Introduction automatisée

Tout d'abord, j'ai créé une map à l'aide de Tiled pour un défilement horizontal de la droite vers la gauche. Je vous laisse faire la map de la taille que vous souhaitez (^_^)

Comme indiqué dans la vidéo de David "Intégrer la map dans le projet Love2D", j'ai créé les variables "map_intro" et "level_intro". Ces 2 variables contiennent le niveau d'intro et toutes ses informations. Mon fichier s'appelle "map_intro.lua".

```
local map_intro = require("map_intro")
local level_intro = map_intro.layers[1].data
```

On crée en premier 2 tables vides : "liste_sprites_intro" et "liste_tir_intro"

```
local liste_sprites_intro = {}
local liste_tirs_intro = {}
```

On déclare ensuite plusieurs varaibles dont nous servirons plus tard (dont "xScrolling" qui définit la position du pixel de la première colonne de la map)

```
local xScrolling = 0
local ind = 1
local indX = 0.0
local indY = 0.0
local timer = 0
```

On crée une nouvelle table nommée "liste_mvt" qui va contenir les différents mouvements à appliquer au vaisseau. Cette table est assez particulière. Elle contient une table qui, elle-même contient une table qui contient 2 informations. Pfiou... Vous trouvez ça compliqué ? En fait, ce n'est pas si compliqué que ça (^_^)
J'ai souhaité que ce programme puisse gérer plusieurs vaisseaux avec des comportements différents. C'est pour cela que la table "liste_mvt" peut contenir plusieurs tables : une table par vaisseau à diriger.
Cette table est elle-même une table qui contient 2 valeurs. La première valeur correspond au déplacement horizontal (en X). Si elle est positive, le vaisseau se déplace à droite de l'écran. Si elle est négative, le vaisseau se déplace à gauche de l'écran. La seconde valeur correspond au déplacement vertical (enY). Si elle est positive, le vaisseau se déplace vers le bas de l'écran. Si elle est négative, le vaisseau se déplace vers le haut de l'écran.

```
local liste_mvt = { -- une sous-table par vaisseau à déplacer
  { --              Gauche/Droite            Haut/Bas
    { 32,   0}, -- Droite de +320 pixels
    {-10,  10}, -- Gauche de -200 pixels    Bas  de 100 pixels
    { 10,   5}, -- Droite de +100 pixels    Bas  de  50 pixels
    { 10,   0}, -- Droite de +100 pixels
    { 10, -10}, -- Droite de +100 pixels    Haut de 100 pixels
    {-10,   0}, -- Gauche de -200 pixels
    { 15, -10}, -- Droite de +150 pixels    Haut de 100 pixels
```



```
      {  5, - 5}, -- Droite de + 50 pixels    Haut de  50 pixels
      {- 5,  10}  -- Gauche de -100 pixels    Bas  de 100 pixels
    },
    { --              Gauche/Droite           Haut/Bas
      { 32,   0}, -- Droite de +320 pixels
      {- 5,  10}, -- Gauche de -100 pixels    Bas  de 100 pixels
      {  5, - 5}, -- Droite de + 50 pixels    Haut de  50 pixels
      { 15, -10}, -- Droite de +150 pixels    Haut de 100 pixels
      {-10,   0}, -- Gauche de -200 pixels
      { 10, -10}, -- Droite de +100 pixels    Haut de 100 pixels
      { 10,   0}, -- Droite de +100 pixels
      { 10,   5}, -- Droite de +100 pixels    Bas  de  50 pixels
      {-10,  10}  -- Gauche de -200 pixels    Bas  de 100 pixels
    }
  }
```

Dans un premier temps, nous allons créer une fonction "InitIntro" qui va servir à initialiser les informations (par exemple, la position des vaisseaux au départ de l'intro).

```
function InitIntro()
  CreateSpriteIntro("ship2", -64, hauteur/2, true, 90, 1)
  CreateSpriteIntro("ship2", -84, hauteur/2 - 25, true, 90, 2)
  indX = 0
  indY = 0
end
```

