Atelier Pixels et origine

Langage : Lua
Framework : Love2D

Lien de l'atelier :
https://www.gamecodeur.fr/atelier-love2d-pixels-origine/

Introduction

Il est important de bien maîtriser les notions de coordonnées et d'origine d'affichage des images dans un jeu vidéo en 2D.

Ces notions sont indispensables pour :

- Afficher des images sur l'écran du jeu, à des coordonnées précises, à gérer leur alignement, etc.
- Gérer les collisions et autres détections de positions
- Gérer les rotations autour d'un axe

Cet atelier vous donnera des informations et des outils pour intégrer ces notions et les expérimenter.

Sommaire :

Ecran et coordonnées x et y

Même si cela peut paraître trivial pour certains, je préfère revoir cette notion de base.

Un écran de jeu est composé de **pixels**. Leur taille et leur nombre dépend de la **résolution** de l'ordinateur au moment où l'écran est affiché. Une résolution est donc exprimée en pixels.

En premier le nombre de pixels horizontaux, en deuxième le nombre de pixels verticaux.

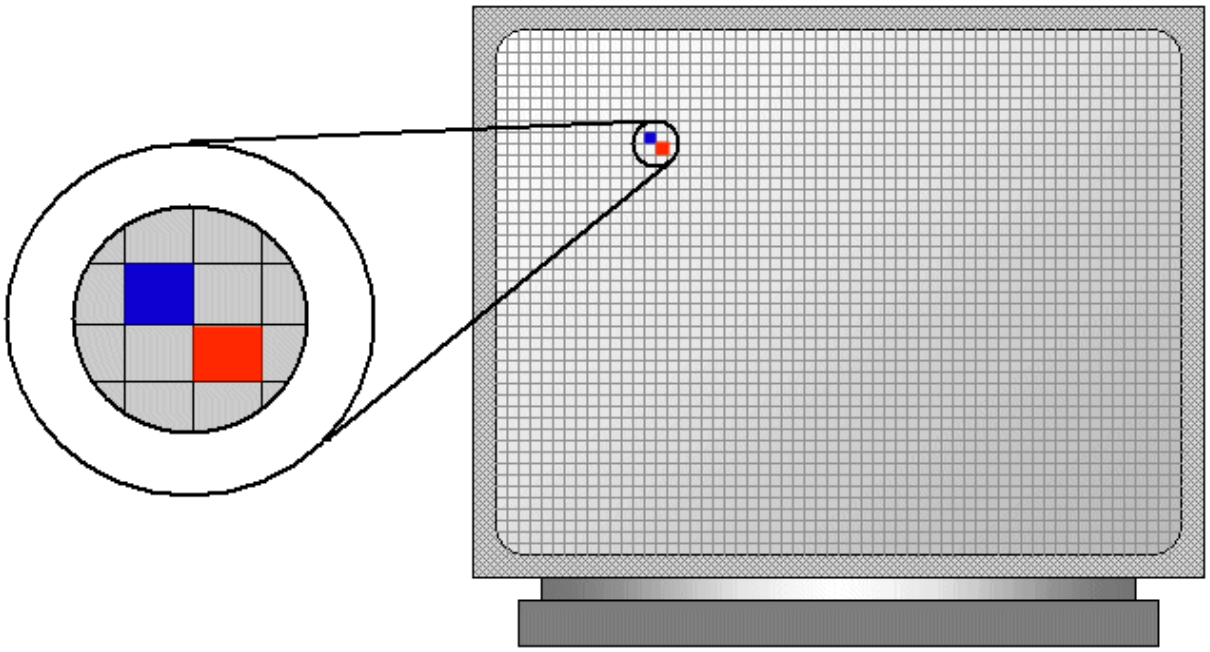
Vocabulaire anglais :

largeur : width
hauteur : height

Moyen mnémotechnique pour retenir lequel veut dire hauteur et l'autre largeur, et leur orthographe :

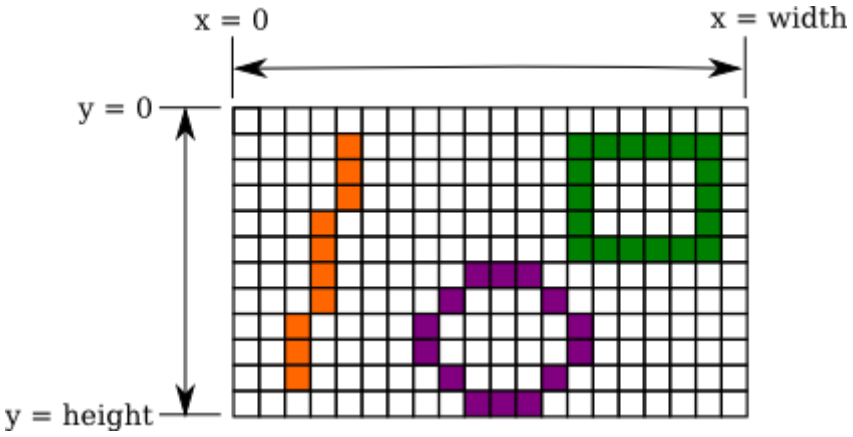
> ***h**eight commence par la lettre **h** comme **h**auteur et se termine par "j'ai acheté" ("g" "h" "t") alors que dans width le **h** est à la fin.*

Exemple : 1024,768 ou 1024x768 (on dit alors 1024 par 768), ce qui signifie 1024 pixels horizontaux (width) , et 768 pixels verticaux (height).



