Créer un jeu avec P5play

Table des matières

1. Création de l'environnement

p5play est une bibliothèque JavaScript amenant des fonctions supplémentaires à la bibliothèque p5js (portage de Processing en JavaScript) permettant de programmer des jeux. Elle est relativement simple à prendre en main et permet de gérer les animations de sprites, les collisions…

Les avantages de p5 sont nombreux. Vous avez à votre disposition des fonctions toutes prêtes pour vous permettre de prendre en charge les événements, y compris les événements tactiles, et de mouvement d'accéléromètre des téléphones.

p5.play a été développée par Paolo Pedercini. Vous trouverez le site officiel du projet ici : http://p5play.molleindustria.org/

Pour apprendre à utiliser p5, n'hésitez pas à lire la documentation, ce qui est suffisant pour se faire une base, ou bien lire leurs tutoriels.

1.1. À faire vous même

Dans un dossier "p5_play", que vous aurez créé au préalable dans votre dossier personnel, créez un dossier "play_0".

Dans le dossier "play_0", créez les dossiers et fichier suivants :

- un fichier "index.html"
- un fichier "script.js"
- un fichier "style.css"
- un dossier "asset"
- un dossier "lib"

Vous devriez donc avoir ceci dans votre dossier :

1.2. À faire vous même

Vous allez maintenant avoir à télécharger 2 bibliothèques : p5.js et p5.play

1. Pour télécharger p5.js, rendez-vous sur le site http://p5js.org/download/
2. Cliquez dans le rectangle "p5.min.js", cela devrait lancer le téléchargement du fichier.
3. Une fois le téléchargement terminé, placez le fichier "p5.min.js" dans le dossier "lib" que vous venez de créer.
4. Ensuite il vous faut télécharger la bibliothèque "p5.play" en allant sur le site http://p5play.molleindustria.org/
5. Une fois sur le site, cliquez sur "Download", cela devrait lancer le téléchargement d'une archive au format zip ("p5.play-master.zip").
6. Dès que le téléchargement est terminé, "dézippez" cette archive. Vous devriez alors vous

retrouver avec un dossier "p5.play-master".

Dans ce dossier "p5.play-master", vous devriez trouver un dossier "lib". Dans ce dossier "lib", vous devriez trouver un fichier "p5.play.js".

Copiez ce fichier "p5.play.js" dans le dossier "lib" que vous avez créé ci-dessus.

Vous devriez maintenant avoir dans votre dossier "lib" 2 fichiers : "p5.min.js" et "p5.play.js"

1.3. À faire vous même

Complétons le fichier index.html à l'aide d'un éditeur de texte (Bloc-notes ou Notepad++ par exemple) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
   <meta charset="utf-8">
   <title>Créer un jeu avec p5.play</title>
   <link rel="stylesheet" href="style.css">
   <script src="lib/p5.min.js"></script>
   <script src="lib/p5.play.js"></script>
   <script src="script.js"></script>
</head>
<body>
</body>
</html>
```

La structure que nous venons de mettre en place (dossier "play_0") pourra être "copier-coller" à chaque fois que vous commencerez à coder un nouvel exemple (il suffira de renommer le dossier en "play_1", "play_2",....). Dans la suite, seul le fichier "script.js" sera modifié. Le fichier "style.css" n'est pour l'instant pas utilisé, il le sera, si nécessaire, dans les prochaines activités.

2. Les sprites

Nous allons commencer par apprendre à afficher un sprite à l'écran. Mais qu'est-ce qu'un sprite ?

Sprite est un mot anglais possédant plusieurs significations. Il est notamment employé dans les domaines de l'infographie et du jeu vidéo, où sprite désigne une image en deux dimensions qui peut être déplacée par rapport au fond de l'écran.

Comme indiqué ci-dessus, un sprite est avant tout une image. Dans un premier temps il est donc nécessaire de télécharger une image.

2.1. À faire vous même

Téléchargez l'image en cliquant ici (clic droit puis "Enregistrer sous"). Placez ensuite cette image dans le dossier "asset".

2.2. À faire vous même

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite;
function preload()
{
	img=loadImage("asset/smiley.png")
}

function setup()
{
	createCanvas(800,300);
	smiley_sprite=createSprite(400,150);
	smiley_sprite.addImage(img);
}

function draw()
{
	background(240,240,240);
	drawSprites();
}
```

Analysons le code ci-dessus :

p5 fonctionne avec deux fonctions principales : setup() et draw(). La fonction setup() s'exécutera en premier et une seule fois au démarrage du programme. Elle vous servira principalement à créer le Canvas. Quant à la fonction draw(), elle s'exécutera en boucle. La fonction preload sera utilisée au besoin pour charger des images.

