GAMECODEUR
École de jeux vidéo en ligne

ATELIER

Jeu de Pong :
Apprenez à coder votre premier jeu vidéo rapidement et facilement

Technos :
Langage Lua + moteur 2D Love2D

Lien de l'atelier :

Sommaire

Objectif

Dans cet atelier, je vais vous faire vivre une grande aventure.

Vous allez utiliser les connaissances en programmation que vous avez acquises pendant l'atelier "**La méthode complète pour apprendre les bases de la programmation même quand on n'a jamais programmé**".

Avec ces connaissances vous allez découvrir comment donner vie à un jeu vidéo.

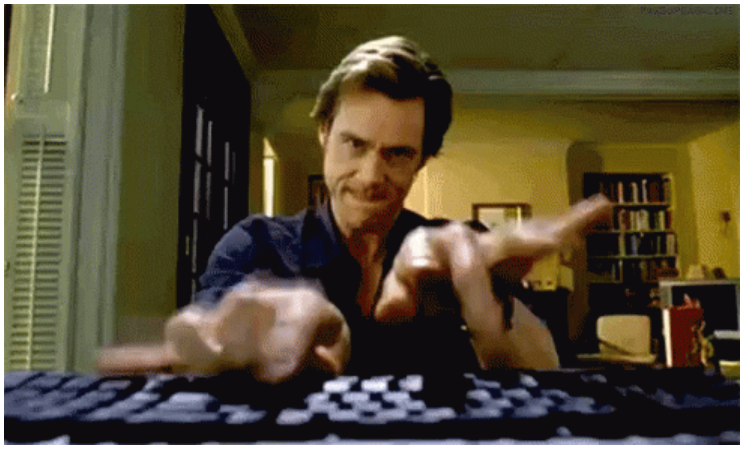
Afficher deux raquettes… une balle… et voilà un jeu de pong qui prend vie avec vos lignes de code.

Suivez-moi, on va faire ça, pas à pas, et je veux que vous compreniez chacune des étapes.

Si vous allez au bout de cet atelier, que vous y prenez du plaisir, et que vous êtes fièr(e) de vous malgré la simplicité du gameplay auquel vous aurez donné vie, alors c'est peut être que vous êtes fait(e) pour programmer des jeux vidéo.

Je suis tellement fier de vous...

Comment suivre cet atelier ?



Je code comme un ouf !

Lancez votre environnement de développement (voir l'atelier "Les bases") et suivez ce support de cours étape par étape.

Regardez les vidéos quand le support de cours y fait référence.

Faite une pause quand vous vous sentez prêt(e) à taper du code et lancez-vous. Revenez en arrière autant de fois que nécessaire.

Vous n'avez rien à apprendre par coeur à ce stade. On n'est pas à l'école !

Il est normal quand on débute de se référer à un support de cours. Vous apprendrez par coeur les choses sans vous en rendre compte, uniquement parce que vous en avez souvent besoin et que vous savez pourquoi vous en avez besoin.

Devenir programmeur prend du temps. Soyez patient(e) et savourez chaque petite victoire.

Savourez chaque chose que vous parvenez à afficher à l'écran.

Savourez chaque fois que vous donnez vie à quelque chose.

Savourez chaque fois que vous comprenez la signification d'une ligne de code que vous venez de taper.

Savourez chaque fois qu'une erreur se produit et que vous réussissez à la corriger.

Voyez les choses simplement et contentez-vous de progresser lentement mais sûrement.

Une fois l'atelier terminé, essayez de le refaire mais cette fois sans les vidéos, uniquement en vous servant du support de cours. Jetez alors un oeil à une vidéo seulement quand vous aurez longuement buté sur une étape.

Créer un nouveau projet

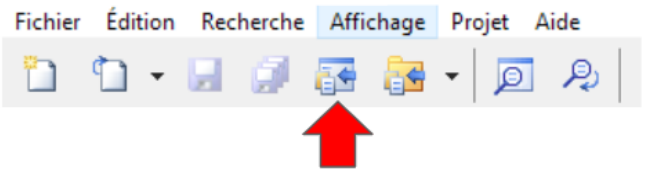
▶️ Une vidéo est disponible sur la page de l'atelier

C'est simple :

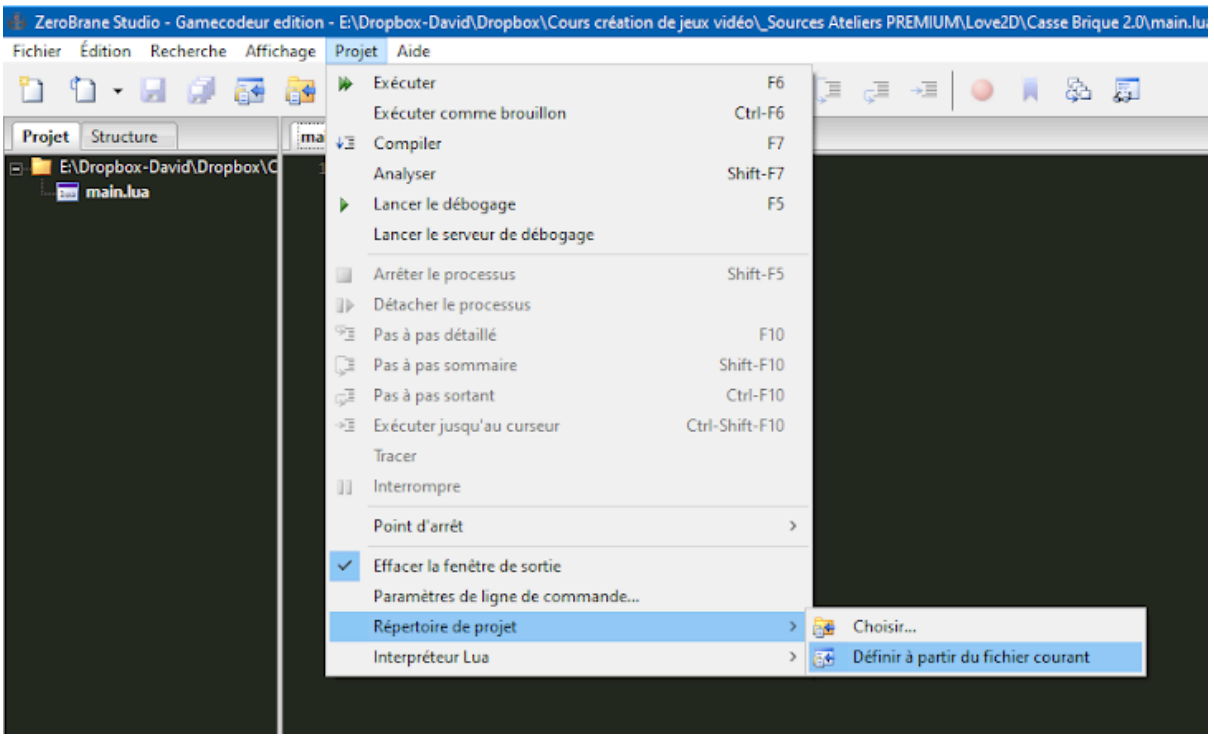
- Créez un dossier (répertoire) vide sur votre disque dur (évitez le répertoire "téléchargements").

