
# GameCode : exemples de code

En JavaScript

**GameCode**

Créer et programmer son
**jeu vidéo**

Bibliothèques Sans Frontières
Libraries Without Borders

V1.0
Auteur : Tralalere

# Sommaire

# Idées d'actions pour le joueur

## Gérer les points de vie du joueur

### Augmenter le nombre de point de vie du joueur au contact d'un bonus

Comme dans tous les jeux vidéo, la valeur de la vie est stockée dans une variable. La valeur de cette variable peut varier au cours de la partie.

Par exemple, la valeur de la vie du joueur peut changer lorsqu'il touche une pièce.

**Résultat attendu :**

| Page de script du joueur | Page de script de la pièce |
|---|---|
|  |  |
| `joueur.vie = 8;` | `quand ('joueur arrive', fonction()`<br>`{`<br>`  joueur.vie = joueur.vie + 1;`<br>`  objet.disparait();`<br>`})` |

**Explications :**

Chaque élément de GameCode a une liste de variables associées, appelées « propriétés ». Cette liste peut varier d'un élément à un autre. Par exemple, les objets n'ont pas de propriété « vie » alors que le joueur en a une..

Dans GameCode, le joueur a 6 vies par défaut, mais il est possible de redéfinir ce nombre à l'aide du code suivant :

```
joueur.vie = 8;
```

Lorsque le programme passe sur cette ligne, il redéfinit le nombre de vies du joueur à la valeur 8.

Cette ligne peut être placée dans un événement appartenant à un bonus :

```
quand ('joueur arrive', fonction()
{
    joueur.vie = 8;
})
```

Quand le joueur entre en contact avec le bonus, son nombre de vies sera redéfinit à la valeur 8.

Pour récupérer la valeur stockée dans cette variable, il est possible d'utiliser directement le code suivant : joueur.vie.

On peut donc effectuer le test suivant :

```
si (joueur.vie == 8)
{
    joueur.dit('J'ai 8 vies');
}
```

Il possible d'avoir envie de faire évoluer la valeur d'une variable sans avoir à vérifier ce qu'il y a dedans.
Par exemple, si on gagne une vie, on doit ajouter la valeur 1 à la valeur contenue dans la variable de vie. Cette valeur peut évoluer constamment au cours du programme. On a donc besoin de pouvoir l'utiliser sans avoir à la vérifier :

```
joueur.vie = joueur.vie + 1;
```

Ici, on ajoute une unité à la valeur contenue dans la variable de vie du joueur.

Enfin, on peut également placer cette ligne dans un événement :

```
quand ('joueur arrive', fonction()
{
    joueur.vie = joueur.vie + 1;
    objet.disparait();
})
```

Quand le joueur entre en contact avec la pièce, il gagne une vie, et l'objet disparaît.

### Faire perdre une vie au joueur lorsqu'il est touché par un personnage

Nous avons vu, plus haut, comment le joueur peut gagner des vies en récupérant des bonus, on peut aussi imaginer qu'il perde des vies lorsqu'il essuie des tirs ennemis.

**Résultat attendu :**

Page de script du joueur


```
/*Déclaration de la variable « vie »*/
joueur.vie = 4;

/*Le joueur perd une vie quand il est touché par le tir d'un personnage*/
quand ('est touché', fonction()
{
    joueur.vie =  joueur.vie - 1;
})
```

**Explications :**

Ce script permet à n'importe quel personnage du niveau de faire perdre une vie au joueur quand son tir le touche.
Au début du jeu, nous avons défini le nombre de vies que possède le joueur. Ici, le joueur a 4 vies. Dès que le joueur va être touché par le tir d'un personnage, une fonction va être appelée pour faire perdre une vie au joueur.

**Remarque :**

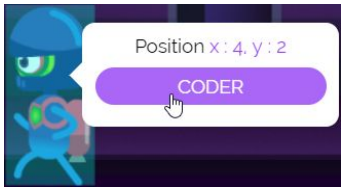
Pour que ce script se déclenche, il est nécessaire de coder en parallèle le tir des personnages ennemis. Un exemple est donné plus bas dans ce document.