Les pixels, en programmation, sont numérotés à partir de 0, de gauche à droite et de haut en bas et la position horizontale est donnée en premier. On parle parfois de **colonnes** et de **lignes**.

La coordonnée la plus en haut à gauche est donc 0,0.

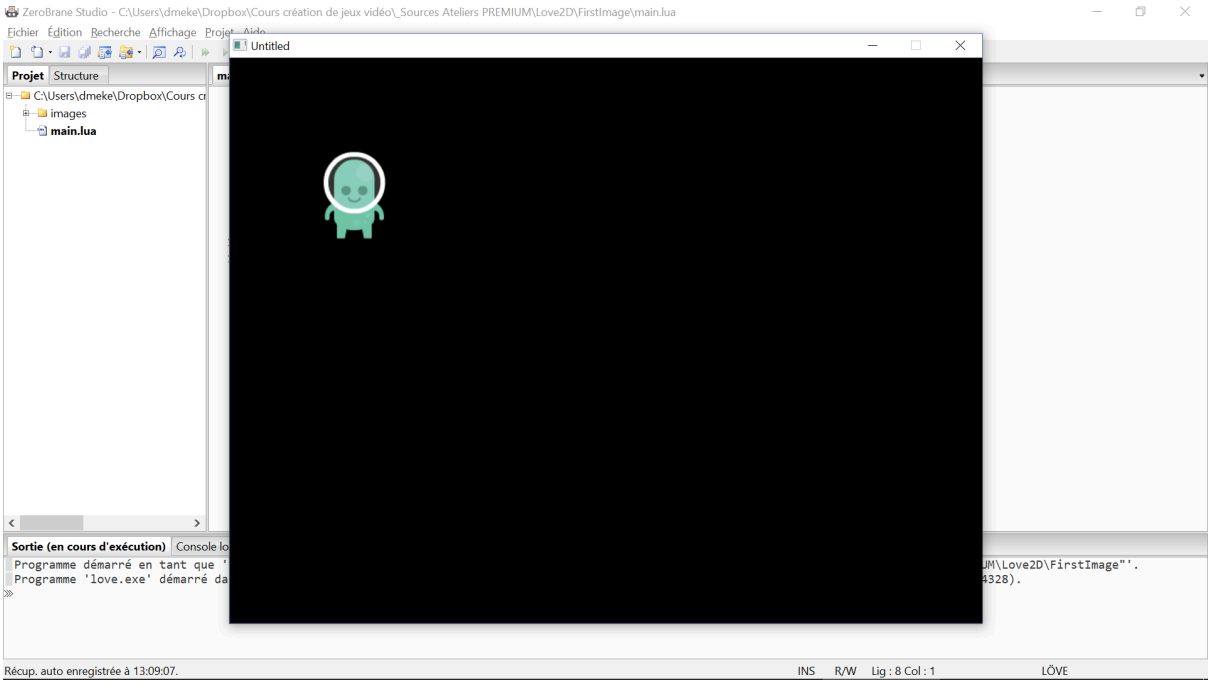


(source : http://math.hws.edu/javanotes/c6/pixel-coordinates.png)

L'écran peut être "fenêtré" (windowed) ou plein écran (full screen).

En code, on parle plutôt de "window" (fenêtre) que d'écran (screen).

En mode fenêtré, l'écran du jeu possède une **barre de titre** (title bar) et utilise la taille des pixels du système en cours (windows, linux, OSX…). Il peut être déplacé comme une application :



Dans cette configuration, si on donne à l'écran une taille (en pixels donc) supérieure à la taille de l'écran, la fenêtre ne rentrera pas et elle dépassera.
En mode plein écran par contre, la résolution s'adapte afin que la fenêtre occupe tout l'écran. Ce qui peut entraîner soit des déformations, soit des bandes noires. Au choix du développeur (ce qui dépasse le cadre de cet atelier mais on en reparlera…).

Dans la plupart des frameworks de programmation, des fonctions permettent de connaître la résolution actuelle de l'écran du jeu, et les modes d'affichage que la carte graphique peut supporter (notamment sa résolution maximale)

En Löve2D :

Pour connaître la hauteur (height) actuelle de l'écran du jeu :

```
height = love.graphics.getHeight( )
```

Pour connaître la largeur (width) actuelle de l'écran du jeu :

```
width = love.graphics.getWidth( )
```

Pour connaître les résolutions (modes) supportées :

```
modes = love.window.getFullscreenModes()

-- modes = {
--    { width = 320, height = 240 },
--    { width = 640, height = 480 },
--    { width = 800, height = 600 },
--    { width = 1024, height = 768 },
--    ...
-- }
```