Ensuite, nous allons dupliquer la fonction permettant de créer un sprite en lui ajoutant 3 nouveaux paramètres :
- pIsShip : pour définir s'il s'agit d'un vaisseau (valeurs possibles : vrai ou faux)
- pRotation : pour indiquer la rotation, en degré, de l'image
- pIndMvt : pour déterminer quelle liste choisir dans la table "liste_mvt"

Et nous allons enregistrer ce sprite dans la table "liste_sprite_intro".

```
function CreateSpriteIntro(pNomImage, pX, pY, pIsShip,
    pRotation, pIndMvt)
  sprite = {}
  sprite.x = pX
  sprite.y = pY
  sprite.isShip = pIsShip
  if (pRotation) then
    sprite.r = pRotation
  else
    sprite.r = 0
  end
  sprite.toDelete = false
  sprite.image =
    love.graphics.newImage("images/"..pNomImage..".png")
  sprite.l = sprite.image:getWidth()
  sprite.h = sprite.image:getHeight()
```



```
    sprite.frame = 1
    sprite.listeFrames = {}
    sprite.maxFrame = 1
    if (pIndMvt) then
      sprite.indMvt = pIndMvt
    else
      sprite.indMvt = 1
    end
    sprite.index = 1
    sprite.indX = 0
    sprite.indY = 0
    sprite.canFire = false
    sprite.timer = 0
    table.insert(liste_sprites_intro, sprite)
    return sprite
  end
```

Comme nous voulons voir nos vaisseaux tirer, on crée fonction de création du tir. Elle est identique à celle du jeu mais nous stockons le tir dans la table "liste_tirs_intro" pour différencier les tirs dans le jeu de ceux de l'introduction.

```
  function CreateShootIntro(pType, pNomImage, pX, pY, pVitesseX,
      pVitesseY)
    local tir = CreateSpriteIntro(pNomImage, pX, pY)
    tir.vx = pVitesseX
    tir.vy = pVitesseY
    tir.type = pType
    table.insert(liste_tirs_intro, tir)
    sonShoot:play()
  end
```

Nous voilà dans le coeur du système de l'introduction automatisée ! La fonction `ShowIntro` !
On peut la décomposer en plusieurs parties. la première partie est la déclaration des variables locales. On définit plusieurs variables qui nous serviront plus loin (pour nos futures boucles sur les tables par exemple). Ensuite, nous déclarons 2 variables pour récupérer la largeur et la hauteur de votre niveau d'intro.

```
  -- Déclaration des variables locales
  local n, ligne, colonne, x, y
  local nbLignes = map_intro.layers[1].height
  local nbcolonnes = map_intro.layers[1].width
```

On crée ensuite une variable "max". Elle va servir à connaître la taille, en pixels, de votre carte.
Tout d'abord, j'ai opté pour un scrolling (défilement) horizontal.
Donc, je prends le nombre de colonnes de ma map (ici, "nbcolonnes") et je multiplie cette valeur par la taille en pixels des tuiles utilisées :

```
  nbcolonnes * map_intro.tilesets[1].tilewidth
```

Pour finir, j'y ajoute la largeur de mon écran pour que l'affichage commence à droite de l'écran.

Je multiplie le tout par -1 pour indiquer la position, en pixels, où se trouvera la première colonne de tuiles dès que toutes les colonnes auront disparu de l'écran. Donc, quand toutes les tuiles se trouveront à gauche de l'écran.

```
local max = ((nbcolonnes * map_intro.tilesets[1].tilewidth) +
   largeur) * -1
```

On effectue un test pour savoir si le premier pixel de la première colonne du niveau (ici, xScrolling) est affiché suffisamment loin. Si c'est le cas, c'est que le niveau a été affiché au complet. Alors, on remet xScrolling à 0 pour afficher le niveau à droite de l'écran; sinon, on décale le niveau vers la gauche.

```
-- ici, on fait la boucle sur niveau
if xScrolling <= max then
  xScrolling = 0
else
  xScrolling = xScrolling - 1
end
```

Ensuite, on dessine le niveau :

```
x = largeur + xScrolling
y = 0
for ligne = 1, nbLignes do
  for colonne = 1, nbcolonnes do
    local tuile = level_intro[(ligne-1)*nbcolonnes + colonne]
    if tuile > 0 then
      love.graphics.draw(imgTuiles[tuile], x, y, 0, 1, 1)
    end
    x = x + 32
  end
  x = largeur + xScrolling
  y = y + 32
end
```