- La méthode **loadImage** nous permet de charger une image. Cette image est "placée" dans la variable img.
- La méthode **createSprite** permet de créer un sprite. Cette méthode prend en paramètres les coordonnées x et y du centre du sprite (dans notre exemple, nous avons placé le sprite au centre de notre canvas (fenêtre de jeu)). Le sprite est placé dans la variable smiley_sprite.
- La méthode **addImage** permet d'associer un sprite à une image. Cette méthode prend en paramètre une image.
- La fonction **drawSprites** permet d'afficher tous les sprites présents. Notez que le background(240,240,240) placé dans la fonction draw permet d'effacer la fenêtre à chaque image.

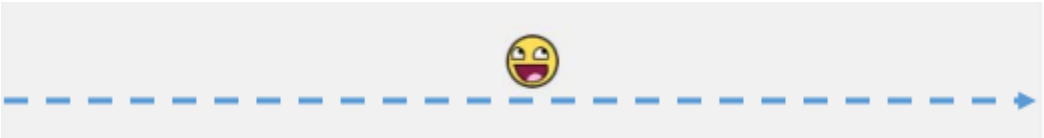
2.3. À faire vous même

En modifiant le code vu dans le "À faire vous même 2.2", placez le sprite aux coordonnées (200,75).

2.4. À faire vous même

Sachant que smiley_sprite.position.x correspond à la coordonnée x du sprite et que

smiley_sprite.position.y correspond à la coordonnée y du sprite (à condition que votre sprite soit "rangé" dans la variable smiley_sprite comme dans l'exemple ci-dessus), écrivez un programme permettant au sprite de traverser l'écran de gauche à droite.



2.5. À faire vous même

p5.play propose la méthode "setVelocity" qui permet de donner à notre sprite une vitesse. Cette méthode prend 2 paramètres : la vitesse selon x et la vitesse selon y.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite;
function preload()
{
	img=loadImage("asset/smiley.png")
}

function setup()
{
	createCanvas(800,300);
	smiley_sprite=createSprite(0,150);
	smiley_sprite.addImage(img);
	smiley_sprite.setVelocity(2,0);
}

function draw()
{
	background(240,240,240);
	drawSprites();
}
```

le smiley_sprite.setVelocity(2,0) donne une vitesse de 2 pixels par image à notre sprite.

Il est possible d'utiliser smiley_sprite.velocity.x et smiley_sprite.velocity.y à la place de la méthode setVelocity.

Par exemple :

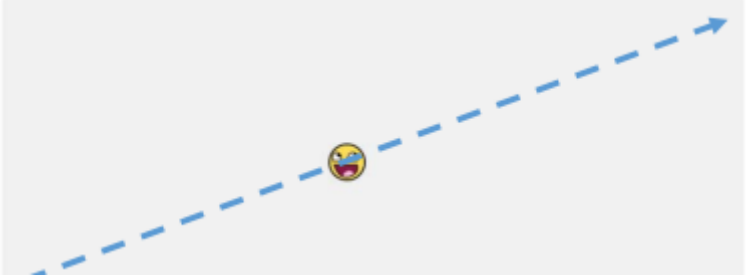
```
	smiley_sprite.setVelocity(2,4);
```

peut être remplacée par :

```
	smiley_sprite.velocity.x=2;
	smiley_sprite.velocity.y=4;
```

2.6. À faire vous même

Écrivez un programme permettant au sprite de traverser l'écran en diagonale.



2.7. À faire vous même

p5 propose 2 fonctions qui devront être complétées par le programmeur :

- le code se trouvant dans la fonction "keyPressed" est exécuté une fois quand l'utilisateur enfonce une touche
- le code se trouvant dans la fonction "keyReleased" est exécuté une fois quand une touche du clavier est relâchée

Ces 2 fonctions ne prennent aucun paramètre et ne retournent aucune valeur.

Saisissez, analysez et testez ce code

```
function setup()
{
	createCanvas(200,200);
	fill(0);
}

function draw()
{
	background(240);
	ellipse(100,100,50,50);
}

function keyPressed()
{
	fill(255);
}

function keyReleased()
{
	fill(0);
}
```

Attention : pour pouvoir utiliser les 2 fonctions que nous venons de voir, il faut que la fonction "draw" soit présente dans le programme (même si elle est vide)

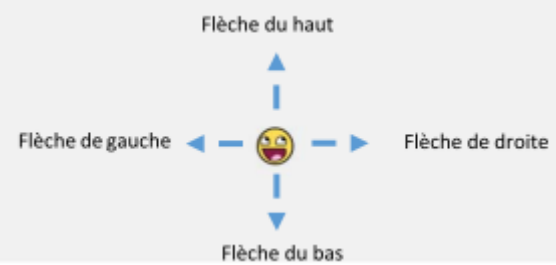
Il est possible de détecter la touche qui a été frappée grâce à la variable "keyCode". Cette variable "keyCode" est égale aux codes des touches JavaScript que vous trouverez ici.

Par exemple pour savoir si c'est la touche "a" qui a été frappée, il suffira d'écrire :