- Créez un fichier "main.lua" à l'aide de ZeroBrane Studio et sauvegardez-le dans le répertoire que vous avez préalablement créé

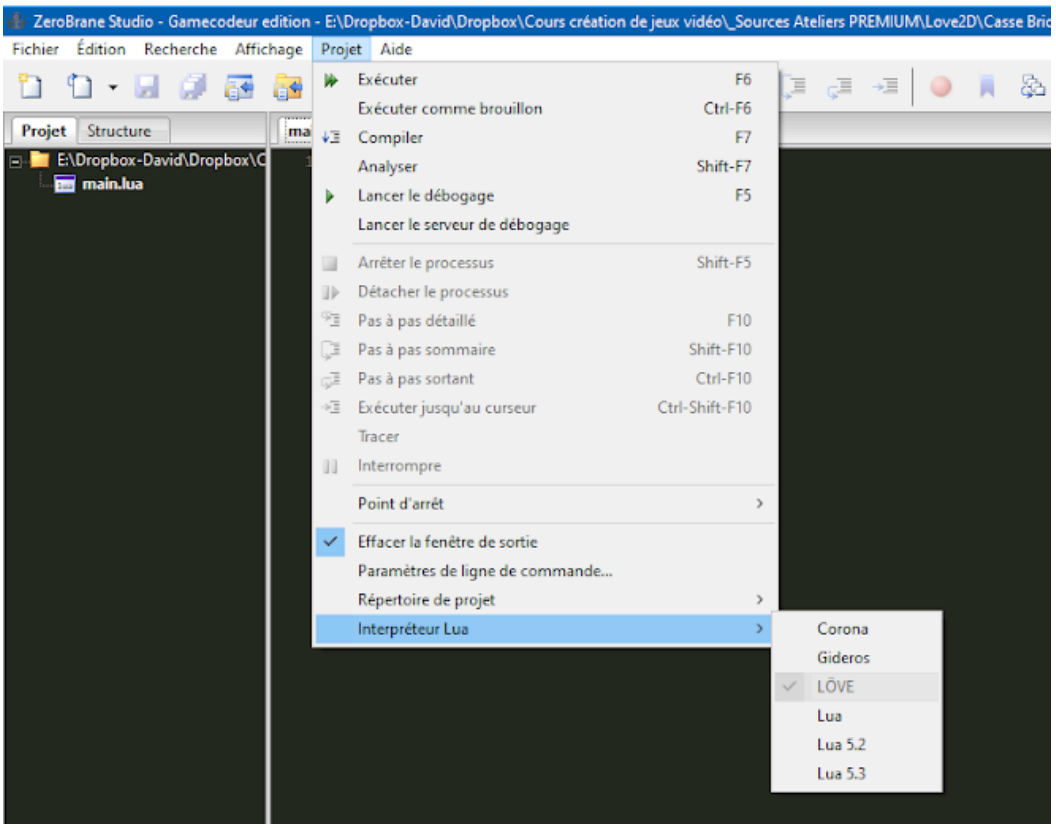
- Cliquez sur l'icône :



- Vous avez maintenant sélectionné le répertoire de travail
(vous pouvez aussi passer par le menu "Projet / Répertoire de projet / Définir à partir du fichier courant)



Et n'oubliez pas de sélectionner Love comme interpréteur Lua :



Vous pouvez programmer votre 1er jeu vidéo maintenant !

Love2D, comment ça marche

Nous rentrerons très en détail dans le fonctionnement de Love2D dans un prochain atelier, mais nous allons voir ici les bases de son fonctionnement.

Love2D a besoin de 3 fonctions "internes" que vous allez coder pour lui et qu'il exécutera quand il en aura besoin.

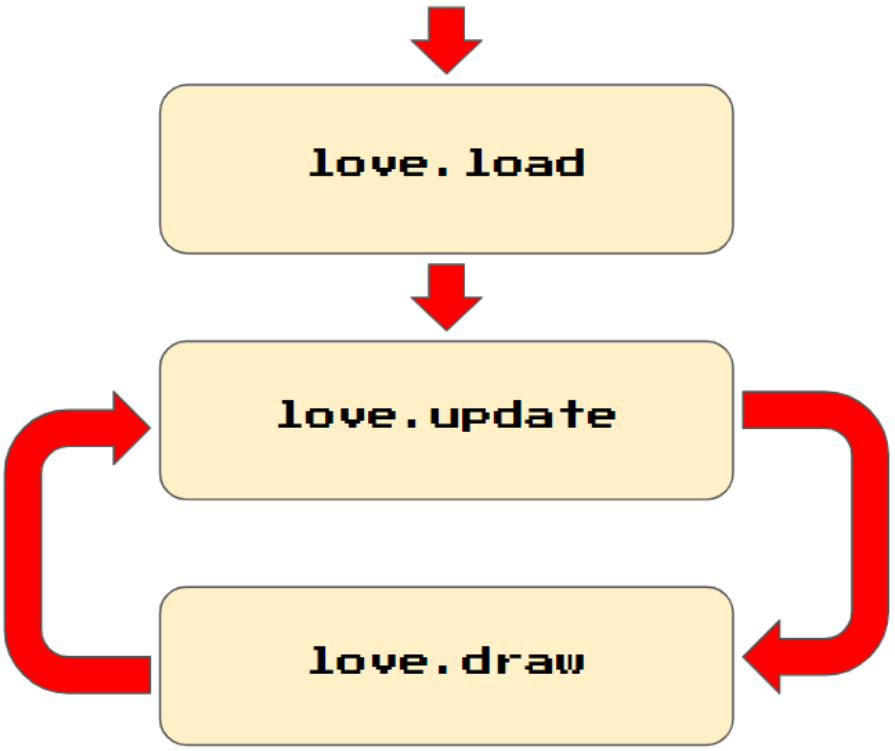
Chaque programme Love2D ressemblera à ça au minimum :

```
function love.load()
end

function love.update(dt)
end

function love.draw()
end
```

Voici comment elle sont exécutées par votre programme :



1) ***love.load*** est exécutée 1 seule fois au lancement de votre jeu

Vous utiliserez cette fonction pour y mettre les commandes d'initialisations de votre jeu comme par exemple le chargement des images, ou le changement de taille de votre fenêtre, etc.

2) ***love.update*** est exécutée 1 fois par rafraîchissement de l'écran (60 fois par seconde sur un écran 60 hertz, 100 fois par seconde sur un écran 100 hertz).