## Faire évoluer le score du joueur

La notion de score est une composante essentielle du jeu. Le score du joueur peut varier en fonction des objectifs atteints. Par exemple, le joueur peut gagner des vies lorsqu'il arrive à faire disparaître des personnages.

**Résultat attendu :**

| Page de script du joueur | Page de script du personnage |
|---|---|





```
joueur.score = 0;
```

```
personnage.vie = 3;

quand ('est touché', fonction()
{
    personnage.vie = personnage.vie - 1;
    si (personnage.vie <= 0)
    {
        joueur.score = joueur.score + 10;
    }
})
```

**Explications :**

En plus des propriétés par défaut des éléments du jeu, comme la « vie » du joueur, il est possible d'ajouter de nouvelles propriétés aux éléments. C'est particulièrement utile avec le joueur, qui est un élément accessible depuis chacune des pages de script des éléments de la scène. On peut donc l'utiliser pour stocker un score ou la valeur d'un interrupteur (vrai ou faux).

Pour ajouter une nouvelle propriété au joueur, comme le score, on écrit :

```
joueur.score = 0;
```

Si le jeu rapporte des points au joueur lorsqu'il tue un personnage, on peut avoir le code suivant dans un personnage :

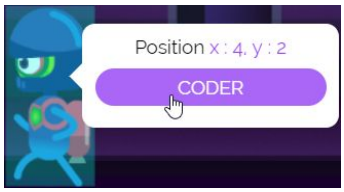
```
quand ('est touché', fonction()
{
    personnage.vie = personnage.vie - 1;
    si (personnage.vie <= 0)
    {
        joueur.score = joueur.score + 10;
    }
})
```

## Donner un boost au joueur lorsqu'il s'approche d'un personnage

Afin d'aider le joueur, à l'approche d'un personnage, ses capacités peuvent être renforcées (vitesse + hauteur de saut) .

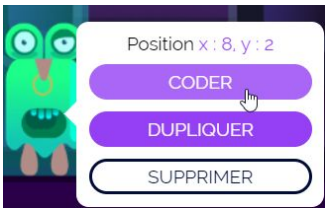
**Résultat attendu :**

Page de script du joueur




```
/*Initialisation de la variable « hauteurSaut » et de la variable « vitesse »*/
joueur.hauteurSaut = 3;
joueur.vitesse = 2;
```

Page de script du personnage




```
/*Déclaration et activation de la variable « sautActif » et déclaration des deux variables « hauteurSautOrig » et « vitesseOrig »*/
var sautActif = vrai
var hauteurSautOrig = joueur.hauteurSaut
var vitesseOrig = joueur.vitesse

/*Boucle « toujours »*/
toujours(fonction()
{
/*Condition : si le joueur et le personnage sont proches et que la variable « sautActif » est égale à vrai (true) alors le joueur fait un boost*/
  si ((distanceEntre(joueur, personnage) <= 3) && (sautActif == vrai))
  {
    joueur.hauteurSaut += 1;
    joueur.vitesse += 1;
    joueur.saute();
  }

/*Condition : activation des deux variables « hauteurSautOrig » et « vitesseOrig », si le personnage est au sol*/
  si (personnage.estAuSol())
  {
    joueur.hauteurSaut = hauteurSautOrig;
    joueur.vitesse = vitesseOrig;
```

```
  }

/*Condition : activation de la variable « sautActif » si le joueur et le personnage sont proches*/
  si (distanceEntre(joueur, personnage) > 2)
  {
    sautActif = vrai;
  }
})
```

**Explications :**

Au début du jeu, la hauteur de saut du joueur et sa vitesse sont définies. La hauteur de saut est de 3 ; ce qui veut dire que le joueur sautera à une hauteur équivalente à 3 cases. La vitesse est de 2 ; ce qui veut dire que le joueur se déplacera à une vitesse équivalente à 2 cases.
Il faut créer une variable pour que le joueur puisse recevoir un boost. Pour activer cette variable, appelée « sautActif », il faut lui donner la valeur « vrai » (true).
Il faut également créer deux autres variables pour sauvegarder la hauteur de saut et la vitesse du joueur définies par défaut.