```
function keyPressed()
{
```

```
	if ( keyCode==65 )
	{
		//mettre ici le code a exécuter quand la touche "a" est enfoncée
	}
}
```

En vous aidant de ce qui a déjà été vu, créez un programme permettant de faire bouger le sprite à l'aide des flèches du clavier.



3. Gestion des collisions et des « rebonds »

P5.play propose une gestion des collisions et des "rebonds" simple et efficace.

3.1. À faire vous même

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite_1;
var smiley_sprite_2;
function preload()
{
	img=loadImage("asset/smiley.png");
}

function setup()
{
	createCanvas(800,300);
	smiley_sprite_1=createSprite(100,150);
	smiley_sprite_1.addImage(img);
	smiley_sprite_1.setVelocity(3,0);
	smiley_sprite_2=createSprite(600,150);
	smiley_sprite_2.addImage(img);
}

function draw()
{
	background(240,240,240);
	smiley_sprite_1.bounce(smiley_sprite_2);
	drawSprites();
}
```

Analysons le code ci-dessus :

Nous avons deux sprites :

- smiley_sprite_2 est, à l'origine, immobile.
- smiley_sprite_1 "fonce" vers smiley_sprite_2

La méthode **bounce** permet de gérer les collisions (et les rebonds éventuels) entre les sprites.

Dans la fonction **draw** nous avons ajouté la ligne smiley_sprite_1.bounce(smiley_sprite_2) qui permet de gérer la collision entre smiley_sprite_1 et smiley_sprite_2

3.2. À faire vous même

Si vous lancez une balle de ping-pong sur une boule de pétanque, au moment de la collision la boule de pétanque ne bougera (quasiment) pas alors que la balle de ping-pong repartira dans l'autre sens.

p5.play permet de gérer ce genre de situation en attribuant des "masses" aux sprites.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite_1;
var smiley_sprite_2;
function preload()
{
	img=loadImage("asset/smiley.png");
}

function setup()
{
  createCanvas(800,300);
  smiley_sprite_1=createSprite(100,150);
  smiley_sprite_1.addImage(img);
  smiley_sprite_1.setVelocity(3,0);
  smiley_sprite_1.mass=1;
  smiley_sprite_2=createSprite(600,150);
  smiley_sprite_2.addImage(img);
  smiley_sprite_2.mass=15;
}

function draw()
{
	background(240,240,240);
	smiley_sprite_1.bounce(smiley_sprite_2);
	drawSprites();
}
```

Par défaut la masse d'un sprite est égale à 1 (la ligne smiley_sprite_1.mass=1; ne sert donc à rien).

3.3. À faire vous même

En repartant de l'exemple précédent, augmentez suffisamment la masse de smiley_sprite_2 pour avoir l'effet "balle de ping pong contre boule de pétanque".

Il est possible de rendre un sprite "insensible" à la collision (comme s'il avait une masse très élevée) grâce à l'attribut immovable.

Un smiley_sprite_2.immovable=true; rendra smiley_sprite_2 "indéplaçable" en cas de collision (quelle que soit sa masse).

3.4. À faire vous même

Saisissez le code suivant dans le fichier script.js et testez cet exemple :

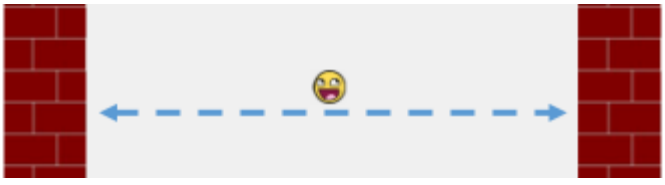
```
var img;
var smiley_sprite_1;
var smiley_sprite_2;
function preload()
{
	img=loadImage("asset/smiley.png");
}

function setup()
{
  createCanvas(800,300);
  smiley_sprite_1=createSprite(100,150);
  smiley_sprite_1.addImage(img);
  smiley_sprite_1.setVelocity(3,0);
  smiley_sprite_1.mass=1;
  smiley_sprite_2=createSprite(600,150);
  smiley_sprite_2.addImage(img);
  smiley_sprite_2.immovable=true;
}

function draw()
{
	background(240,240,240);
	smiley_sprite_1.bounce(smiley_sprite_2);
	drawSprites();
}
```

3.5. À faire vous même

En utilisant l'image que vous trouverez ici, écrivez un programme permettant d'obtenir un smiley qui rebondit entre les deux murs.



3.6. À faire vous même

La méthode **bounce** peut prendre un second paramètre : une fonction de "**callback**". Cette fonction de "callback" sera exécutée uniquement en cas de collision.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite_1;
var smiley_sprite_2;
var cont=false;
function preload()
{
	img=loadImage("asset/smiley.png");
}

function setup()
{
  createCanvas(800,300);
  smiley_sprite_1=createSprite(100,150);
  smiley_sprite_1.addImage(img);
  smiley_sprite_1.setVelocity(3,0);
  smiley_sprite_1.mass=1;
  smiley_sprite_2=createSprite(600,150);
  smiley_sprite_2.addImage(img);
  smiley_sprite_2.immovable=true;
}

function draw()
{
	background(240,240,240);
	smiley_sprite_1.bounce(smiley_sprite_2,contact);
	if (cont==true)
		text("il y a eu contact !",350,100)
	drawSprites();
}
function contact()
{
	cont=true;
}
```

Ci-dessus la fonction contact sera exécutée uniquement s'il y a un contact entre smiley_sprite_1 et smiley_sprite_2.