   Vous utiliserez cette fonction pour y mettre les commandes qui donnent vie à votre jeu (changement de variables en fonction de l'appui sur une touche, mouvements, animations, calcul de score, etc.)

3) ***love.draw*** est exécutée, à la suite de love.update, 1 fois par rafraîchissement de l'écran (60 fois par seconde sur un écran 60 hertz, 100 fois par seconde sur un écran 100 hertz).

   Vous utiliserez cette fonction pour y mettre les commandes qui "dessinent" votre jeu. Il est important que vous compreniez que 60 fois par seconde, vous redessinez TOUT l'écran. C'est comme ça dans tous les jeux vidéo. Et même l'écran que vous regardez actuellement est effacé et redessiné en permanence...

4) Ensuite Love va successivement exécuter love.update puis love.draw, indéfiniment jusqu'à que vous quittiez le jeu.

Récapitulatif :

1) Love2D vous invite à séparer la logique de votre jeu et son affichage. Tout ce qui concerne la logique (mouvement, calculs pour le gameplay, interaction avec le joueur, etc.) doit se trouver dans la fonction :

```
love.update(dt)
```

Le paramètre "dt" est le "delta time". La variable dt contiendra le temps qui s'est écoulé depuis le dernier appel de "update". Soit 0.166 environ pour 1/60e de seconde. Inutile de vous en soucier pour l'instant, on a déjà à faire !

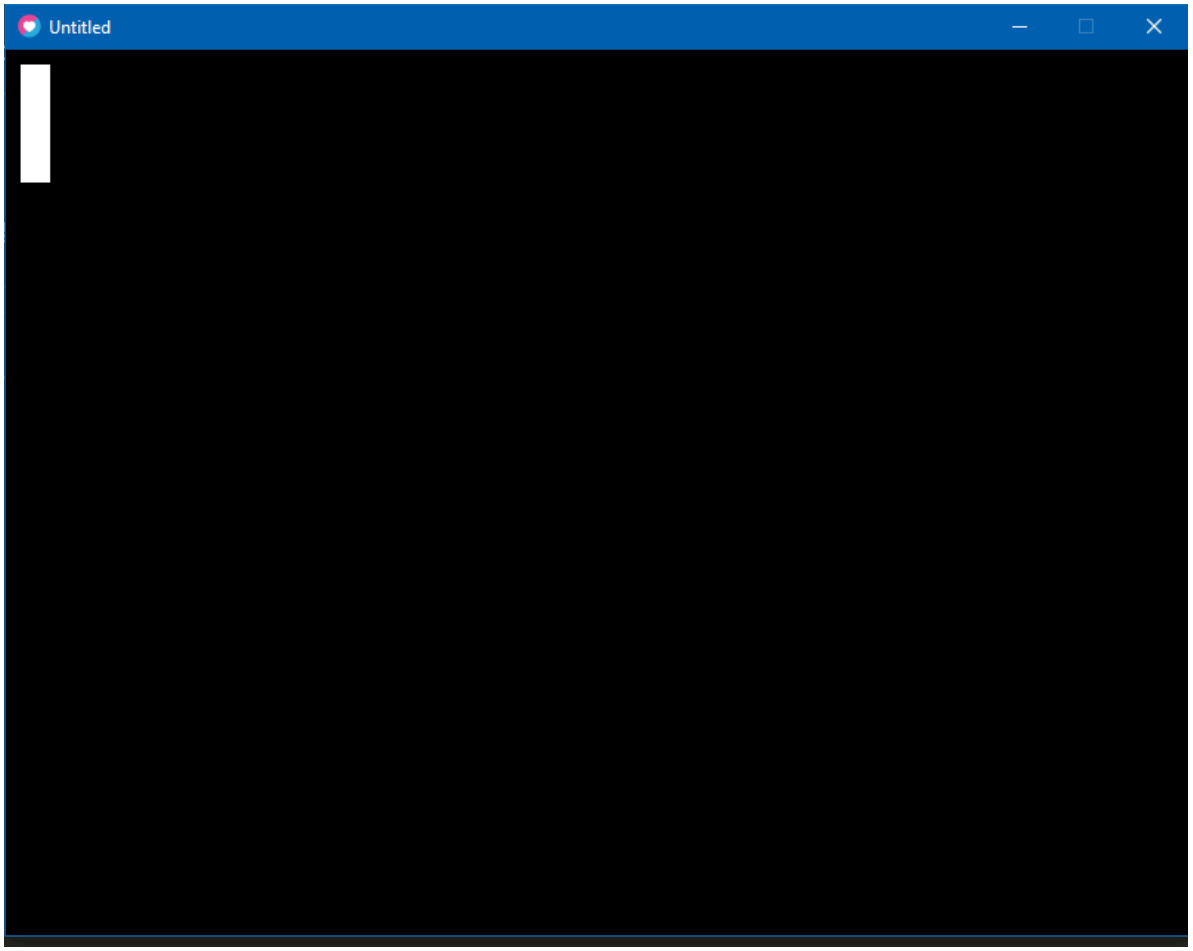
2) Love2D vous demande de redessiner l'écran environ 60 fois par secondes dans la fonction :

```
love.draw()
```

Vous en savez assez pour commencer. C'est encore flou ? C'est pas grave, relisez 2 ou 3 fois, buvez un verre d'eau, puis avancez. Les choses vont devenir plus claires pour vous en les mettant en pratique.

▶️ Une vidéo est disponible sur la page de l'atelier

Afficher quelque chose pour la première fois



Dans quelques minutes vous aurez réussi cet exploit… Afficher un rectangle !

Nous allons dessiner un rectangle comme sur cette image (ça vous fait déjà penser à quelque chose ?), il faut pour cela ajouter du code dans la fonction ***love.draw***.

Note : On peut aussi utiliser une image, mais pour débuter, il est beaucoup plus simple d'utiliser ce qu'on appelle des "primitives graphiques" : rectangle, lignes, cercle.

Si vous ne le savez pas encore, votre écran à une certaine taille en pixels (un pixel = un point à l'écran).

Par défaut, sous Love2D, votre jeu aura une taille de 800 pixels en largeur et de 600 pixels en hauteur (vous pourrez modifier cette taille par programmation plus tard)

Lorsque l'on souhaite afficher quelque chose sur l'écran de son jeu, on doit lui donner une position (on parle aussi de "coordonnées").