Trois conditions sont définies dans le code du personnage :

- La première établit que, si la distance entre le joueur et le personnage est inférieure ou égale à 3 cases et que la variable « sautActif » est égale à vrai (true), alors le joueur aura une hauteur de saut et une vitesse supplémentaire. Sa hauteur de saut va donc être égale à 4 cases et sa vitesse à 3 cases. Il faut également ajouter la fonction « saute » pour que le joueur puisse sauter.
- La deuxième établit que, si le personnage est au sol, alors les deux variables concernant la hauteur de saut et la vitesse du joueur s'activent.
- La troisième établit que, si la distance entre le joueur et le personnage est supérieure à 2 cases, alors le saut s'active (true) et le joueur fait un boost en étant propulsé en avant par un saut avec une vitesse de 3 cases et une hauteur de saut de 4 cases.

Avec la boucle « toujours », les trois conditions seront activées en continu. Dès que le joueur passera sur ce personnage, il aura un boost.

# Idées d'actions pour un personnage

## Déplacer un personnage au sol

### Faire avancer le personnage

Le script suivant permet de faire en sorte que le personnage avance en continue s'il est au sol.

**Résultat attendu :**

Page de script du personnage 


```
/*Boucle « toujours »*/
toujours (fonction()
{
    /*Condition : si le personnage est au sol alors il avance*/
    si (personnage.estAuSol())
    {
        personnage.avance();
    }

})
```

**Explications :**

Il faut créer une condition qui précise que si le personnage est au sol, alors il avance. Il faut ensuite placer cette condition dans une boucle « toujours » pour que l'événement se répète indéfiniment.

### Faire aller et venir le personnage

Il est possible de faire en sorte que le personnage effectue une ronde régulière en faisant des allées et venues.

**Résultat attendu :**

Page de script du personnage 


```
toujours(fonction()
{
    personnage.avance();
});
```

```
repeterChaque('2 secondes', fonction () {
    personnage.retourneToi();
})
```

**Explications :**

L'effet de ronde est donné par le fait que le personnage avance tout le temps et se retourne toutes les deux secondes.

Pour faire varier la distance parcourue par le personnage durant sa ronde, on fait varier la notion de temps dans la boucle « répéter chaque ».

Ce déplacement peut également être paramétré selon la distance par rapport au point d'origine du personnage et non au temps.

### Définir un parcours précis pour un personnage

Le script suivant permet de définir un parcours précis pour le personnage.

**Résultat attendu :**

Page de script du personnage 


```
/* Déclaration de la variable « positionOrigX »*/
var positionOrigX = personnage.x;

/*Boucle « toujours »*/
toujours(fonction()
{
    /*Condition : le personnage se retourne en fonction de deux positions définies*/
    si ((personnage.x - positionOrigX >= 2) || (personnage.x - positionOrigX <= -2))
    {
        personnage.retourneToi();
    }
    /*Déplacement en continu du personnage*/
    personnage.avance();
})
```

**Explications :**

Il faut déclarer une variable qui va prendre en compte la position initiale du personnage et l'associer à une valeur x, qui est la position du personnage sur l'axe des abscisses.

```
var positionOrigX = personnage.x;
```

Puis, il faut déclarer une condition qui va faire en sorte que si la distance entre la position initiale du personnage et sa position sur l'axe des abscisses (x) est inférieure ou égale à 2 cases ou à -2 cases, alors le personnage se retourne. Le terme « ou » est représenté par le symbole « || ».

```
si ((personnage.x - positionOrigX >= 2) || (personnage.x - positionOrigX <= -2))
{
    personnage.retourneToi();
}
```