3.7. À faire vous même

La fonction "text" permet d'afficher une chaîne de caractères. Cette fonction prend 3 paramètres : la chaîne de caractères, la coordonnée x du coin "haut-gauche" du texte, la coordonnée y du coin "haut-gauche" du texte .

Saisissez et testez ce code :

```
function setup()
{
   createCanvas(200,200);
   background(240);
   fill(0);
   text("Hello World",60,100);
}
function draw()
{
}
```

- "fill" permet de choisir la couleur du texte.
- Il est possible de choisir la taille de la police de caractère avec la fonction "textSize" qui prend pour unique paramètre la taille de la police de caractère.
- Les 2 derniers paramètres de la fonction "text" correspondent aux coordonnées du coin "haut-gauche" du texte. Il est possible de modifier ce comportement par défaut grâce à la fonction "textAlign". Cette fonction prend un seul paramètre qui peut être : "LEFT", "RIGHT" ou "CENTER".

3.8. À faire vous même

Sachant que la méthode **remove** appliquée à un sprite supprime ce sprite (smiley_sprite_1.remove() supprime le sprite smiley_sprite_1), en repartant de ce qui a été fait ci-dessus, écrivez un programme qui permettra d'obtenir ceci : « lorsque que le smiley_sprite_1 rencontre le smiley_sprite_2, le smiley_sprite_2 disparaît ».

3.9. À faire vous même

Il est aussi possible de gérer les collisions sans pour autant gérer les rebonds en utilisant la méthode **overlap** à la place de la méthode bound.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite_1;
var smiley_sprite_2;
function preload()
{
        img=loadImage("asset/smiley.png");
}

function setup()
{
   createCanvas(800,300);
   smiley_sprite_1=createSprite(100,150);
   smiley_sprite_1.addImage(img);
   smiley_sprite_1.setVelocity(3,0);
   smiley_sprite_1.mass=1;
   smiley_sprite_2=createSprite(600,150);
   smiley_sprite_2.addImage(img);
```

```
   smiley_sprite_2.immovable=true;
}

function draw()
{
        background(240,240,240);
        smiley_sprite_1.overlap(smiley_sprite_2,contact);
        drawSprites();
}

function contact()
{
        smiley_sprite_2.remove();
}
```

3.10. À faire vous même

En repartant de l'exemple vu dans le "À faire vous même 3.5", écrivez un code permettant d'obtenir ceci : « le smiley rebondit entre les deux murs et au bout du troisième rebonds sur le mur de droite, celui-ci disparaît ».

L'attribut **removed** permet de savoir si un sprite existe encore : smiley_sprite_1.removed est true si le sprite smiley_sprite_1 n'existe pas (ou plus).

3.11. À faire vous même

Il est possible de faire rebondir un sprite sur les bords de la fenêtre de jeu en utilisant des sprites. Inutile d'associer une image à ces sprites car ils seront placés en dehors de la fenêtre de jeu, en revanche, il sera nécessaire de préciser la taille des ces sprites en ajoutant deux arguments à la méthode createSprite (quand on associe une image au sprite, c'est la taille de l'image qui définie la taille du sprite). La méthode createSprite s'écrira createSprite(positionX, positionY, tailleX, taille Y).

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite;
function preload()
{
   img=loadImage("asset/smiley.png")
}

function setup()
{
   createCanvas(800,300);
   smiley_sprite=createSprite(random(20,780),random(20,280));
   smiley_sprite.addImage(img);
   smiley_sprite.setVelocity(random(-5,5),random(-5,5));
   wall_g=createSprite(-5,150,10,300);
   wall_d=createSprite(805,150,10,300);
```

```
   wall_h=createSprite(400,-5,800,10);
   wall_b=createSprite(400,305,800,10);
   wall_d.immovable=true;
   wall_g.immovable=true;
   wall_h.immovable=true;
   wall_b.immovable=true;
}

function draw()
{
   background(240,240,240);
   smiley_sprite.bounce(wall_g);
   smiley_sprite.bounce(wall_d);
   smiley_sprite.bounce(wall_b);
   smiley_sprite.bounce(wall_h);
   drawSprites();
}
```

4. Les groupes de sprites

4.1. À faire vous même

Écrivez un programme permettant d'afficher 2 sprites à l'écran. Ces 2 sprites devront avoir une vitesse d'origine et une position d'origine aléatoires. Les 2 sprites devront rebondir sur les bords de la fenêtre de jeu.

4.2. À faire vous même

Imaginez que maintenant nous désirions créer 10 sprites. Les choses commenceraient à se compliquer. Pour nous simplifier la tâche, p5.play nous propose de créer un groupe.