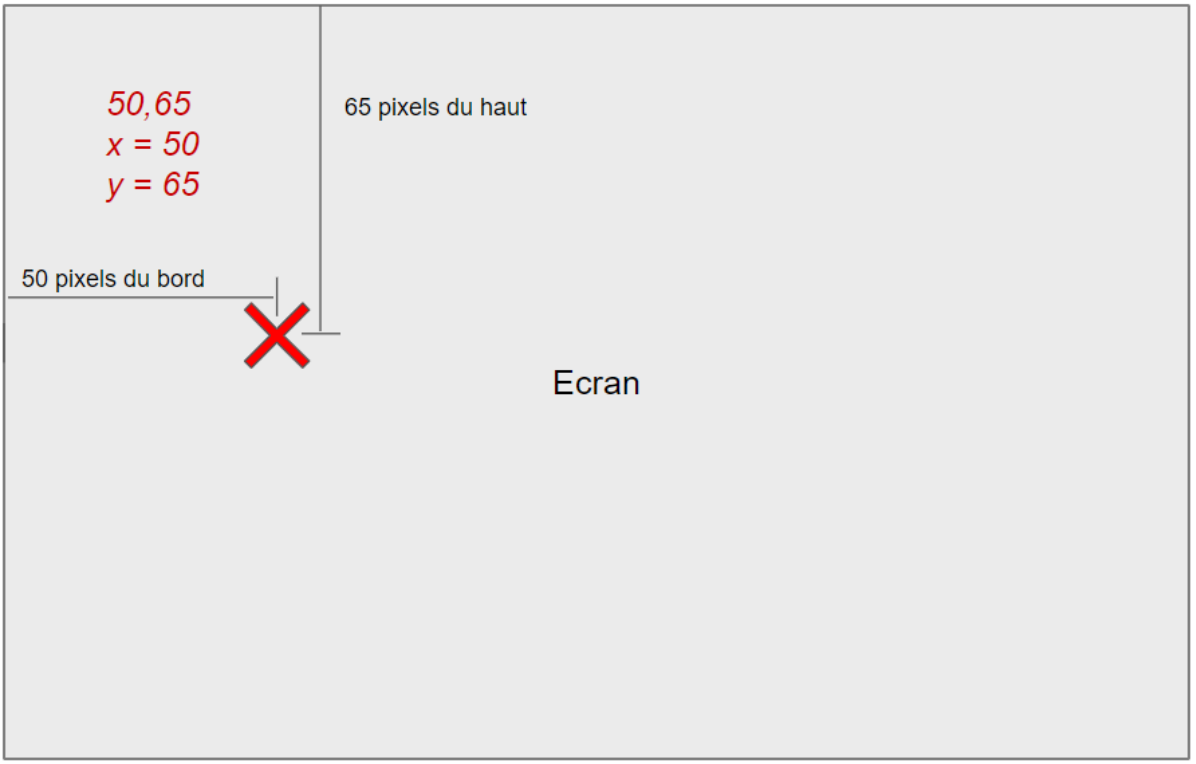
La position est exprimée en pixels à l'aide de 2 valeurs : la position horizontale et la position verticale. C'est souvent dans cet ordre : horizontal, vertical.

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.

Quand on parle de position horizontale, on parle de x (lettre X).
Quand on parle de position verticale, on parle de y (lettre Y).

Voilà un schéma pour que vous visualisiez un peu le principe :
(Le centre de la croix rouge vous montre une hypothétique coordonnée 50,65)



Voyons comment afficher un rectangle.

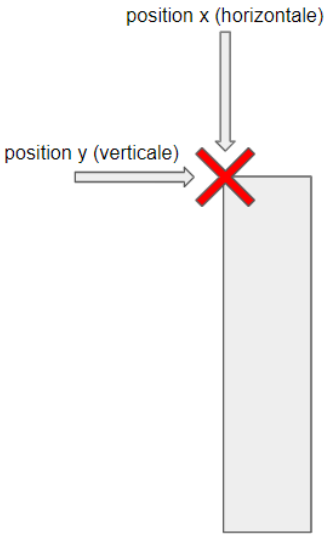
Facile ! Love2D propose une fonction *love.graphics.rectangle* pour cela.

```
love.graphics.rectangle("fill", 10, 10, 80, 20)
```

Lisez la documentation de la fonction ici : https://love2d.org/wiki/love.graphics.rectangle.

Cette ligne de code, ajoutée à "draw", va dessiner un rectangle plein ("fill") à la position x=10, y=10 et de taille 80 pixels en largeur et 20 pixels en hauteur. C'est notre future raquette !

Et là je suis en train de vous apprendre quelque chose d'important : la position d'affichage d'un rectangle (et plus tard, d'une image) est son coin supérieur gauche !



A ce stade, ce qui va beaucoup vous aider, c'est de "jouer" avec cette première ligne de code.

Changez les paramètres de la fonction love.graphics.rectangle et observez le résultat.

Changez les paramètres un par un, et non pas tous d'un coup, sinon vous ne comprendrez pas ce que vous avez changé. Le paramètre "fill" peut être remplacé par "line", testez ça aussi.

Exercices :

- Dessinez un rectangle horizontal (comme dans un casse brique) et non pas vertical
  (indice : les 2 derniers paramètres représentent la largeur et la hauteur)
- Dessinez un carré
  (indice : pour dessiner un carré il suffit d'avoir une largeur et une hauteur identique)
- Dessinez 2 rectangles côte à côte
  (indice : vous devez ajoutez une ligne de code supplémentaire)

A vous !

▶️ Une vidéo est disponible sur la page de l'atelier

Donner la vie

Nous avons réussi à afficher des primitives graphiques à une position donnée à l'écran.

Mais notre raquette ne va pas rester toute sa vie au même endroit. Alors comment lui donner vie ?

Regardez ce qui se passe si je remplace le paramètre y de la fonction love.graphics.rectangle, par une variable ?

Créez une variable "position_y" au début de votre code, avec la valeur 10 :

```
position_y = 10
```

et utilisez la à la place du paramètre y de love.graphics.rectangle :

```
love.graphics.rectangle("fill", 10, position_y, 20, 80)
```

Résultat : il ne se passe rien. Mais au moins on a conservé notre comportement original :).

Mais si je modifie cette variable lors du rafraîchissement de l'écran ?

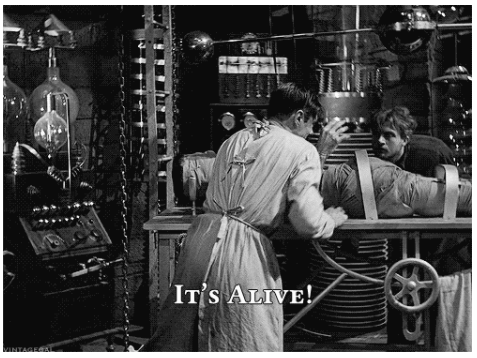
Ajoutez cette ligne de code dans love.update :

```
position_y = position_y + 1
```

Magie ! Vous avez donné vie à ce rectangle. C'est là tout le principe de la programmation de jeux vidéo. Le reste ce n'est que du code qui sort de votre tête.

Imaginez si cette variable était modifiée quand on appuie sur une touche du clavier ! Vous l'avez compris, vous donnez alors au joueur la possibilité de contrôler la position du rectangle. C'est ça un jeu vidéo !