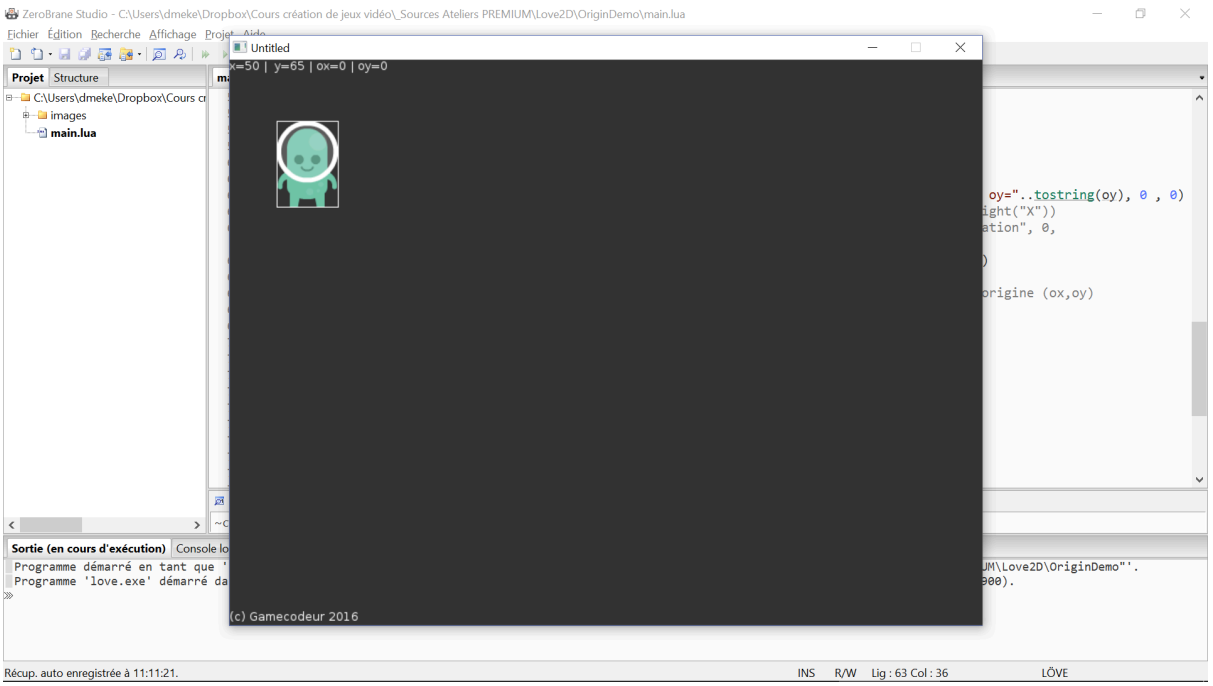
Coordonnées d'affichage d'une image

Lorsqu'on souhaite afficher une image sur l'écran de son jeu, on doit lui donner une position (on parle aussi de coordonnées).

La position est donc exprimée en pixels à l'aide de 2 valeurs : la position horizontale et la position verticale.

Par exemple : 50,65 signifie : Affiché à la position 50 horizontalement, et 65 verticalement. Donc 50 pixels en partant de la gauche et 65 pixels en partant du haut.

Voilà ce que ça donne sur un écran de feu fenêtré, en 800x600 :



Mais une image n'a pas une taille de 1 pixel par 1 pixel ! Donc ce 50,65 désigne quel endroit de l'image ?

C'est là que la notion hyper importante d'origine intervient !

Origine d'affichage d'une image

Par défaut, et dans la plupart des framework 2D, l'origine est en 0,0. Ça veut dire quoi ?

Ca veut dire que la position d'affichage de l'image est exprimée par rapport à son coin supérieur gauche.

Pourquoi 0,0 ??

Imaginez que l'image elle-même soit un écran. Elle a donc une résolution en pixels, par exemple 66x92. Soit 66 pixels de large, et 92 pixels de haut :