Saisissez le code suivant dans le fichier script.js et testez cet exemple :

```
var img;
var group_smiley;
function preload()
{
    img=loadImage("asset/smiley.png")
}

function setup()
{
    createCanvas(800,300);
    group_smiley=new Group();
    for (i=0;i<10;i=i+1){
        var smiley_sprite;
        smiley_sprite=createSprite(random(20,780),random(20,280));
        smiley_sprite.addImage(img);
        smiley_sprite.setVelocity(random(-5,5),random(-5,5));
        group_smiley.add(smiley_sprite);
        }
    wall_g=createSprite(-5,150,10,300);
```

```
    wall_d=createSprite(805,150,10,300);
    wall_h=createSprite(400,-5,800,10);
    wall_b=createSprite(400,305,800,10);
    wall_d.immovable=true;
    wall_g.immovable=true;
    wall_h.immovable=true;
    wall_b.immovable=true;
}

function draw() {
    background(240,240,240);
    group_smiley.bounce(wall_g);
    group_smiley.bounce(wall_d);
    group_smiley.bounce(wall_b);
    group_smiley.bounce(wall_h);
    drawSprites();
}
```

Analysons le code ci-dessus :

1. Nous commençons par créer un groupe : *group_smiley=new Group();* (ce groupe se nommera *group_smiley*.
2. La boucle *for* permet de créer 10 sprites. Grâce à la méthode *add* (*group_smiley.add(smiley_sprite);*) chaque sprite est "rangé" dans le groupe *group_smiley*
3. Pour la collision avec les bords de la fenêtre, nous utilisons la méthode *bounce* avec le groupe *group_smiley* (exemple : *group_smiley.bounce(wall_g);*, grâce à cette ligne, tous les sprites appartenant au groupe *group_smiley* rebondiront sur le bord gauche de la fenêtre de jeu).

4.3. À faire vous même

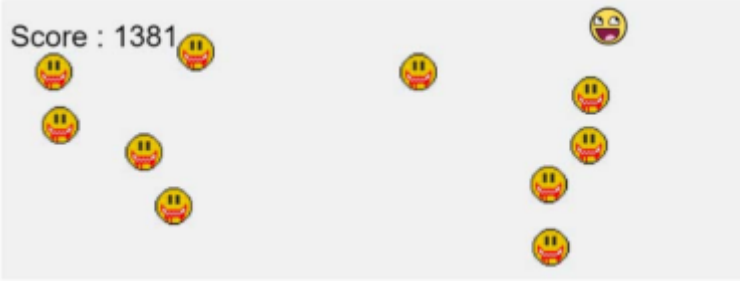
Pour qu'un sprite appartenant à un groupe *group_smiley* rebondisse sur les autres membres du groupe, il suffit d'écrire : *group_smiley.bounce(group_smiley);*.

Écrivez un programme permettant d'afficher 20 sprites à l'écran. Ces 20 sprites devront avoir une vitesse d'origine et une position d'origine aléatoires. Les 20 sprites devront rebondir sur les bords de la fenêtre de jeu et entres eux.

4.4. À faire vous même

Écrivez un programme permettant d'obtenir d'afficher 1 sprite « rigolant » et 10 sprites « sanglant » à l'écran. Ces 11 sprites devront avoir une vitesse d'origine et une position d'origine aléatoires. Les 11 sprites devront rebondir sur les bords de la fenêtre de jeu et entres eux. À chaque collision entre le sprite « rigolant » et un des sprites « sanglant » le score augmente d'une unité.

Vous pouvez télécharger ici l'image du sprite « sanglant ».



4.5. À faire vous même

Saisissez le code suivant dans le fichier script.js et testez cet exemple :

```
var img_1;
var img_2;
var smiley_sprite;
var group_smiley;
var score=0;
function preload()
{
	img_1=loadImage("asset/smiley.png")
	img_2=loadImage("asset/smiley_blood.png")
}

function setup()
{
	createCanvas(800,300);
	textSize(32);
	smiley_sprite=createSprite(random(20,780),random(20,280));
	smiley_sprite.addImage(img_1);
	smiley_sprite.setVelocity(random(-5,5),random(-5,5));
	group_smiley=new Group();

	for (i=0; i<10; i=i+1) {
		var smiley_sprite_blood;
		smiley_sprite_blood=createSprite(random(20,780),random(20,280));
		smiley_sprite_blood.addImage(img_2);
		smiley_sprite_blood.setVelocity(random(-5,5),random(-5,5));
		group_smiley.add(smiley_sprite_blood);
		}

	wall_g=createSprite(-5,150,10,300);
	wall_d=createSprite(805,150,10,300);
	wall_h=createSprite(400,-5,800,10);
	wall_b=createSprite(400,305,800,10);
	wall_d.immovable=true;
	wall_g.immovable=true;
	wall_h.immovable=true;
	wall_b.immovable=true;
```

```
}

function draw()
{
	background(240,240,240);
	group_smiley.bounce(wall_g);
	group_smiley.bounce(wall_d);
	group_smiley.bounce(wall_b);
	group_smiley.bounce(wall_h);
	smiley_sprite.bounce(wall_g);
	smiley_sprite.bounce(wall_d);
	smiley_sprite.bounce(wall_b);
	smiley_sprite.bounce(wall_h);
	group_smiley.bounce(group_smiley);
	smiley_sprite.overlap(group_smiley,contact);
	text("Score : "+score,10,50)
	drawSprites();
}

function contact(sprite_1,sprite_2)
{
	sprite_2.remove();
	score=score+1;
}
```