Pour que le personnage puisse avancer en continu à l'aide de la méthode « avance() » et afin que la condition de retournement se répète indéfiniment, il ne faut pas oublier d'inclure ce code dans une boucle « toujours ».

```
toujours(fonction()
{
    /*Condition : le personnage se retourne en fonction de deux positions définies*/
    si ((personnage.x - positionOrigX >= 2) || (personnage.x - positionOrigX <= -2))
    {
        personnage.retourneToi();
    }
    /*Déplacement en continu du personnage*/
    personnage.avance();
})
```

Le personnage va donc avancer de 2 cases à droite de sa position initiale, puis se retourner et réavancer à gauche de 4 cases (2 cases pour aller à sa position initiale, puis 2 cases après sa position initiale) et se retourner pour recommencer en continu le parcours défini.

### Faire sauter le personnage au contact d'un obstacle

Une fois que le personnage se déplace, il sera bloqué par les différents obstacles sur sa trajectoire. Il est possible de le faire sauter par-dessus les obstacles.

**Résultat attendu :**

Page de script du personnage 


```
quand ('est bloqué', fonction()
{
    personnage.saute();
})
```

**Explications :**

La méthode « saute » du personne est appelée dès qu'il rencontre un obstacle.

**Remarque :**

On peut, de la même façon; faire se retourner le personnage lorsqu'il rentre en contact avec l'obstacle.

```
quand ('est bloqué', fonction()
{
    personnage.retourneToi();
})
```

## Faire sauter un personnage au-dessus du vide

Les personnages sont capables de détecter des bonus.

On peut très bien placer le bonus  sur la scène et quand le personnage entrera en contact avec, il sautera. Il suffit de placer ce bonus à l'endroit où l'on veut que le personnage saute.

Exemple de mise en scène :

Il est possible aussi de rendre invisible l'objet dès le début de la partie pour ne pas laisser d'indices au joueur sur la trajectoire du personnage.

**Résultat attendu :**

Page de script du bonus 


```
objet.estVisible = faux;
quand ('personnage arrive', fonction(personnage)
{
    personnage.saute();
})
```

**Explications :**

L'événement « quand 'personnage arrive' » récupère l'entité du personnage détecté. Ainsi, même s'il peut y avoir plusieurs personnages sur la scène, le code identifie le personnage qui vient de rentrer en contact avec le bonus afin de le faire sauter lui et pas un autre.

**Remarque :**

La hauteur du saut du personnage dépendra de la valeur de la variable « personnage.hauteurSaut » propre à chaque personnage de la scène.

## Téléporter un personnage

Plutôt que de marcher, le personnage peut également disparaître et réapparaître aussitôt à un tout autre emplacement de la scène. Cela lui permet, par exemple, de se dérober lorsque le joueur approche.

**Résultat attendu :**

Page de script du personnage 


```
/*Boucle « toujours »*/
```

```
toujours (fonction()
{
/*Condition : si le personnage et le joueur sont proches alors le personnage se téléporte*/
    si (distanceEntre(joueur, personnage) <= 1)
    {
        personnage.teleporte({ x: 9, y: 4});
    }
})
```

**Explications :**

Ce script permet au personnage de se téléporter à un endroit précis du jeu.
Pour cela, il faut créer une boucle « toujours » pour que l'événement se répète indéfiniment. Il faut également définir une condition permettant au personnage de se téléporter si la distance entre le personnage et le joueur est inférieure ou égale à 1 case.

Dans cet exemple, le personnage se téléporte à 9 sur l'axe des abscisses et à 4 sur l'axe des ordonnées.

## Faire voler un personnage

### Faire léviter un personnage

Pour faire léviter un personnage, il faut paramétrer la propriété « vole ». Et pour le faire voler, il est nécessaire de le faire se déplacer.