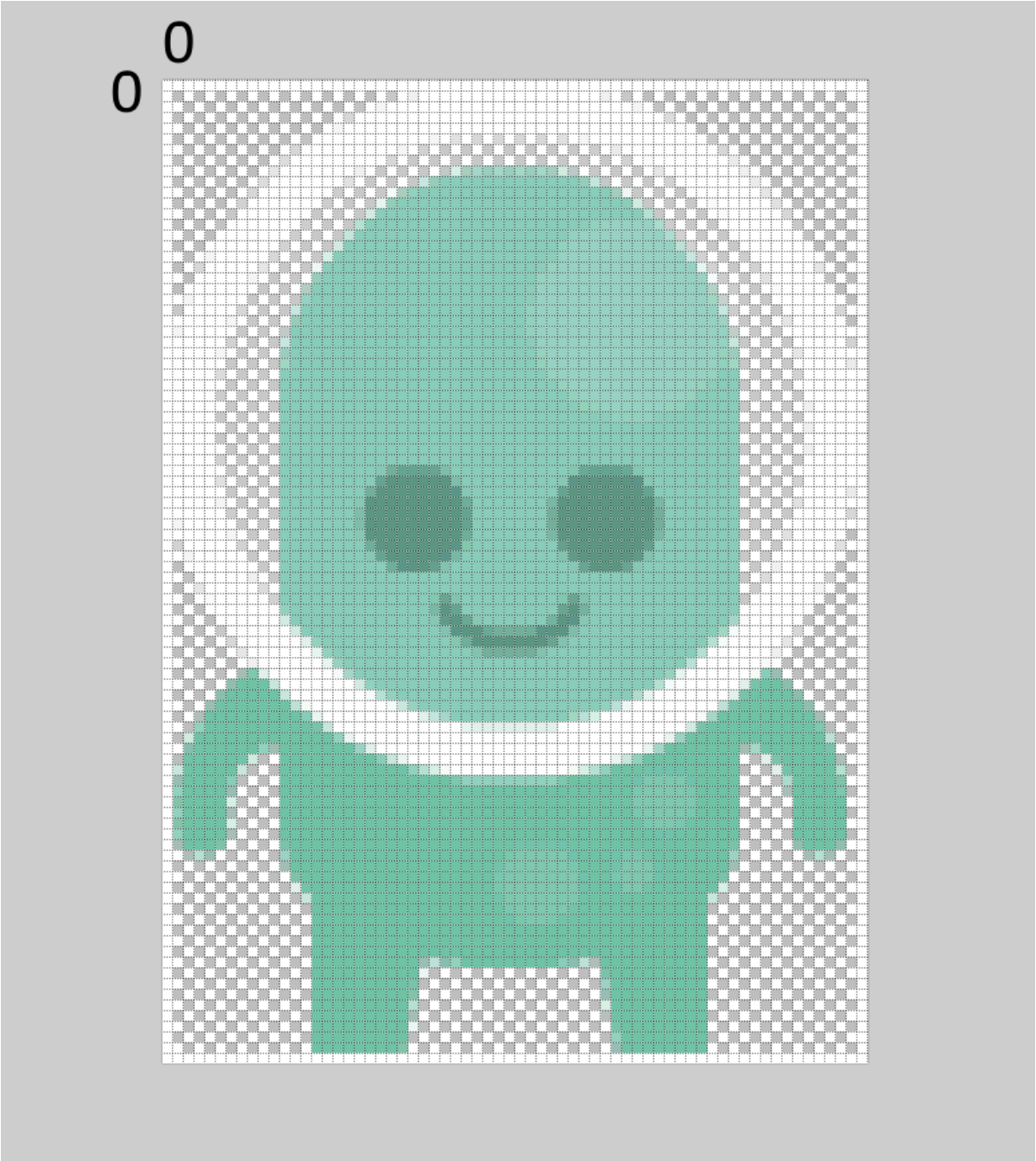


Image de 66x92 pixels

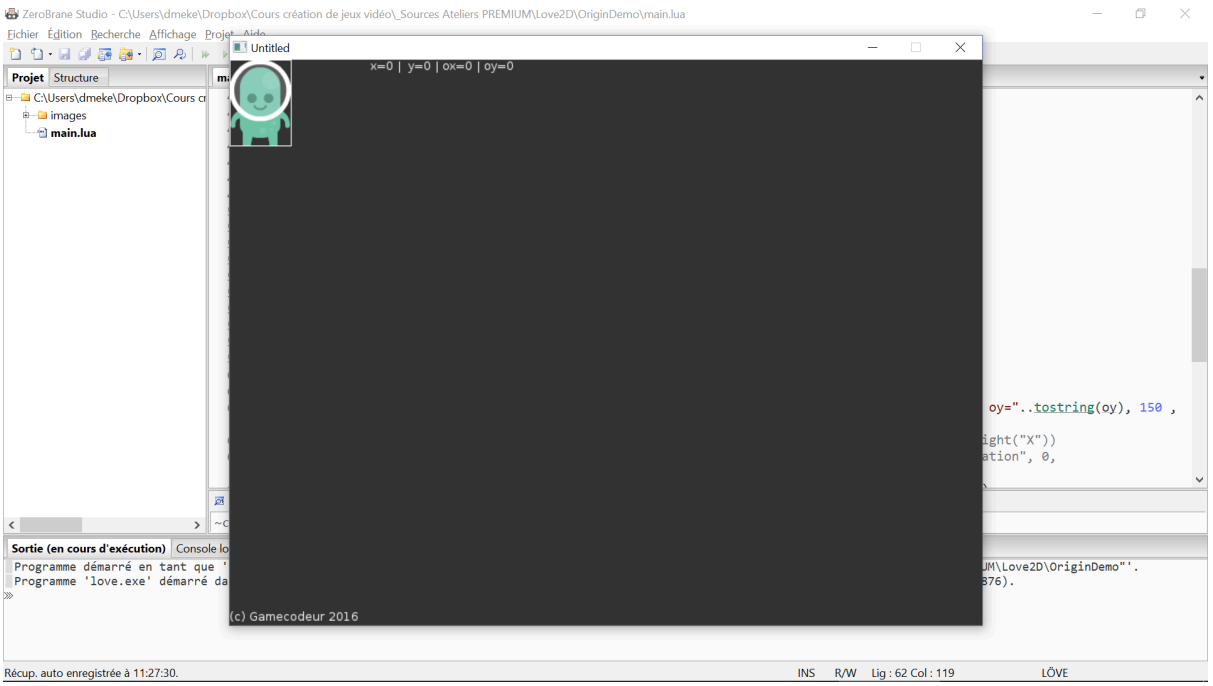
Si on la voit comme un écran, le pixel le plus à gauche est donc le pixel à la colonne 0 et le plus en haut à la ligne 0.

Quant au pixel le plus à droite, c'est le 65ème (de 0 à 65, on a bien 66 pixels), et le plus en bas le 91ème (de 0 à 91, soit 92 pixels).

Vous voyez mieux pourquoi on parle d'origine 0,0 ?