Comme vous pouvez le constater dans le code précédent, la fonction de callback contact peut prendre en paramètres les 2 sprites concernés par la collision. Si vous avez X.overlap(Y,contact); et function contact(sprite_1,sprite_2) alors X correspond à sprite_1 et Y correspond à sprite_2 (avec X et Y des sprites). Si X ou Y est un groupe de sprite (comme dans l'exemple ci-dessus), le fonctionnement reste le même : sprite_1 ou sprite$_2$ correspond au sprite du groupe qui a été concerné par la collision.

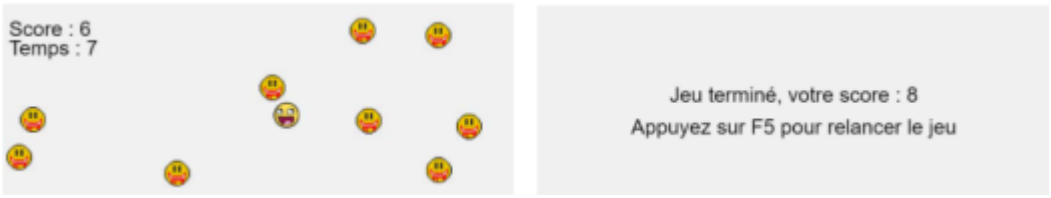
4.6. À faire vous même

Pour terminer cette activité, vous allez créer un petit jeu vidéo (il faut cliquer sur la fenêtre de jeu à l'aide du pointeur de la souris faute de quoi vous n'arriverez pas à diriger le sprite à l'aide des flèches).

Écrivez un programme permettant :

- D'afficher 1 sprite « rigolant » et 10 sprites « sanglant » à l'écran.
- Les 10 sprites « sanglant » devront avoir une vitesse d'origine et une position d'origine aléatoires.
- Le sprite « rigolant » pourra être déplacé à l'aide des flèches directionnelles.
- Les 10 sprites « sanglant » devront rebondir sur les bords de la fenêtre de jeu et entres eux.
- Le sprite « rigolant » ne pourra pas sortir de la fenêtre.
- À chaque collision entre le sprite "rigolant" et un des sprites "sanglants" le score augmente d'une unité et le sprite « sanglant » disparaît.

- La durée de la partie est de 8s.
- Pendant la partie, le score et le temps doivent être affichés.
- À la fin de la partie, le score est affiché au centre de la fenêtre.



Le sprite "souriant" est dirigé par le joueur à l'aide des flèches du clavier. Ce sprite doit attraper le plus possible de sprite "sanglant" en moins de 8 secondes.

Pour vous aider :

- la fonction *millis()* renvoie le nombre de millisecondes écoulées depuis que le lancement du programme.
- la fonction *round* permet d'arrondir une valeur : *round(4.3)* renvoie 4.

5. Les forces extérieures

La méthode ***addSpeed*** permet de modifier la vitesse d'un sprite. D'un point de vue physique, en simplifiant les choses, nous pouvons dire que la modification de la vitesse d'un sprite est similaire à l'application d'une force sur ce même sprite (pour les puristes, cela n'est pas tout à fait vrai puisque la modification de vitesse d'un objet A, aussi appelée accélération de A, est égale à la force divisée par la masse de A). Dans toute la suite de cette activité, nous considérerons la méthode *setSpeed* comme une méthode nous permettant d'appliquer une force sur un sprite.

Cette méthode *addSpeed* prend 2 paramètres : la valeur de la "force" et le couple "direction + sens" de la force (une force est modélisée par une "flèche" appelée vecteur en mathématiques, une force a donc une intensité (une valeur), une direction et un sens).