▶️ Une vidéo est disponible sur la page de l'atelier



Il n'y a aucune magie

Ce que je vais vous dire ici est fondateur pour toute la suite de votre apprentissage. Vous devez comprendre cela pour avancer. Ecoutez bien (ou lisez plutôt…)

Ce que vous voyez à l'écran c'est vous qui l'affichez.

Love2D et le langage Lua ne font rien d'autre que ce que vous leur demandez de faire. Il n'y a aucune fonction "PONG", "Raquette" ou "Balle" dans Lua ou Love2D.

Lua ne sert qu'à exécuter des calculs sur des variables, répondre à des conditions et faire des boucles. Et en fonction de ces valeurs, vous affichez des choses à des coordonnées.

Et oui : Love2D ne sert qu'à afficher des images ou des formes primitives, jouer des sons et recevoir les commandes du clavier, de la souris ou du joypad. Rien d'autre.

Ce que contiennent vos variables c'est vous qui le décidez.

Nous avons appelé notre variable ***position_y*** mais n'oubliez pas que c'est vous qui décidez du nom des variables. On aurait pu l'appeler "concombre" et notre jeu fonctionnerait de la même manière ! On ne leur donne un nom clair que pour se rappeler de ce qu'on a prévu d'en faire. Donnez toujours un nom clair à vos variables mais n'imaginez pas que ça les rend intelligentes....

Ce que vous faites avec vos variables, c'est vous qui le décidez.

Ce n'est pas parce qu'on a appelé une variable ***position_y*** que notre rectangle se déplace. C'est seulement parce que nous avons utilisé cette variable dans l'appel de la fonction love.graphics.rectangle. Ce qui fait que 60 fois par seconde, notre rectangle est dessiné avec une valeur de position_y qui a changé.

Tout ce qui se passe sur votre écran c'est vous qui l'avez programmé.

Alors non, désolé, même si j'ai utilisé l'expression "magique !", il n'y a aucune magie.

La raquette : Votre premier élément de gameplay

Rendons notre code plus propre en utilisant les variables complexes (tables de Lua) et créons une "raquette" (un pad en anglais, et dans le vocabulaire de pong).

Ajoutez ce code au début de votre main.lua !

```
pad = {}
pad.x = 0
pad.y = 0
pad.largeur = 20
pad.hauteur = 80
```

Et modifiez la fonction love.draw :

```
-- Dessin du pad
love.graphics.rectangle("fill", pad.x, pad.y, pad.largeur, pad.hauteur)
```

Voilà !

Maintenant nous allons la faire se déplacer au clavier.

Dans la fonction love.update :

```
if love.keyboard.isDown("down") then
  pad.y = pad.y + 2
end
if love.keyboard.isDown("up") then
  pad.y = pad.y - 2
end
```

Vous le voyez, la fonction "love.keyboard.isDown" permet de savoir si une touche est enfoncé. Elle reçoit un seul paramètre, le nom de la touche sous forme de chaîne de caractères. Si vous en voulez toute la liste c'est ici : https://love2d.org/wiki/KeyConstant.

Note : Ce jeu sera plus rapide sur un PC avec un moniteur à 100 Hz que sur un PC avec un moniteur à 60 Hz. Nous apprendrons plus tard à utiliser le "deltatime" pour avoir une vitesse de déplacement homogène sur tous les PC. Mais à ce stade c'est inutile, cela serait vous embrouiller et vous ne verrez pas la différence !

Exercice

Comme l'indique le titre imaginé avec génie, voici un exercice.

Vous constaterez que dans votre jeu, la raquette peut "sortir" de l'écran si on va trop haut ou trop bas.

Imaginez une solution pour qu'elle ne puisse plus le faire.

Indices :

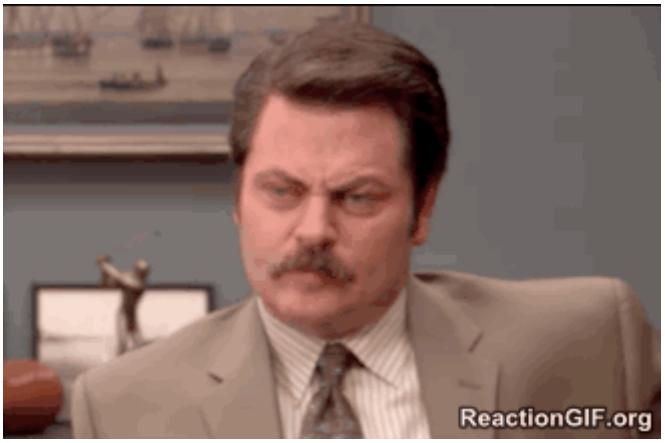
1. Ca se passe dans love.update

2. Ajoutez une condition aux lignes "if love.keyboard.isDown…" pour éviter que le déplacement ne soit effectué si les limites sont atteintes

3. La hauteur de l'écran s'obtient par la fonction "*love.graphics.getHeight()*"

4. N'oubliez pas que la position y de votre pad est sa position supérieure et non pas son centre

Vous ne comprenez rien à ce que je vous demande ?



Tant pis ! Lancez-vous.

Essayez de ne pas tricher, quitte à faire n'importe quoi… Rien ne va exploser. Personne ne regarde votre code...

Faites un dessin, bidouillez… vous n'y arriverez peut être pas mais vous exercerez votre cerveau. Vous allez apprendre des choses.

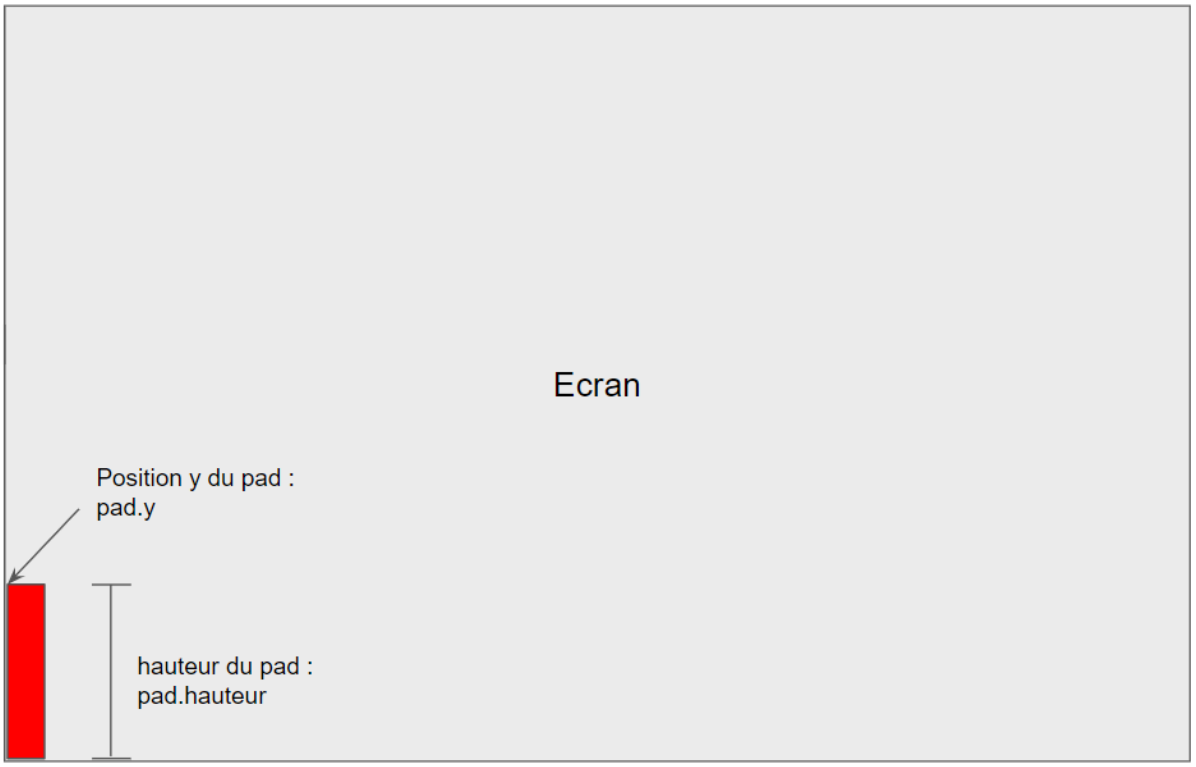
La solution est sur la page suivante.