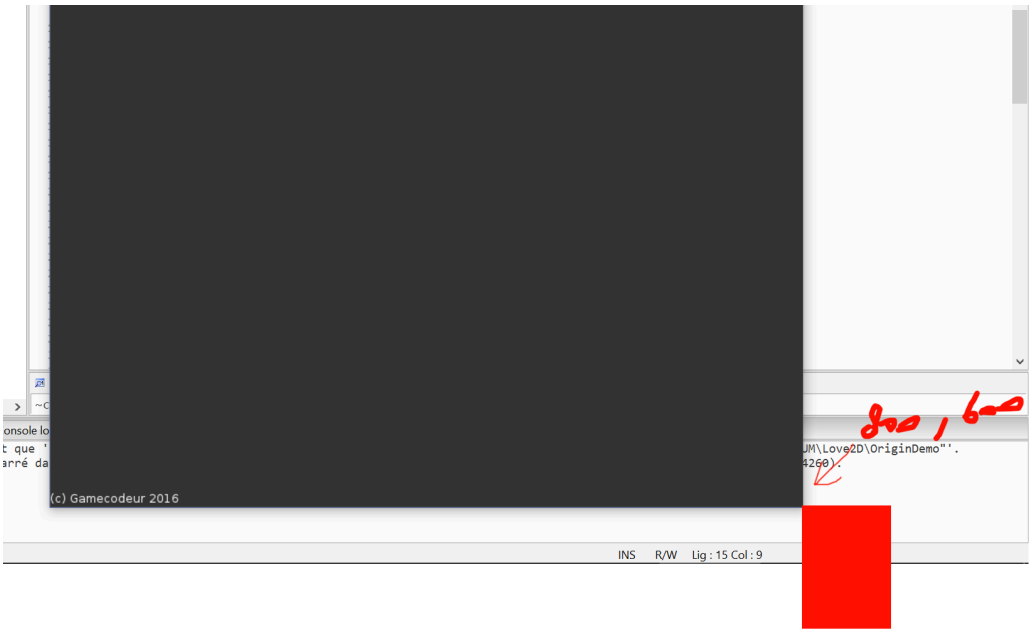
Démonstration avec l'origine 0,0

Pour afficher l'image dont l'origine est 0,0 (par défaut donc) tout en haut à gauche de l'écran, rien de plus simple : on l'affiche aux coordonnées 0,0 :



Mais pour l'afficher tout en bas à droite ?

Si on l'affiche en 800,600 elle sera hors écran :

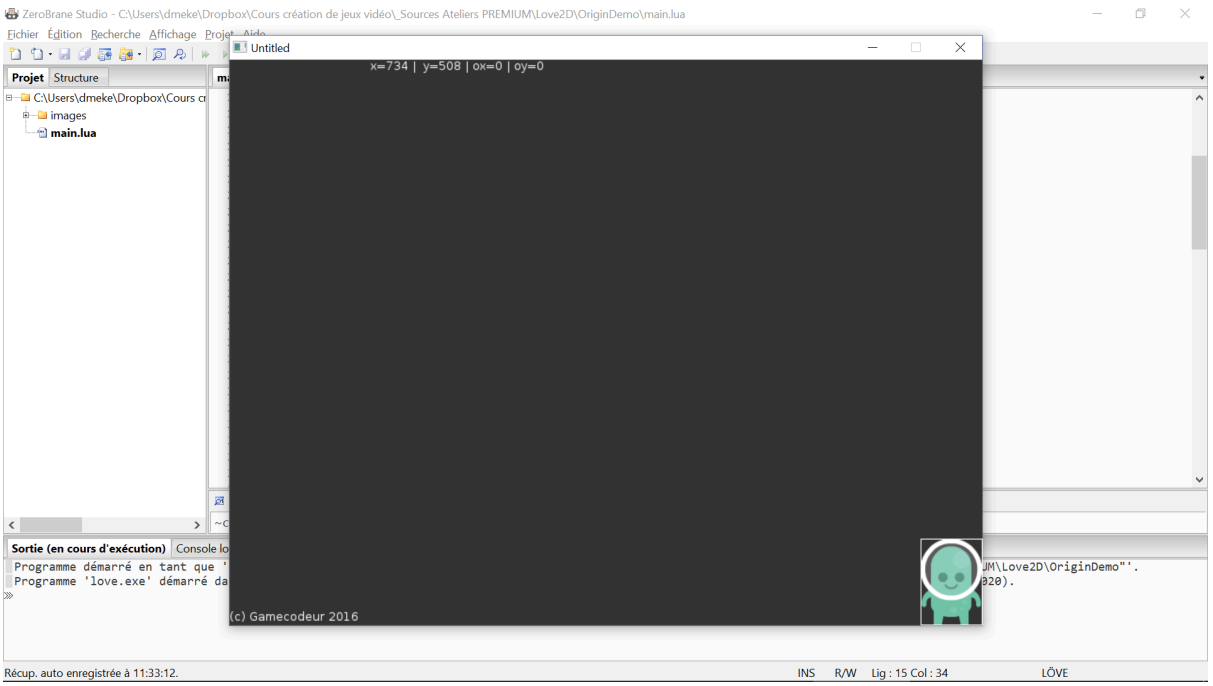


On doit donc se livrer à un calcul simple :

La coordonnée horizontale doit être la largeur de l'écran "moins" la largeur de l'image.