**Résultat attendu :**

Page de script du personnage 


```
personnage.vole = vrai;

var newY = 2;

toujours(fonction()
{
    si (personnage.y <= 2)
    {
        newY = 6;
    }
```

```
    sinon si (personnage.y >= 6)
    {
        newY = 2;
    }
    personnage.vaVers(personnage.x, newY);
})
```

**Explication :**

Lorsque la valeur de la propriété « vole » est égale à « vrai », la gravité n'est plus appliquée au personnage. La fonction « vavers » permet alors de déplacer verticalement le personnage (axe Y), paramètre qui n'est pas pris en compte lorsque la gravité s'applique.

## Faire voler le personnage d'un point A à un point B

De la même manière que nous avons vu, précédemment, comment faire aller et venir horizontalement un personnage, nous allons voir comment faire aller et venir un personnage dans les air, entre deux positions.

**Résultat attendu :**

Page de script du personnage 


```
/*Activation de la variable « vole »*/
personnage.vole = vrai;

/*Déclaration de la variable « positions » dans un tableau*/
var positions = [
        {x: 23, y: 2},
        {x: 23, y: 6}
];

/*Déclaration de la variable « superman »*/
var superman = 0;

/*Boucle « répéter chaque X secondes » et calcul de la variable « superman »*/
repeterChaque('2 secondes', fonction()
{
    superman = (superman + 1) % positions.length;
```

```
    personnage.vaVers(positions[superman]);
})
```

**Explication :**

Ce script permet de faire voler le personnage dans les airs verticalement entre deux points définis.

Pour cela, il faut activer la variable « vole » en lui donnant la valeur « vrai ». Si elle est activée, la gravité ne s'applique plus sur le personnage..

```
personnage.vole = vrai;
```

Il faut ensuite créer un tableau dans lequel seront définies les positions (abscisses et ordonnées) de la trajectoire de vol du personnage. Il va donc voler d'un point A à un point B.
Dans l'exemple suivant, le point A est placé à x : 23 et y : 2, et le point B est placé à x : 23 et y : 6.

```
var positions = [
        {x: 23, y: 2},
        {x: 23, y: 6}
];
```

Une deuxième variable, la variable « superman », doit être déclarée en lui donnant la valeur 0. Elle permettra de déterminer si le personnage est sur l'allé ou le retour.

```
var superman = 0;
```

Il faut ensuite faire un calcul pour activer les positions de vol du personnage.
Le signe « % » est un modulo qui permet de calculer le reste d'une division. Le résultat est toujours un nombre entier.
La propriété « positions.length » permet de récupérer le nombre d'éléments que contient le tableau. Dans notre exemple « positions.length » vaut 2 car il y a deux éléments dans le tableau.

```
superman = (superman + 1) % positions.length;
```

Lors du premier calcul de la variable superman on a donc : (0+1) % 2, ce qui est égal à 1. La variable superman passe donc à 1.
Lors du calcul suivant, on aura : (1+1) % 2, ce qui est égal à 0. La variable superman repassera donc à 0.
Et ainsi de suite.

Pour que l'événement se répète toutes les deux secondes, il faut créer une boucle « répéter chaque seconde ».
Le personnage va donc faire des allers-retours en continu entre la position A et B.

## Suivre le joueur

### Faire suivre le joueur par un personnage

La position du joueur, à un instant donné, est inconnue car il peut être en mouvement. Il est cependant possible de programmer un personnage pour qu'il le suive.

**Résultat attendu :**

Page de script du personnage 


```
toujours(fonction()
{
    personnage.vaVers(joueur.x, joueur.y);
});
```

**Explication :**

Pour qu'un personnage poursuive le joueur durant la partie, il faut utiliser 3 notions :

- l'action « vaVers() » pour déplacer le personnage vers un point précis ;
- les coordonnées du joueur ;
- une boucle « toujours » pour répéter une action tout au long de la partie.

**Remarque :**

Avec ce script, le personnage sera bloqué dès le premier obstacle.

### Faire suivre le joueur par un personnage qui lui tire dessus lorsque qu'il s'approche

En plus de suivre le joueur, le personnage peut également défendre son territoire en lui tirant dessus.

