir Initialisation**, on attribue à chaque clone lors de sa création ses propriétés **Caractère** et **Costume**.

Chaque clone reste caché, mais se met à sa place définitive, prêt à s'afficher : pour se placer, on crée une variable **Position horizontale** initialisée par Abby à  $\text{longueur de Mot à deviner} * -15 + 15$  (formule de centrage du mot horizontalement dans la scène), puis augmentée de 30 après chaque création de clone. Il suffit de faire exécuter **aller à x: Position horizontale y: 150** par le clone pour qu'il se place convenablement.

Le script de **Erreurs** est plus court : seule la propriété **Costume** devra être créée et il n'y aura besoin que de 2 parties :

**quand je reçois** Démarrage du programme ▾

Cette partie est identique à celle de **Mot**.

Pour modifier la couleur du texte sans retoucher chaque image, on pourra utiliser

**mettre l'effet** couleur ▾ à 175

**quand je commence comme un clone**

La création du clone est appelée à chaque fois qu'une lettre perdante est proposée par le joueur. Dans cette procédure, le clone se place (une variable **Position verticale** pourra être utile), puis on calcule le numéro du costume correspondant à **réponse** avant d'afficher le clone.