La coordonnée verticale doit être la hauteur de l'écran "moins" la hauteur de l'image.

Ca donne alors ça :



BRAVO !

En Löve on sait déjà obtenir la largeur et la hauteur de l'écran (voir plus haut) mais pour l'image c'est comme ceci :

Obtenir la largeur :

```
largeurImage = image:getWidth()
```

Obtenir la hauteur :

```
hauteurImage = image:getHeight()
```

`image` étant le nom de la variable qui a été utilisée pour charger l'image avec *love.graphics.newImage*.

Exercice : affichez une image en haut à droite, et en bas à gauche…

Décaler l'origine de l'image

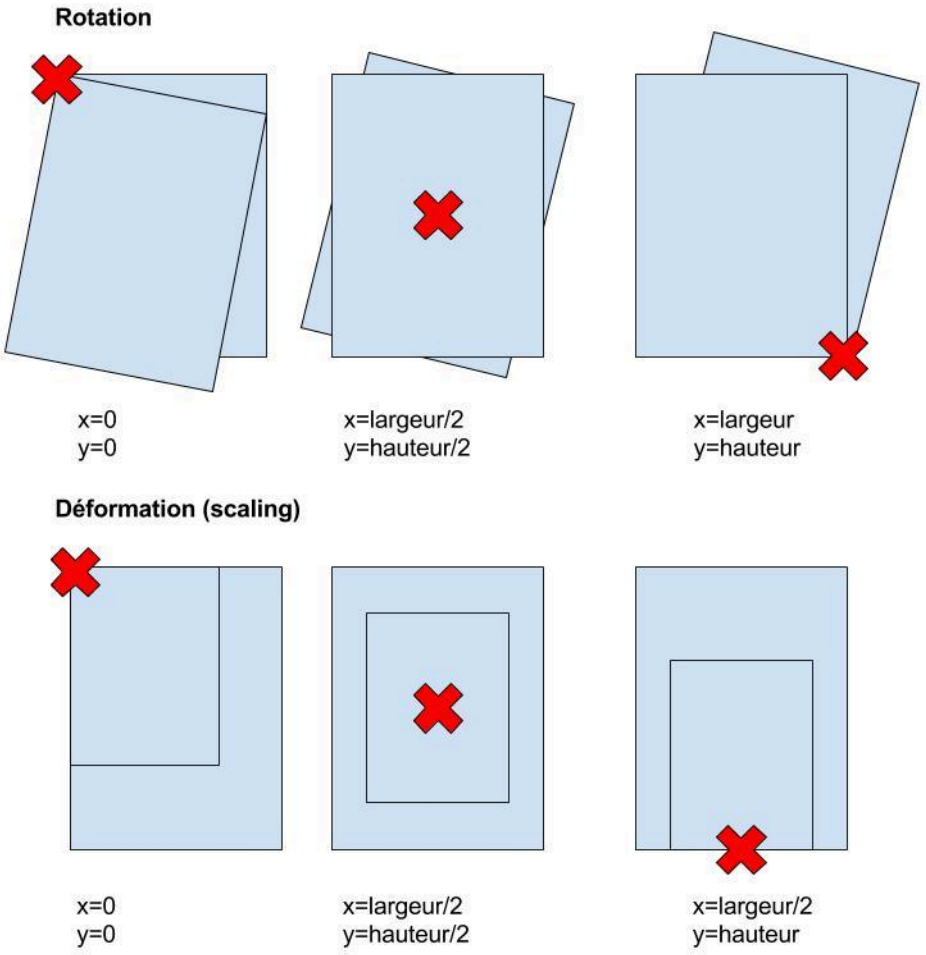
On peut facilement appliquer un décalage à l'origine d'affichage (on parle d'"offset").

Mais pourquoi changer l'origine de l'image ?

Plusieurs raisons possibles :

1. Vous avez besoin de déformer l'image (scaling)
2. Vous avez besoin d'appliquer une rotation à l'image
3. Pour faciliter l'alignement d'images par rapport à quelque chose (le bord de l'écran par exemple)

Ca donne ça :

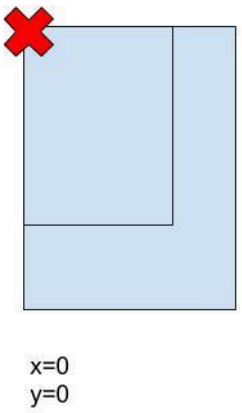


Mais on peut avoir d'autres raisons, notamment une préférence du programmeur...

Je vous détaille maintenant les 3 raisons principales...

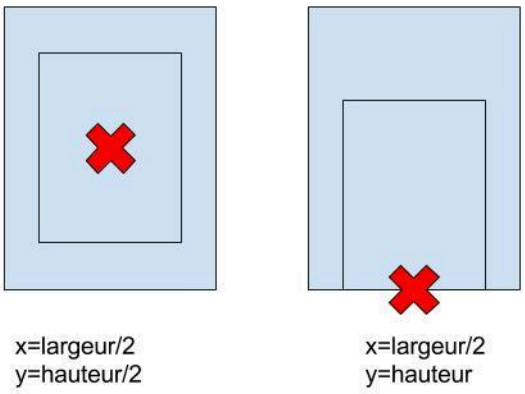
Raison 1 : déformer une image

La déformation (scaling) d'une image se fera toujours par rapport à son origine. Cela peut être vraiment contrariant par exemple si on souhaite animer cette déformation :



En se déformant, le coin reste en haut à gauche, ce qui peut ne pas être l'effet voulu.