Solution de l'exercice

Dans love.update :

```
if love.keyboard.isDown("down") and
   pad.y < love.graphics.getHeight() - pad.hauteur then
  pad.y = pad.y + 2
end
if love.keyboard.isDown("up") and pad.y > 0 then
  pad.y = pad.y - 2
end
```

Je vérifie tout simplement que le pad est à une position qui lui permet de se déplacer :



- Si il veut descendre, il doit avoir une position y inférieure au bas de l'écran "moins" sa hauteur. Essayez sans sa hauteur pour voir ce qui se passe.

- Si il veut monter, il doit avoir une position y supérieure à 0.

Note : j'ai mis la 1ère condition sur 2 lignes par coquetterie de présentation dans ce document, mais dans votre code vous pouvez tout mettre sur une seule ligne.

▶️ Une vidéo est disponible sur la page de l'atelier

Ajouter une balle et lui donner une impulsion

Faisons la même chose pour notre balle, juste après notre raquette.

```
balle = {}
balle.x = 400
balle.y = 300
balle.largeur = 20
balle.hauteur = 20
```

Et n'oubliez pas de la dessiner dans **love.draw** :

```
-- Dessin de la balle
love.graphics.rectangle("fill", balle.x, balle.y, balle.largeur,
balle.hauteur)
```

Comme vous le voyez, j'ai arbitrairement centrée la balle sur l'écran. Mais ce n'est pas assez précis, elle n'est pas bien centrée.

Pour la centrer réellement, ajoutons ceci à **love.load** :

```
  balle.x = love.graphics.getWidth() / 2
  balle.x = balle.x - balle.largeur / 2

  balle.y = love.graphics.getHeight() / 2
  balle.y = balle.y - balle.hauteur / 2
```

J'ai volontairement détaillé chaque opération pour ne pas tout mettre sur une seule ligne et vous embrouiller.

Pourquoi dans **love.load** ? Car c'est le moment où on est sûr que Love2D est initialisé et que la taille de l'écran est accessible.

J'explique plus en détail le calcul dans la vidéo correspondant à cette étape.

Ajoutez, pour le fun, ces 2 lignes à love.update :

```
  balle.x = balle.x + 2
  balle.y = balle.y + 2
```

Que se passe t'il ?

La balle se déplace en diagonale !

C'est pas mal mais nous n'avons aucun contrôle sur la balle si on fait comme cela. Ajoutons donc 2 variables pour le déplacement vertical et horizontal.

Que diriez-vous de les appeler vitesse_x, et vitesse_y ?

```
balle.vitesse_x = 2
balle.vitesse_y = 2
```

Et améliorez le code que vous avez mis dans love.update ainsi :

```
  balle.x = balle.x + balle.vitesse_x
  balle.y = balle.y + balle.vitesse_y
```

Maintenant nous avons un peu plus le contrôle.

Vérifiez-le ainsi :

Changez une des 2 vitesses en la rendant négative (ajoutez le signe moins devant).

Exemple :

```
balle.vitesse_x = -2
balle.vitesse_y = 2
```

Que se passe t'il ?

Observez le résultat de vos modifications et essayer de faire aller la balle dans les 4 diagonales. Quelles sont les valeurs que vous avez utilisé ?

Voilà, vous commencez à comprendre.

▶️ Une vidéo est disponible sur la page de l'atelier

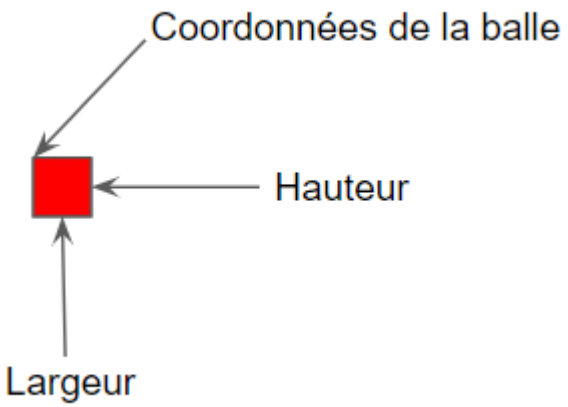
La balle se déplace… Je comprends !

Faire rebondir la balle sur les murs

Quand on calcule si un de nos objets touche quelque chose (collision), le plus important est de comprendre ce que représentent ses coordonnées.

La clé est de retenir que :

Les coordonnées représentent le coin supérieur gauche de notre objet !



Alors a partir de quand considère-t-on que la balle touche le bord de l'écran ?

Pour le haut et la gauche c'est facile. C'est quand x < 0 ou y < 0.

Pour le bas et la droite c'est plus compliqué, il faut prendre en compte la taille de l'écran et la taille de la balle.

Comment inverser la direction de la balle ?

En inversant la vitesse.

Ce sont les questions que vous devez vous poser pour la faire rebondir sur les bords de l'écran.

Grâce à la magie du "je sais programmer", vous savez déjà le faire parce-que :

- Vous avez déjà détecté si la raquette atteignait le bas de l'écran, c'est EXACTEMENT la même chose
- Vous savez faire aller la balle dans les 4 diagonales… Et vous avez compris qu'il suffisait d'inverser vitesse_x ou vitesse_y pour inverser le trajet de la balle !