Pour ceux qui ont fait l'atelier sur Tiled, vous remarquerez qu'il n'y a que peu de différence entre la fonction "drawJeu" de David et celle-ci :

- l'initialisation des variables X et Y

```
x = largeur + xScrolling
y = 0
```

- la remise à zéro de la variable X et Y

```
x = largeur + xScrolling
y = y + 32
```

Pour les autres, il y a en plus l'initialisation de la variable "tuile" :

```
local tuile = level_intro[(ligne - 1) * nbcolonnes + colonne]
```

Ici, vous pouvez tester votre intro, enfin le défilement du niveau tout du moins.

Maintenant que l'on sait que notre niveau s'affiche correctement et défile de droite à gauche, on peut passer aux tirs. Ici, on procède exactement comme David l'a fait.

```
-- boucle sur les tirs
for n=#liste_tirs_intro, 1, -1 do
  local tir = liste_tirs_intro[n]
  tir.x = tir.x + tir.vx
  tir.y = tir.y + tir.vy
  if tir.x > largeur then
    -- ici, on efface le tir
    tir.toDelete = true
    table.remove(liste_tirs_intro, n)
  end
end
```

C'est ici qu'il va falloir être attentifs ! C'est la partie qui permet le déplacement des vaisseaux !

Tout d'abord, nous allons faire une boucle sur la liste des sprites.
On teste si ce sprite est un vaisseau :

```
if s.isShip == true then
```

Si c'est bien un vaisseau et non pas un tir, on récupère le nombre de mouvements à effectuer :

```
local nbMvt = #liste_mvt[s.indMvt]
```

Si notre vaisseau n'a pas fini tous ses mouvements, on récupère les valeurs du mouvement :

```
local mvtX = liste_mvt[s.indMvt][s.index][1]
local mvtY = liste_mvt[s.indMvt][s.index][2]
```

On définit également 2 variables à `false` :

```
local x_fini = false
local y_fini = false
```

On traite ensuite les mouvements. On vérifie tout d'abord si le vaisseau doit aller vers la droite (`mvtX > 0`). Si c'est le cas, on vérifie si l'index `indX` est inférieur à `mvtX` (le vaisseau n'a pas fini son mouvement). Si c'est le cas, on déplace le vaisseau vers la droite et on incrémente `indX` de `0.1`. Dans le cas contraire, cela signifie que le vaisseau a terminé son mouvement vers la droite et on l'indique par `x_fini = true`.

```
-- on teste si le vaisseau doit aller vers la droite
if mvtX > 0 then
  if s.indX < mvtX then
    s.x = s.x + 1
    s.indX = s.indX + 0.1
  else
    x_fini = true
  end
```

On sait que ce n'est pas un mouvement vers la droite. Donc on vérifie si c'est un mouvement vers la gauche (`mvtX < 0`). Si c'est le cas, on vérifie si l'index `indX` est supérieur à `mvtX` (le vaisseau n'a

pas fini son mouvement vers la gauche). Si c'est le cas, on déplace le vaisseau de 2 pixels vers la gauche (rappel : le scrolling est de 1 pixel vers la gauche) et on incrémente `indX` de `0.1`. Dans le cas contraire, cela signifie que le vaisseau a terminé son mouvement vers la gauche et on l'indique par `x_fini = true`.

```
      -- on teste si le vaisseau doit aller vers la gauche
      elseif mvtX < 0 then
        if s.indX > mvtX then
          s.x = s.x - 2
          s.indX = s.indX - 0.1
        else
          x_fini = true
        end
```

On sait que ce n'est pas un mouvement vers la droite ni vers la gauche. Donc le vaisseau n'avait aucun mouvement vers la droite ou vers la gauche à faire. On l'indique par `x_fini = true`.

```
      else
        -- ici, le vaisseau ne va ni à droite ni à gauche
        x_fini = true
      end
```

