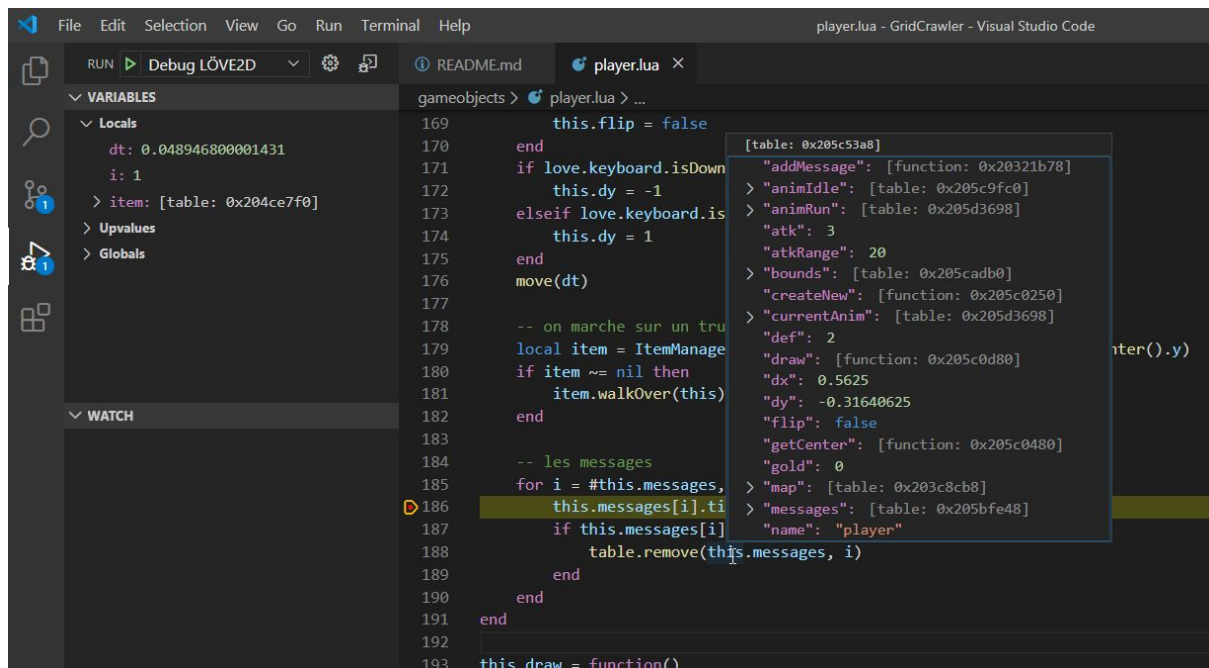


LUA - LÖVE2D

Debug Pas à pas avec VSCode

Ce document a pour but de lister les opérations à effectuer afin de pouvoir coder nos jeux en LUA et LÖVE2D avec VS Code, et ce afin d'avoir toutes les fonctionnalités suivantes :

- Coloration syntaxique
- Auto-complétion
- Lancement de l'application
- Débogage pas à pas



Je pars du principe que Visual Studio Code est déjà installé sur votre machine.

Pour tester le paramétrage, je pars d'un PC vide de toute installation (VirtualBox), juste VS Code et love2D (version 11.3, 64bits dans mon cas).

Créer un nouveau projet

Pour démarrer un nouveau projet, j'utilise une version un peu évoluée du squelette de projet de David. Rien de très avancé finalement, j'ai juste déplacé quelques lignes de code du main.lua vers le fichier conf.lua et ajouter des liens utiles en début du fichier main.lua :

main.lua

```
-- [Wiki Love2D](https://love2d.org/wiki/Main_Page), [Wiki LUA](https://www.lua.org/docs.html)
-- [Fonction mathématiques Love2D](https://love2d.org/wiki/General_math)

love.graphics.setDefaultFilter("nearest") -- pas d'aliasing pour la 2D old school

function love.load()
    love.window.setTitle("<NOM DU JEU> (by Wile)")
```

```

--love.window.setMode(1280, 768)
WIDTH = love.graphics.getWidth()
HEIGHT = love.graphics.getHeight()
end

function love.update(dt)
    print("dt : "..dt)
end

function love.draw()
end

function love.keypressed(key)
    if key=="escape" then love.event.quit() end
    print(key)
end

```

conf.lua:

```

if os.getenv("LOCAL_LUA_DEBUGGER_VSCODE") == "1" then
    require("lldebugger").start()
end
io.stdout:setvbuf("no")
if arg[#arg] == "-debug" then require("mobdebug").start() end -- debug pas à pas