Cette direction et ce sens sont représentés par un angle :

- si l'angle est égal à 0 : direction : horizontale, sens : vers la droite
- si l'angle est égal à 90 : direction : verticale, sens : vers le bas
- si l'angle est égal à 180 : direction : horizontale, sens : vers la gauche
- si l'angle est égal à 270 : direction : verticale, sens : vers le haut

5.1. À faire vous même

Nous allons commencer par modéliser le poids d'un objet (le poids d'un objet A est la force exercée par la Terre sur l'objet A). Cette force est modélisée par un vecteur vertical dirigé vers le bas.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite;
function preload()
{
        img=loadImage("asset/smiley.png");
}
```

```
function setup()
{
        createCanvas(800,300);
        smiley_sprite=createSprite(400,20);
        smiley_sprite.addImage(img);
}

function draw()
{
        background(240,240,240);
        smiley_sprite.addSpeed(0.1,90);
        drawSprites();
}
```

Nous avons placé *smiley_sprite.addSpeed(0.1,90)* dans la fonction *draw* car le poids s'applique sur le sprite en "permanence" (à chaque image). Si vous désirez appliquer une force "ponctuellement" (par exemple une "pichenette" avec les doigts sur une bille), il ne faudra pas placer *addSpeed* dans la fonction *draw*.

5.2. À faire vous même

Rajoutons un "sol" afin de faire rebondir le sprite.

Saisissez le code suivant dans le fichier script.js et testez cet exemple :

```
var img;
var smiley_sprite;
var sol;
function preload()
{
        img=loadImage("asset/smiley.png");
}

function setup()
{
        createCanvas(800,300);
        smiley_sprite=createSprite(400,20);
        smiley_sprite.addImage(img);
        sol=createSprite(400,305,800,10);
        sol.immovable=true;
}

function draw()
{
        background(240,240,240);
        smiley_sprite.addSpeed(0.2, 90);
        smiley_sprite.bounce(sol);
        drawSprites();
}
```

En testant l'exemple ci-dessus, vous avez dû remarquer que la balle rebondissait de plus en plus haut, ce qui est évidemment impossible. Au mieux, le choc contre le sol est dit "élastique" (l'énergie de la balle se conserve : la balle rebondit toujours à la même hauteur). Ici, la balle gagne de l'énergie à chaque rebond, ce qui est problématique (visiblement, il y a un problème avec la gestion des rebonds dans p5.play)

5.3. À faire vous même

Afin de rendre cette scène plus réaliste, nous allons rajouter une force de "frottements" qui s'appliquera à chaque rebond.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var img;
var smiley_sprite;
var sol;
function preload()
{
        img=loadImage("asset/smiley.png");
}

function setup()
{
        createCanvas(800,300);
        smiley_sprite=createSprite(400,30);
        smiley_sprite.addImage(img);
        sol=createSprite(400,305,800,10);
        sol.immovable=true;
}

function draw()
{
        background(240,240,240);
        smiley_sprite.addSpeed(0.2, 90);
        smiley_sprite.bounce(sol,frottement);
        drawSprites();
}
function frottement()
{
        smiley_sprite.addSpeed(2,90);
}
```

Dans l'exemple ci-dessus, à chaque rebond, la fonction *frottement* est exécutée. Cette fonction permet d'appliquer une force (*smiley_sprite.addSpeed(2,90)*) qui permettra de diminuer l'energie du sprite à chaque rebond.

5.4. À faire vous même

Écrivez un programme permettant :

- D'afficher un sprite « posé » sur le sol au milieu de la fenêtre.
- À chaque appui sur la touche « Entrée » le sprite doit effectuer un « saut » et revenir sur le sol sans rebondir.
- Lorsque le sprite est en « l'air », l'appuie sur la touche « Entrée » n'a aucun effet.

Vous devez remarquer qu'il est impossible de sauter si le sprite est en l'air.

6. L'animation des sprites

Dans cette dernière activité, nous allons travailler sur l'animation des sprites. Pour suivre cette activité, il est nécessaire de télécharger cette archive. Une fois cette archive téléchargée, placez les

images contenues dans cette archive dans le dossier "asset".

L'animation d'un sprite est basée sur le principe du dessin animé : le "défilement" d'une série d'images fixes donne l'illusion du mouvement.

Avant de pouvoir animer un sprite, il faudra créer une animation. Il existe plusieurs méthodes permettant de créer une animation, nous allons en étudier deux.

6.1. À faire vous même

Saisissez le code suivant dans le fichier script.js et testez cet exemple :

```
var perso;
var perso_walk;
function preload()
{
        perso_walk = loadAnimation(
            "asset/ghost_walk0001.png", "asset/ghost_walk0002.png",
            "asset/ghost_walk0003.png", "asset/ghost_walk0004.png");
}

function setup()
{
        createCanvas(800,300);
        perso=createSprite(400,150);
        perso.addAnimation("walk",perso_walk);
        perso.changeAnimation("walk");
}

function draw()
{
        background(240,240,240);
        drawSprites();
}
```

Analysons le code ci-dessus :

La méthode ***loadAnimation*** permet de créer une animation (nous "stockons" cette animation dans la variable *perso_walk*). La méthode *loadAnimation* prend en paramètres les différentes images qui composent l'animation : *perso_walk = loadAnimation("asset/ghost_walk0001.png", "asset/ghost_walk0002.png","asset/ghost_walk0003.png", "asset/ghost_walk0004.png");*

Une fois l'animation créée, il faut l'associer à un sprite en utilisant la méthode ***addAnimation*** : *perso.addAnimation("walk",perso_walk);*. Cette méthode prend en paramètres le nom de l'animation (*walk*) et l'animation elle-même (*perso_walk*).