C'est le même principe pour le haut et le bas :

```
      -- on teste si le vaisseau doit aller vers le bas
      if mvtY > 0 then
        if s.indY < mvtY then
          s.y = s.y + 1
          s.indY = s.indY + 0.1
        else
          y_fini = true
        end
      -- on teste si le vaisseau doit aller vers le haut
      elseif mvtY < 0 then
        if s.indY > mvtY then
          s.y = s.y - 1
          s.indY = s.indY - 0.1
        else
          y_fini = true
        end
      else
        -- ici, le vaisseau ne va ni en haut ni en bas
        y_fini = true
      end
```

Enfin, quand on sait que les mouvements vers la droite ou la gauche et le haut ou le bas sont terminés, on remet tous les index à 0 et on passe au mouvement suivant (ligne : `s.index = s.index + 1`).

```
      if x_fini == true and y_fini == true then
```



```
    -- on a fini de faire le déplacement
    -- on passe au mouvement suivant
    s.indX = 0
    s.indY = 0
    s.index = s.index + 1
    s.canFire = true
  end
```

Si on veut laisser notre vaisseau se déplacer à l'infini, dès que tous les mouvements sont terminés, on positionne `s.index` à 2.

```
  else
    -- ici, on boucle sur les mouvements du vaisseau
    s.index = 2
  end
```

Il ne reste plus que la gestion des tirs. On a initialisé la variable `canFire` à `false` dans la fonction `CreateSpriteIntro`. Cette variable a été mise à `true` lorsque le premier mouvement du vaisseau était terminé. On a également créé la variable `timer` à 0 dans `CreateSpriteIntro`. Cette variable `timer` permet de stocker la durée entre 2 tirs. Dès qu'elle arrive à 0, on crée un nouveau tir et on la réinitialise avec une valeur aléatoire comprise entre 10 et 100. Tant qu'elle n'arrive pas à 0, on lui retranche 1 à chaque exécution de la fonction `Showintro`.

```
  -- gestion des tirs
  if s.canFire == true then
    if s.timer == 0 then
      CreateShootIntro("heros", "laser1", s.x + s.l/2, s.y, 8, 0)
      -- on définit un délai aléatoire entre 10 et 100 pour tirer
      s.timer = math.random(10, 100)
    else
      s.timer = s.timer - 1
    end
  end
```

Et, pour finir, on affiche le sprite.

```
  love.graphics.draw(s.image, s.x, s.y, DegreeToRadian(s.r), 1,
     1, s.l/2, s.h/2)
```