Il est alors plus logique, de centrer l'origine (pour une planète par exemple), ou de la placer en bas au centre (pour un personnage).



La plupart des frameworks 2D permettent de déplacer l'origine; En Löve il faut utiliser une version de **draw** avec plus de paramètres, car l'origine est en 7e et 8e position (ici ox et oy) :

```
love.graphics.draw(imgPerso, x, y, rotation, 1, 1, ox, oy)
```

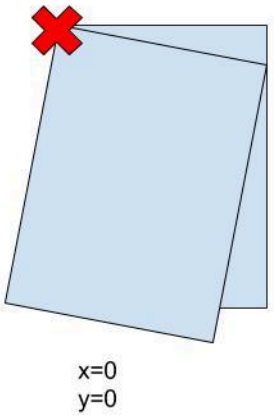
En 5e et 6e position on a la déformation (scaling). Il s'agit d'un facteur, donc 1 est la taille normale, 2 serait le double et 0.5 la moitié… On donne la déformation horizontale et la déformation verticale, donc 2 valeurs (ici 1,1 pour une taille normale). Si on donne une valeur négative, on obtient un effet miroir.

Si on veut déplacer l'origine en bas au centre ça va donc donner :

En origine x : la moitié de la largeur de l'image (33 dans notre exemple, mais mieux vaut le faire calculer par le programme) et 92 verticalement (ou 91 si on est tatillon).

Raison 2 : la rotation

Encore plus moche et contrariant : si vous appliquez une rotation à votre image, elle va "tourner" autour de son origine. C'est très rarement l'effet voulu…



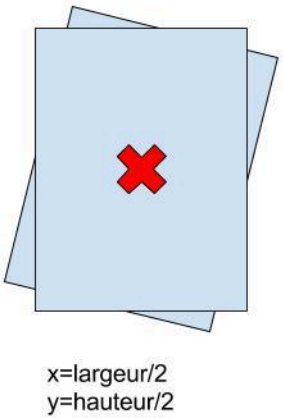
L'image tourne par rapport à son origine, ici 0,0

On va alors centrer l'origine sur l'image. Rien de plus simple :
Origine x = largeur de l'image divisée par 2
Origine y = largeur de l'image divisée par 2

En Löve :

```
ox = image:getWidth() / 2
oy = image:getHeight() / 2
```

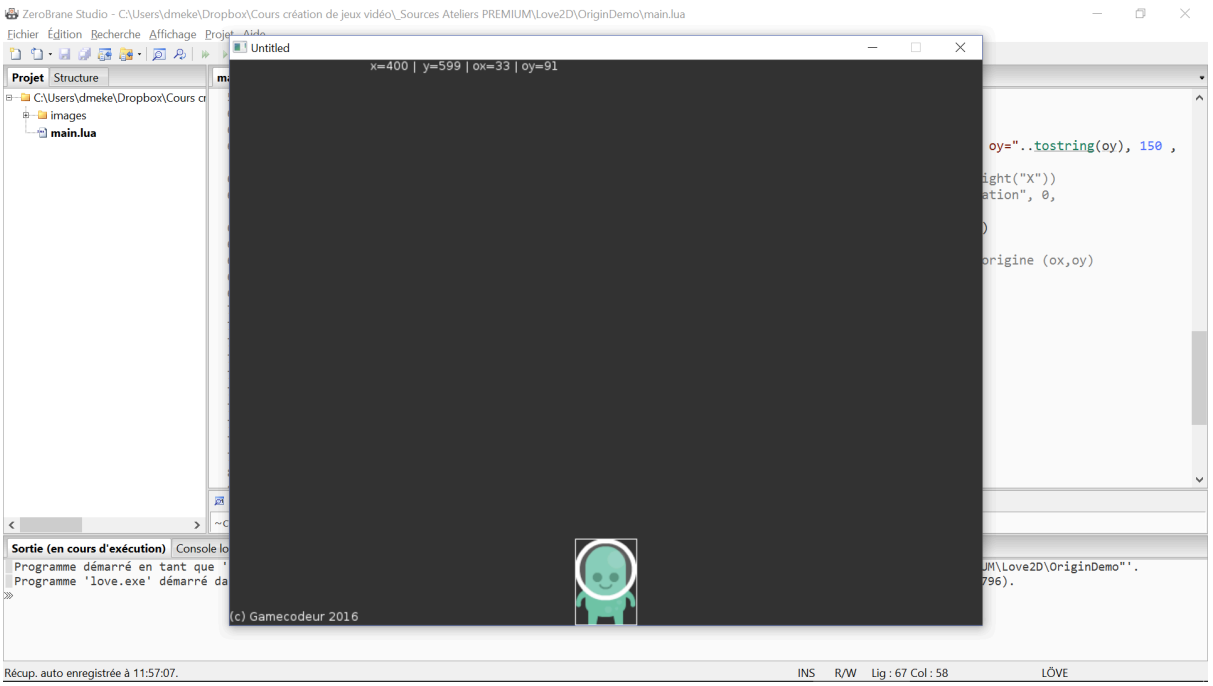
On utilise alors ces valeurs en 7e et 8e paramètres de draw (voir plus haut) !



