Phaser 3 : moteur physique

Qu'est-ce qu'un moteur physique ?

Il s'agit du composant qui gère les intéractions physiques entre les différents éléments au sein d'une même scène, qu'il s'agisse d'accélérations ou de collisions.

Au sein de Phaser, nous utilisons le moteur Arcade, accessible via l'objet this.physics.

Créer un sprite

Le moteur physique gère des *sprites*. Il s'agit d'images que le moteur physique peut traiter. Il intègre une Factory : c'est son composant *this.physics.add* qui lui sert à générer de nouveaux *game objects*, qu'il pourra traiter. Pour créer un sprite, passons par cette Factory.

sprite(x, y, key [, frame])

Creates a new Arcade Sprite object with a Dynamic body.

Parameters:

Name	Type	Argument	Description
x	number		The horizontal position of this Game Object in the world.
y	number		The vertical position of this Game Object in the world.
key	string		The key of the Texture this Game Object will use to render with, as stored in the Texture Manager.
frame	string | number	<optional>	An optional frame from the Texture this Game Object is rendering with.

Since: 3.0.0
Source: src/physics/arcade/Factory.js (Line 193)

Returns:

The Sprite object that was created.

Type
Phaser.Types.Physics.Arcade.SpriteWithDynamicBody

On remarque que la spritesheet devra avoir été pré-chargée (dans la fonction *preload*) à la clef (*key*) voulue, comme cela était le cas dans le tutoriel que nous avons fait ensemble.

Gérer un sprite

Nous pouvons voir que SpriteWithDynamicBody renvoie à la classe Body

SpriteWithDynamicBody

Type:

- object

Properties:

Name	Type
body	Phaser.Physics.Arcade.Body

Source: src/physics/arcade/typedefs/SpriteWithDynamicBody.js (Line 1)

Prenez donc le temps de l'explorer.
Ce sont là tous les attributs et méthodes d'un sprite géré par notre moteur physique. De quoi gérer le positionnement, la vitesse, l'accélération, la gravité, etc…

Colliders & overlaps

La Factory du moteur physique nous permet aussi de créer des colliders.

Syntaxe d'exemple :
this.physics.add.collider(elem1, elem2, fctConsequence,fctCondition,this);

- *elem1* et *elem2* sont les deux éléments dont on veut gérer la collision
- *fctConsequence* est la fonction appelée quand la collision a lieu
- *fctCondition* est une fonction appelée juste avant de déclencher la collision. Si la fonction rend *false*, alors la collision est désactivée. Si elle rend *true*, alors la collision aura bien lieu et *fctConsequence* sera appelée.

fctConsequence et fctCondition sont ici à écrire sans parenthèses ni paramètres. Il faudra toutefois bien penser à les déclarer dans votre code (en dehors des fonctions update, create…). Si elles sont inutiles, on peut les remplacer par le mot-clé *null*.

Il y a aussi des *overlap*. Ils sont créés de la même façon que les colliders, avec la méthode *this.physics.add.overlap* plutôt que *this.physics.add.collider*. En fait, ce sont des colliders, mais qui n'empêchent plus aux deux éléments de se superposer, et ne déclenchent donc pas automatiquement d'interruption du déplacement ni de rebond.