La méthode ***changeAnimation*** permet de "jouer" l'animation, elle prend en paramètre le nom de l'animation à jouer : *perso.changeAnimation("walk")*

6.2. À faire vous même

Il est possible d'associer plusieurs animations à un sprite.

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var perso;
var perso_walk;
function preload()
```

```
{
        perso_walk = loadAnimation(
            "asset/ghost_walk0001.png", "asset/ghost_walk0002.png",
            "asset/ghost_walk0003.png", "asset/ghost_walk0004.png");
        perso_wait = loadAnimation(
            "asset/ghost_standing0001.png","asset/ghost_standing0002.png",
            "asset/ghost_standing0003.png","asset/ghost_standing0004.png",
            "asset/ghost_standing0005.png","asset/ghost_standing0006.png");
}

function setup()
{
        createCanvas(800,300);
        perso=createSprite(400,150);
        perso.addAnimation("walk",perso_walk);
        perso.addAnimation("wait",perso_wait);
        perso.changeAnimation("wait");

}

function draw()
{
        background(240,240,240);
        drawSprites();
}

function keyPressed()
{
      if ( keyCode == 37 ) {
            perso.mirrorX(-1);
            perso.changeAnimation("walk");
            }
      else if ( keyCode == 39 ) {
            perso.mirrorX(1);
            perso.changeAnimation("walk");
            }
            else
                  perso.changeAnimation("wait");
}

function keyReleased()
{
      perso.changeAnimation("wait");
}
```

Seule nouveauté dans l'exemple ci-dessus : l'utilisation de la méthode ***mirrorX*** (*perso.mirrorX(-1)* et *perso.mirrorX(1)*). Comme vous l'avez sans doute remarqué, dans le dossier "asset", nous n'avons pas d'image du personnage "regardant" vers la gauche. Pourtant, en appuyant sur la flèche gauche du clavier, nous avons bien une animation du personnage vers la gauche.

Ceci est possible grâce à la méthode *mirrorX* qui permet d'avoir "l'animation symétrique" (*perso.mirrorX(-1)* permet d'avoir "l'animation symétrique", *perso.mirrorX(1)* permet de retrouver l'animation dans le "bon sens".

P5.play propose aussi, pour créer des animations, d'utiliser des spritesheets (les images nécessaires à l'animation sont regroupées dans un unique fichier)

Exemple de spritesheet :

ATTENTION : les exemples ci-dessous (6.3 et 6.4) ne fonctionnent pas avec les dernières versions de p5js. Vous devez donc utiliser une ancienne version de p5js à télécharger ici (clic de souris droit puis "Enregistrer sous") pour travailler sur les deux exemples suivants. N'oubliez pas, si nécessaire, de modifier le fichier index.html."

6.3. À faire vous même

Saisissez le code suivant dans le fichier script.js et testez cet exemple

```
var perso;
var perso_walk;
function preload()
{
	perso_sprite_sheet_walk = loadSpriteSheet('asset/sonic_1.png', 48, 48, 6);
}
function setup()
{
	createCanvas(800,300);
	perso=createSprite(400,150);
	perso_walk=loadAnimation(perso_sprite_sheet_walk);
	perso_walk.frameDelay=4;
	perso.addAnimation("walk",perso_walk);
	perso.changeAnimation("walk");
}
function draw()
{
	background(240,240,240);
	drawSprites();
}
```

Analysons le code ci-dessus :

- La méthode ***loadSpriteSheet*** (*perso_sprite_sheet_walk = loadSpriteSheet('asset/sonic_1.png', 48, 48, 6)*) va nous permettre d'utiliser un spritesheet pour créer notre animation. Cette méthode prend 4 paramètres : le chemin vers l'image qui servira de spritesheet, la largeur du sprite, la hauteur du sprite et le nombre d'images utilisé.
- La méthode ***loadAnimation*** prend en paramètre le nom du spritesheet (*loadAnimation(perso_sprite_sheet_walk)*).
- *frameDelay* (*perso_walk.frameDelay=4*) permet de modifier la vitesse de l'animation : plus la valeur est petite, plus l'animation est rapide (les différentes images défilent plus rapidement). Par défaut, *frameDelay* a pour valeur 2.

6.4. À faire vous même

Écrivez un programme permettant :

- D'afficher l'animation sonic au milieu de la fenêtre
- Lorsque l'utilisateur n'appuie sur aucune touche, sonic attend en tapant du pied.
- Lorsque l'utilisateur appui sur les flèches de droite ou de gauche, sonic se déplace en courant.

7. Pour aller plus loin...

Il reste beaucoup de choses à découvrir sur p5.play. Pour commencer, il serait bien d'étudier attentivement les exemples proposés sur le site officiel de p5.play : voir ici.

La documentation officielle vous sera d'un grand secours afin d'étudier tous les aspects de p5.play que nous n'avons pas étudiés : voir ici.

Après cela, vous serez prêt à vous lancer dans la création de votre propre jeu !