Voici le code pour gérer les rebonds dans les 4 directions :

```
if balle.x < 0 then
  balle.vitesse_x = balle.vitesse_x * -1
  -- ou : balle.vitesse_x = -balle.vitesse_x
end
if balle.y < 0 then
  balle.vitesse_y = balle.vitesse_y * -1
end
if balle.x > love.graphics.getWidth() - balle.largeur then
  balle.vitesse_x = balle.vitesse_x * -1
end
if balle.y > love.graphics.getHeight() - balle.hauteur then
  balle.vitesse_y = balle.vitesse_y * -1
end
```

Pour inverser la vitesse, je vous donne les 2 formules possibles :

- Multiplier par -1 (balle.vitesse_x = balle.vitesse_x * -1)
- Assigner la vitesse négative (balle.vitesse_x = -balle.vitesse_x)

Utilisez la formule qui vous donne le moins mal à la tête.

- Fois moins un ??

▶️ Une vidéo est disponible sur la page de l'atelier

Faire rebondir la balle sur la raquette

Pour faire rebondir la balle sur la raquette, on va faire de calculs simples.

Inutile d'être bon en math !

Vous avez juste besoin :

1) De réfléchir
2) De savoir faire des additions et des soustractions

Prenez un papier, un crayon et une gomme.

Dessinez votre écran et votre raquette.

Maintenant demandez-vous comment savoir :

1) Si la balle a atteint la raquette
2) Si la balle est sur la raquette ou en dehors

Répondez individuellement à ces 2 questions en vous creusant la tête.

Il y a plusieurs solutions. Laquelle est la bonne ? Celle qui marche et que vous comprenez !

Maintenant, voici MA solution :

Comment savoir si la balle a atteint la raquette ?

Tout simplement en observant sa position x (position_x) et le bord droit de la raquette.

Où se trouve le bord droit de la raquette ?

Il suffit d'ajouter la largeur de la raquette à sa position x.

Donc si la position x de la balle devient inférieure ou égale à la position du bord droit de la raquette, c'est qu'elle est potentiellement en contact avec elle (en tout cas qu'il faut qu'on regarde si elle l'est) :

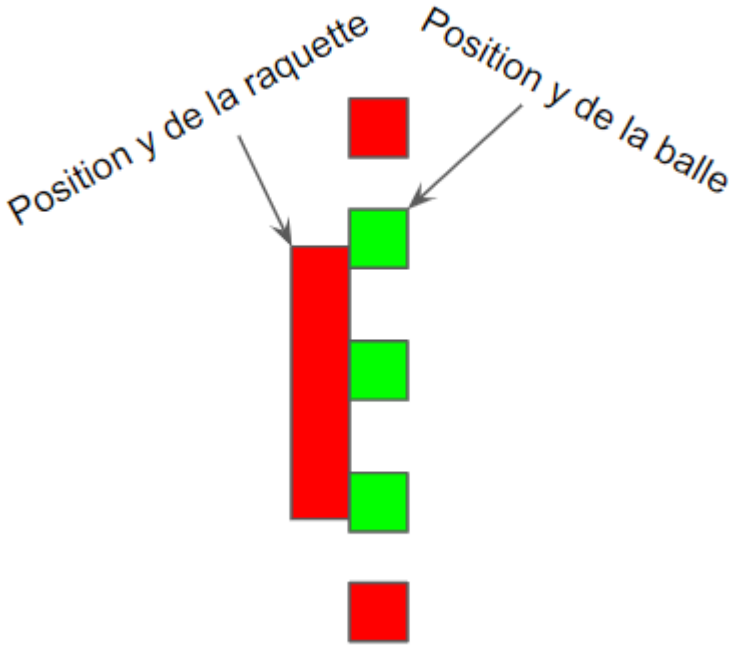
Dans love.update :

```
-- La balle a-t-elle atteint la raquette ?
if balle.x <= pad.x + pad.largeur then
  -- Tester maintenant si la balle est sur la raquette ou pas
  -- ...
end
```

Maintenant on a la deuxième énigme :

Comment savoir si la balle est en contact avec la raquette ?

Maintenant, regardez votre dessin, c'est avec la position verticale (position_y) qu'on va jouer.



Sur ce magnifique schéma, les balles colorées en vert sont en contact avec la raquette.

On constate que :

- Si la position y du bord du bas de la balle (position y de la balle + sa hauteur) est supérieure la position y de la raquette, alors on a au moins une partie de la balle qui est plus basse que le haut de la raquette
- Si en plus de cela la position y de la balle est inférieure au bord bas de la raquette (la position y de la raquette + sa hauteur) alors on a bien la balle en contact !

C'est plus long à écrire qu'à comprendre.

Gardez en tête que vous allez passer votre vie à faire ce genre de calculs simples en additionnant des hauteurs ou des largeurs à des positions.

Je vais vous aider à vous en souvenir.

Suivez simplement ce conseil :

Faites-vous tatouer sur l'intérieur de la cuisse (la douleur rendra le souvenir indélébile) ceci :

- Si vous voulez le bord du haut : position y
- Si vous voulez le bord du bas : position y + hauteur
- Si vous voulez le bord de gauche : position x
- Si vous voulez le bord de droite : position x + largeur

Allez on avance. On a donc nos 2 conditions à vérifier.

Et dans le cas où les 2 conditions sont vérifiées, on inverse la vitesse x de la balle pour simuler un rebond.

BOOM !

En code ça donne :

```
-- La balle a-t-elle atteint la raquette ?
if balle.x <= pad.x + pad.largeur then
  -- Tester maintenant si la balle est sur la raquette ou pas
  if balle.y + balle.hauteur > pad.y and balle.y < pad.y + pad.hauteur
  then
    balle.vitesse_x = balle.vitesse_x * -1
  end
end
```

C'est tout ! Vous êtes en train de coder un jeu vidéo !

Incroyable... Je suis en train de coder un jeu…

▶️ Une vidéo est disponible sur la page de l'atelier

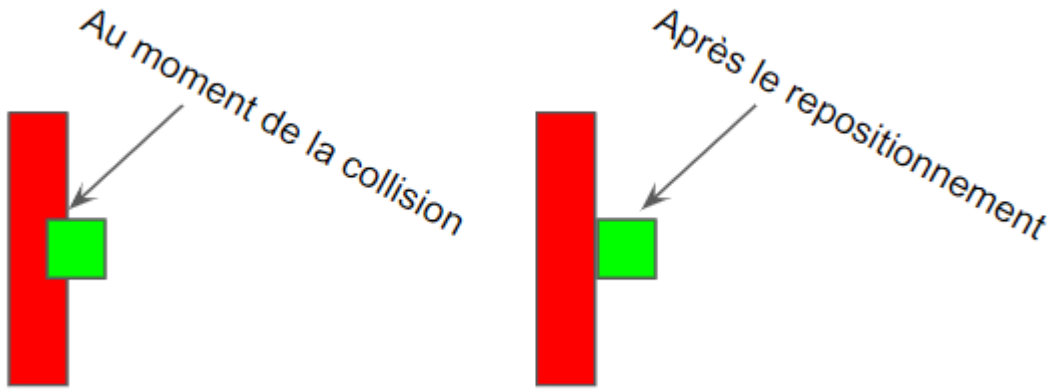
Petit cours de précision

Il est d'usage, quand une collision se produit, de replacer l'objet qui est en mouvement, à une position la plus proche de l'objet qu'il a touché.