**Résultat attendu :**

Page de script du personnage 


```
/*Boucle « toujours »*/
toujours(fonction()
{
/*Condition : si le joueur et le personnage sont proches alors le personnage va vers le joueur et lui tire dessus*/
    si (distanceEntre(joueur, personnage) <= 3)
    {
        personnage.vaVers(joueur);
        personnage.tire();
    }
})
```

**Explication :**

Ce script fait aller le personnage vers le joueur pour lui tirer dessus lorsque la distance qui les sépare est inférieure ou égale à 3 cases.
Pour que l'événement se répète indéfiniment, il faut mettre en place une boucle « toujours » .
Si le joueur est plus loin que 3 cases, alors le personnage ne peut pas aller vers lui ni lui tirer dessus.

### Faire tirer le personnage en continu sur le joueur en le suivant et le faire arrêter lorsque le personnage disparaît

On fait disparaître le personnage poursuiveur lorsque le joueur lui tire dessus.

**Résultat attendu :**

Page de script du personnage 


```
/*Déclaration et activation de la variable « disparaît »*/
var disparaît = vrai;
```

```
/*Boucle « répéter chaque x secondes »*/
repeterChaque('1 secondes', fonction ()
{
    personnage.vaVers(joueur);

    /*Boucle « toujours »*/
    toujours (fonction()
    {
        personnage.tire();
    })
/*Condition : le joueur perd une vie si le joueur et le personnage sont proches et (&&) si la variable « disparaît » est égale à « vrai » (true)*/
    si ((distanceEntre(joueur, personnage) < 1) && (disparaît === vrai))
    {
        joueur.vie = joueur.vie - 1;
    }
})

/*Quand le personnage est touché, il disparaît et la variable « disparaît » devient égale à « faux » (false)*/
quand('est touché', fonction ()
{
    personnage.disparait();
    disparaît = faux;
})
```

**Explication :**

Ce script permet au personnage de suivre sans cesse le joueur pour lui tirer dessus. Le joueur peut se défendre en tirant lui-même sur le personnage. Pour cela, il faut créer une variable qui fasse en sorte que le personnage disparaisse, ainsi que ses tirs, quand il est touché par les tirs du joueur.
Si la variable n'est pas mise en place, le personnage disparaît mais ses tirs restent actifs et font perdre inutilement des vies au joueur. Pour activer la variable « disparaît », il faut que sa valeur soit égale à « vrai » (true).
Pour cela, deux boucles sont nécessaires :

- une boucle « répéter chaque seconde » pour que le personnage aille vers le joueur toutes les unes secondes. Si le joueur s'éloigne, le personnage va vers lui ;
- une boucle « toujours » pour que le personnage tire indéfiniment sur le joueur.

Il faut ensuite une condition avec laquelle le joueur perd une vie si la distance entre le personnage et le joueur est inférieure à 1 case et si la variable « disparaît » est égale à « vrai » (true). Cette condition va alors appeler une fonction qui va faire

perdre une vie au joueur. Si le joueur est plus loin (supérieur à 1 case) alors le personnage ne peut pas aller vers lui ni lui tirer dessus.
Pour que les tirs du personnage ne soient plus actifs et ne touchent plus le joueur, il faut que la valeur de la variable « disparaît » soit mise à « faux » (false).

# Idées d'actions pour le joueur et un personnage

## Déclencher un dialogue

L'idée est de créer un dialogue entre le personnage et le joueur en le débutant quelques secondes après le début du jeu.

**Résultat attendu :**

Page de script du personnage 


```
/* Boucle « retarder x secondes »  */
retarder('3 secondes', fonction () {
    /*Dialogue entre deux personnages*/
    personnage.dit('Comment tu t\'appelles ?');
    retarder('5 secondes', fonction ()
    {
        personnage.dit('Lou');
    })
})
```

**Explications :**

Il faut créer une boucle « retarder trois secondes » pour débuter le dialogue trois secondes après le début du jeu.
Il faut mettre aussi un temps de quelques secondes entre les deux répliques pour éviter que le personnage et le joueur parlent en même temps.