Vous devriez obtenir la fonction suivante :

```
  function ShowIntroduction()
    -- Déclaration des variables locales
    local n, ligne, colonne, x, y
    local nbLignes = map_intro.layers[1].height
    local nbcolonnes = map_intro.layers[1].width
    local max = ((nbcolonnes * map_intro.tilesets[1].tilewidth) +
     largeur) * -1

    -- ici, on fait la boucle sur niveau
    if xScrolling <= max then
```



```
  xScrolling = 0
else
  xScrolling = xScrolling - 1
end

-- affichage du niveau
x = largeur + xScrolling
y = 0
for ligne = 1, nbLignes do
  for colonne = 1, nbcolonnes do
    local tuile = level_intro[(ligne - 1) * nbcolonnes +
       colonne]
    -- Dessine la tuile
    if tuile > 0 then
      love.graphics.draw(imgTuiles[tuile], x, y, 0, 1, 1)
    end
    x = x + 32
  end
  x = largeur + xScrolling
  y = y + 32
end

-- boucle sur les tirs
for n=#liste_tirs_intro, 1, -1 do
  local tir = liste_tirs_intro[n]
  tir.x = tir.x + tir.vx
  tir.y = tir.y + tir.vy
  if tir.x > largeur then
    -- ici, on efface le tir
    tir.toDelete = true
    table.remove(liste_tirs_intro, n)
  end
end

-- boucle sur les sprites
for n=1, #liste_sprites_intro do
  local s = liste_sprites_intro[n]
  -- test si c'est un vaisseau
  if s.isShip == true then
    local nbMvt = #liste_mvt[s.indMvt]
    if s.index <= nbMvt then
      --ici, on applique le mouvement défini
      local mvtX = liste_mvt[s.indMvt][s.index][1]
      local mvtY = liste_mvt[s.indMvt][s.index][2]
      local x_fini = false
      local y_fini = false

      -- on teste si le vaisseau doit aller vers la droite
```



```
if mvtX > 0 then
  if s.indX < mvtX then
    s.x = s.x + 1
    s.indX = s.indX + 0.1
  else
    x_fini = true
  end
-- on teste si le vaisseau doit aller vers la gauche
elseif mvtX < 0 then
  if s.indX > mvtX then
    s.x = s.x - 2
    s.indX = s.indX - 0.1
  else
    x_fini = true
  end
else
  -- ici, le vaisseau ne va ni à droite ni à gauche
  x_fini = true
end

-- on teste si le vaisseau doit aller vers le bas
if mvtY > 0 then
  if s.indY < mvtY then
    s.y = s.y + 1
    s.indY = s.indY + 0.1
  else
    y_fini = true
  end
-- on teste si le vaisseau doit aller vers le haut
elseif mvtY < 0 then
  if s.indY > mvtY then
    s.y = s.y - 1
    s.indY = s.indY - 0.1
  else
    y_fini = true
  end
else
  -- ici, le vaisseau ne va ni en haut ni en bas
  y_fini = true
end

if x_fini == true and y_fini == true then
  -- on a fini de faire le déplacement
  -- on passe au mouvement suivant
  s.indX = 0
  s.indY = 0
  s.index = s.index + 1
  s.canFire = true
```



```
            end
          else
            -- ici, on boucle sur les mouvements du vaisseau
            s.index = 2
          end

          -- gestion des tirs
          if s.canFire == true then
            if s.timer == 0 then
              CreateShootIntro("heros","laser1",s.x+s.l/2,s.y,8,0)
              -- délai aléatoire entre 10 et 100 pour tirer
              s.timer = math.random(10, 100)
            else
              s.timer = s.timer - 1
            end
          end
        end

        love.graphics.draw(s.image, s.x, s.y, DegreeToRadian(s.r),
            1, 1, s.l/2, s.h/2)
      end
    end
```

A ce stade, il nous reste une dernière chose pour que tout soit complet : la fonction `DegreeToRadian` utilisée dans la fonction `love.graphics.draw` ci-dessus. Cette fonction permet de convertir une valeur de degré en radian.

```
function DegreeToRadian(pDegree)
  return math.pi * (pDegree) / 180
end
```

Pour l'explication, nous avions rajouté le paramètre `pRotation` dans la fonction `CreateSpriteIntro`. Or, la fonction `love.graphics.draw` n'utilise pas les degrés mais les radians pour effectuer les rotations sur les images.

Je vous fournis également la fonction permettant de passer de radian en degré (on ne sait jamais ^^).

```
function RadianToDegree(pRadian)
  return 180 * (pRadian) / math.pi
end
```

Il ne vous reste plus qu'une dernière chose : ajouter votre nouvelle intro au jeu (^_^)
Pour cela, il faut ajouter la fonction InitIntro dans la fonction love.load(), juste avant la fonction DemarrerJeu. Cela devrait vous donner ceci :

```
function love.load()
  love.window.setMode(1024,768)
  love.window.setTitle("Atelier Shooter Gamecodeur")
  largeur = love.graphics.getWidth()
```



```
  hauteur = love.graphics.getHeight()
  heros = CreeSprite("heros", largeur/2, hauteur/2)

  InitIntro()

  DemarreJeu()
end
```

Vous devriez avoir également une fonction updateMenu qui ne contient aucune ligne. Ici, nous allons rajouter l'appel à notre fonction Showintroduction() pour obtenir ceci :

```
function updateMenu()
  ShowIntroduction()
end
```

Voilà, normalement, si vous avez bien suivi les indications ci-dessus, vous devriez obtenir le même résultat que sur la vidéo.

J'attends vos plus belles vidéos d'introduction sur vos shooters.
Egalement, si vous avez des améliorations à proposer, je suis toujours à l'écoute de bonnes idées (^_^)
A très bientôt,

Dominique LACOMBE