Pour notre balle, il va falloir la repositionner au bord de la raquette.

C'est une bonne habitude à prendre et qui peut vous éviter d'avoir des bugs de collisions. En effet, pourquoi laisser la balle en position de collision, plantée dans la raquette ?

Si c'est un peu abstrait pour vous, normal à ce stade, mais voici une explication visuelle qui peut vous aider à comprendre :



A gauche vous avez la situation quand la balle est en situation de collision.

A droite, c'est la position que devrait avoir votre balle une fois qu'elle a été repositionnée par vos soins.

En code cela donne :

```
  -- La balle a-t-elle atteint la raquette ?
  if balle.x <= pad.x + pad.largeur then
    -- Tester maintenant si la balle est sur la raquette ou pas
    if balle.y + balle.hauteur > pad.y and balle.y < pad.y + pad.hauteur
then
      balle.vitesse_x = balle.vitesse_x * -1
      -- Positionne la balle au bord de la raquette
      balle.x = pad.x + pad.largeur
    end
  end
```

Détecter quand la balle sort du terrain

Vous pourriez faire cette partie seul j'en suis sûr.

Comment savoir que la balle a atteint le bord de l'écran à gauche ?

Reprenez votre dessin et cassez-vous la tête encore une fois.

Réfléchissez un peu !

C'est le plus simple du monde :

La balle sort si le bord gauche de la balle atteint le bord gauche de l'écran.

Le bord gauche de la balle c'est balle.x.

Le bord gauche de l'écran c'est… 0 !

```
-- La balle a-t-elle atteint le bord gauche de l'écran
if balle.x <= 0 then
  -- Perdu !
end
```

A vous de voir ce que vous faites quand la balle sort.

Perso je vais la remettre au milieu de l'écran.

Et si on apprenait à créer une fonction ?

Comme je l'ai déjà fait pour positionner la balle au début et qu'il n'est pas propre de dupliquer du code, je vais créer une fonction, que j'appellerai dans love.load à la place du code en question :

Je vous ai expliqué les fonctions dans le cours **La méthode complète pour apprendre les bases de la programmation même quand on n'a jamais programmé**. Voici une occasion d'utiliser ces connaissances.

Ca donne :

```
function CentreBalle()
  balle.x = love.graphics.getWidth() / 2
  balle.x = balle.x - balle.largeur / 2

  balle.y = love.graphics.getHeight() / 2
  balle.y = balle.y - balle.hauteur / 2
end

function love.load()

  CentreBalle()

end
```

Vous n'avez plus qu'à appeler votre fonction *CentreBalle* au moment où la balle sort de l'écran à gauche.

```
  -- La balle a-t-elle atteint le bord gauche de l'écran
  if balle.x <= 0 then
    -- Perdu !
    CentreBalle()
  end
```

Quand vous aurez un deuxième joueur, ça sera l'occasion de compter les points…

Mini exercice :

Profitez de votre fonction CentreBalle pour rétablir la vitesse x et y de la balle afin qu'elle reparte en diagonale vers le bas et vers la droite.

Solution Express :

Ajoutez ces 2 lignes à la fonction CentreBalle :

```
  balle.vitesse_x = 2
  balle.vitesse_y = 2
```

▶️ Une vidéo est disponible sur la page de l'atelier

Exercice : Ajoutez une deuxième raquette

Quand on parle du loup… A vous d'essayer !

Vous allez essayer de le faire seul, avec plein d'indices et une vidéo qui vous donne la solution…

Voici les indices :

- Il faut sûrement créer un deuxième pad…
- Vous ne pouvez pas appeler votre deuxième pad comme le premier
- Vous devrez, si vous voulez voir ce deuxième pad, l'afficher dans *love.draw*. Parce que sinon ça sert à rien. Pas de magie n'oubliez pas !
- Vous allez devoir tester 2 autres touches, par exemple "a" et "q" pour qu'un deuxième joueur puisse jouer en même temps que le premier, car ils pourront difficilement jouer avec les même touches
- Il me semble judicieux que le joueur de gauche utilise les touches a et q et que le joueur de droite utilise les touche haut et bas...
- Pour tester si la deuxième raquette touche les bords haut et bas de l'écran, c'est PAREIL que pour le premier pad
- Pour tester si la balle touche le 2ème pad, c'est le plus compliqué
- Vous allez devoir comparer le bord de droite de la balle avec le bord de gauche de la 2ème raquette
- N'oubliez pas de tester si la balle sort de l'écran à droite
- Pour obtenir la largeur de l'écran c'est *love.graphics.getWidth()*

Vous avez tout !

La solution, avec le comptage des points, est dans la vidéo correspondante de l'atelier.

▶️ **Une vidéo est disponible sur la page de l'atelier**

La balle est dans votre camp

PONG !? Je l'ai fait !

Grâce à cet atelier vous avez, je l'espère, découvert la magie qui se cache derrière la programmation d'un jeu vidéo.

Vous avez pu mettre en pratique vos premières compétences en programmation pour donner vie à une gameplay, modeste, mais mythique.

A partir de ces bases, la balle est dans votre camp.

Voici quelques pistes si vous voulez vous exercer avec cet atelier :

- Faire repartir la balle à l'opposé du joueur qui a gagné le point
- Centrer les raquettes verticalement au début de la partie
- Ajouter des sons
- Tracer une ligne centrale (avec love.graphics.line)
- Permettre de recommencer une partie
  Indice : Créer une fonction genre "NouvellePartie"
- Et toute autre idée, simple à mettre en place.

→ Vous avez un aperçu de ces évolutions dans le projet contenu dans le dossier "6 - Pong amélioré" fourni avec cet atelier.

Et maintenant ?!

J'ai ouvert la voie, je vous invite maintenant à me suivre pour continuer notre aventure qui ne fait que commencer.

Nous allons ensemble coder un casse brique, faire voler un vaisseau spatial, programmer les bases d'un RPG avec une map, programmer une intelligence artificielle pour lancer des zombis à nos trousses, coder des effets visuels, un shooter avec un scrolling, générer des donjons, des labyrinthes, apprendre d'autres langages de programmation, des moteurs plus puissants…

Vous n'aurez plus aucune limite.

C'est la méthode Gamecodeur.

Suivez le guide pour la suite de ce parcours débutant :
https://www.gamecodeur.fr/parcours-debutant/

Rendez-vous sur
https://www.gamecodeur.fr/acces-premium/
si vous n'avez pas encore adhéré à Gamecodeur.

Merci et bon code !

David de Gamecodeur