# Idées d'actions pour un objet

## Faire disparaître un objet

Dans l'exemple suivant, on cherche à faire disparaître une pièce afin de faire croire que le joueur a pris l'objet avec lui et l'a mis dans son inventaire.

**Résultat attendu :**

Page de script de la pièce 


```
quand ('joueur arrive', fonction()
{
    objet.disparait();
})
```

**Explications :**

Pour faire disparaître un objet, on a besoin de deux choses :

- déterminer quand le joueur touche l'objet à l'aide d'un événement ;
- appliquer l'action « disparaît » à l'objet.

**Remarque :**
Il est également possible de faire disparaître temporairement un objet à l'aide de la propriété « objet.visible ».

## Faire clignoter un objet

On peut jouer avec la propriété « visible » d'un objet afin de le faire clignoter.

**Résultat attendu :**

```
/* Boucle « répéter chaque x secondes » */
repeterChaque('1 secondes', fonction ()
{
/* Condition : si la caractéristique de l'objet est égale à vrai alors l'objet devient invisible, sinon il devient visible */
    si (objet.visible == vrai)
    {
```

```
        objet.visible = faux;
    }
    sinon
    {
        objet.visible = vrai;
    }
})
```

**Explications :**

Ce script permet de faire clignoter un objet toutes les secondes.
Il fait appel à la boucle « répéter chaque seconde » pour que le script soit activé toutes les unes secondes.

```
repeterChaque('1 secondes', fonction ()
{
}
```

On ajoute ensuite une condition : si l'objet est égal à « vrai » alors l'objet devient invisible (false), sinon l'objet devient visible (true).

```
si (objet.visible == vrai)
    {
        objet.visible = faux;
    }
    sinon
    {
        objet.visible = vrai;
    }
```

L'objet va donc devenir visible puis invisible continuellement, ce qui donnera un effet de clignotement toutes les secondes.

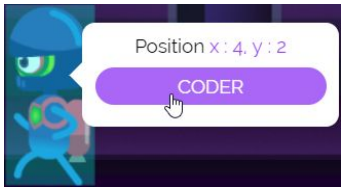
## Programmer un interrupteur

Pour programmer un interrupteur, on a besoin :

- d'un objet servant d'interrupteur ;
- d'un objet à activer avec l'interrupteur ;
- du joueur pour stocker la valeur de l'interrupteur.

**Résultat attendu :**

| Page de script du joueur | Page de script du bonus « interrupteur » | Page de script des bonus modifiés par l'interrupteur |
|---|---|---|

| | | |
|---|---|---|
|  |  |  |
| joueur.interrupteur = faux; | quand (“joueur arrive”, fonction()<br>{<br>joueur.interrupteur = vrai;<br>}) | objet.traversable = faux;<br><br>toujours (fonction()<br>{<br>si (joueur.interrupteur == vrai)<br>{<br>objet.disparaît();<br>}<br>}); |

**Explications :**

Il faut commencer par initialiser la valeur de l’interrupteur dans le code du joueur.

```
joueur.interrupteur = faux;
```

On programme ensuite l’objet qui fonctionnera comme interrupteur :

```
//code de l’objet
quand (“joueur arrive”, fonction()
{
    joueur.interrupteur = vrai;
})
```

Dans l’exemple, une porte est fabriquée avec des objets rendus non traversables. Il faut que ces objets vérifient régulièrement la valeur de l’interrupteur pour savoir s’ils peuvent s’activer ou non.

```
//code des bonus “porte”
objet.traversable = faux;

toujours (fonction()
{
    si (joueur.interrupteur == vrai)
    {
        objet.disparaît();
    }
```

```
});
```

Ces objets, bloquants pour le joueur, disparaissent lorsque celui-ci actionne l'interrupteur.