Raison 3 : aligner / positionner une image par rapport à quelque chose

Si on souhaite afficher une image par rapport à sa base par exemple. Dans cette exemple j'ai placé l'origine en bas au centre, du coup il me suffit de dessiner l'image en bas de l'écran (sans calcul) pour qu'elle soit "posée" sur le bas de l'écran et non pas en dehors :

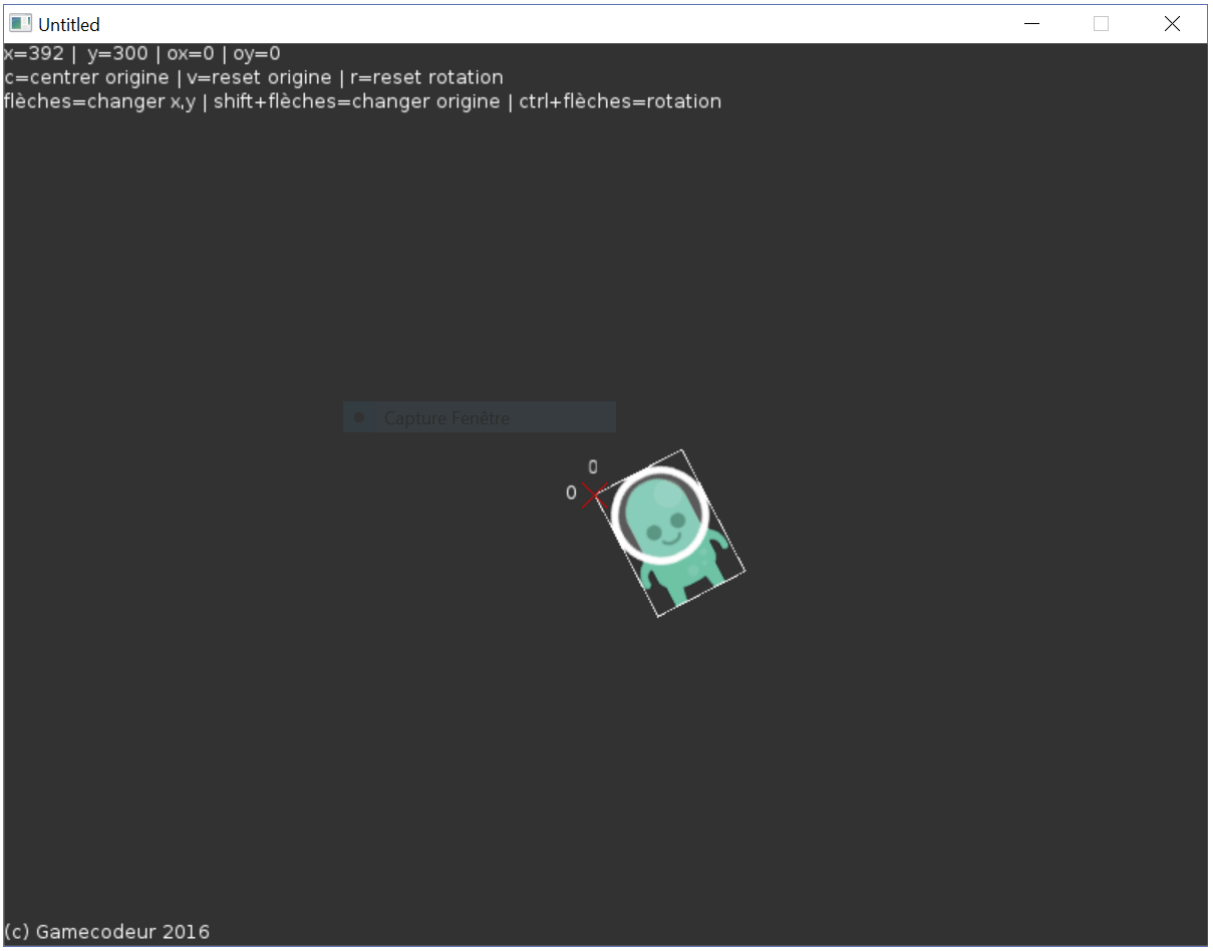
```
ox = image:getWidth() / 2
oy = image:getHeight()
```



Un projet de démo pour expérimenter !

J'ai créé un projet complet pour vous permettre d'expérimenter et de "voir" le décalage d'origine en action !.

Il permet, avec le clavier, de déplacer l'image, de changer son origine, de la faire tourner (rotation), et il affiche les valeurs à l'écran.



Utilisation :

- Déplacer l'image (changer ses coordonnées x et y) avec les flèches du clavier
- Maintenez SHIFT enfoncé + les flèches pour décaler l'origine
- Maintenez CTRL enfoncé + les flèches droite/gauche pour appliquer une rotation
- Pressez la touche "c" pour placer l'origine sur le centre de l'image
- Pressez la touche "v" pour replacer l'origine à 0,0 (valeur par défaut de Löve)
- Pressez la touche "r" pour annuler la rotation

Le lien du projet (sources Lua/Love2D) est sur la page de l'atelier, voir :
http://www.gamecodeur.fr/atelier-love2d-pixels-origine/