```

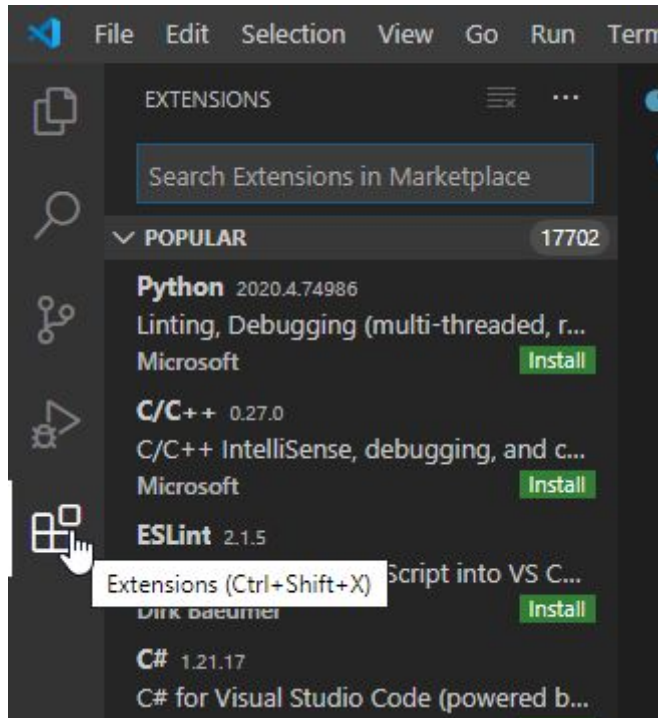
J'ai ces 2 fichiers de côté et quand je crée un nouveau projet, je les copie comme le squelette de code de David, avec en plus la création des dossiers "sons" et "images", pour préparer un espace de travail le plus propre possible.

Remarque : pour que certaines extensions fonctionnent, il faut éviter les espaces dans le nom du dossier où se trouvent nos sources.

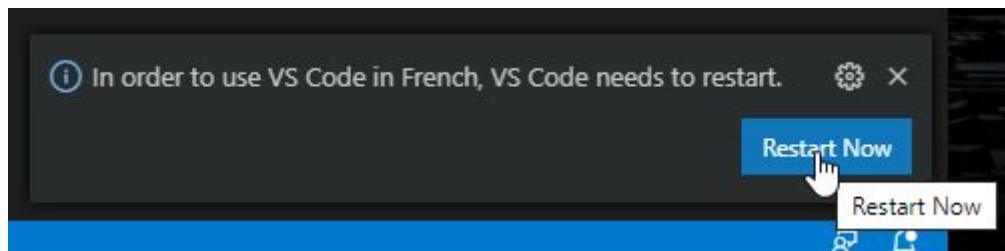
Extensions VSCode utilisées

Tout d'abord, j'ai installé le pack de langue Français. Certains d'entre-vous préfère travailler avec des outils en français, ça peut se comprendre. L'extension "French Language Pack" de Microsoft est le pack officiel, donc pas de soucis.

Pour ceux qui ne connaissent pas VSCode et comment installer une extension, c'est tout simple. Commençons par installer pour s'entraîner l'extension du pack français. Cliquer dans la marge sur l'icône des extensions (Ctrl+Shift + X).



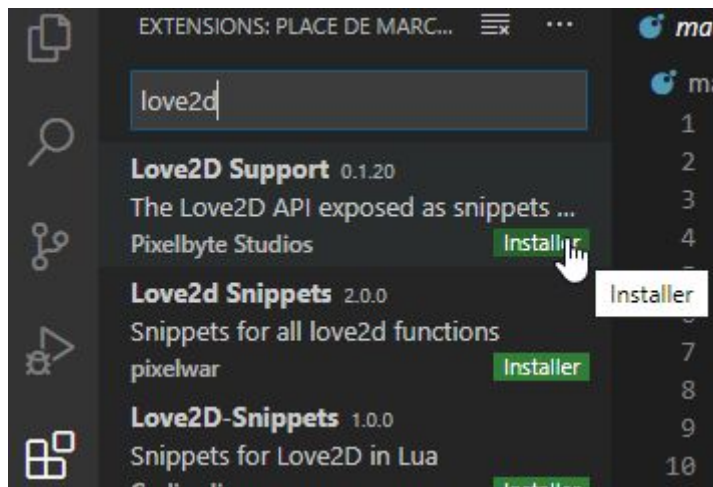
Dans la zone de recherche, taper "french" puis valider. L'extension à choisir est celle de Microsoft, généralement la première proposée. La sélectionner puis cliquer sur l'icône verte "Install". L'extension s'installe et propose de redémarrer VSCode pour être prise en compte.



Voilà, nous avons installé notre première extension et maintenant, VSCode est en français. Nous pouvons rentrer dans le vif du sujet, installer le reste pour déboguer nos projets.

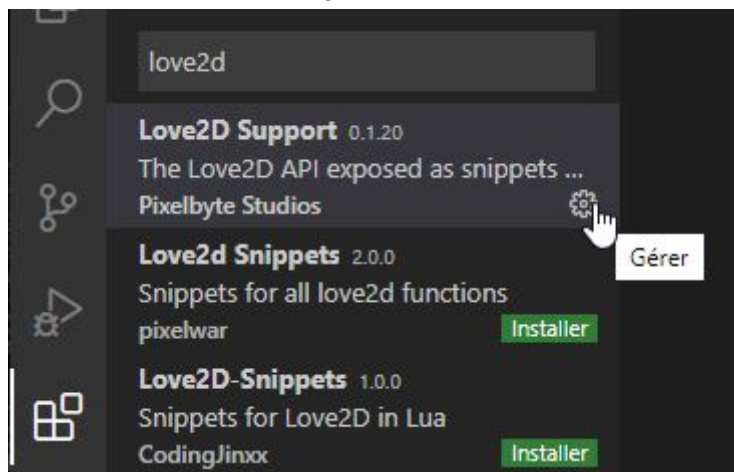
"Love2D Support" de Pixelbyte Studios

Cette extension permet de "compiler" et de lancer notre application en utilisant "ALT+L"

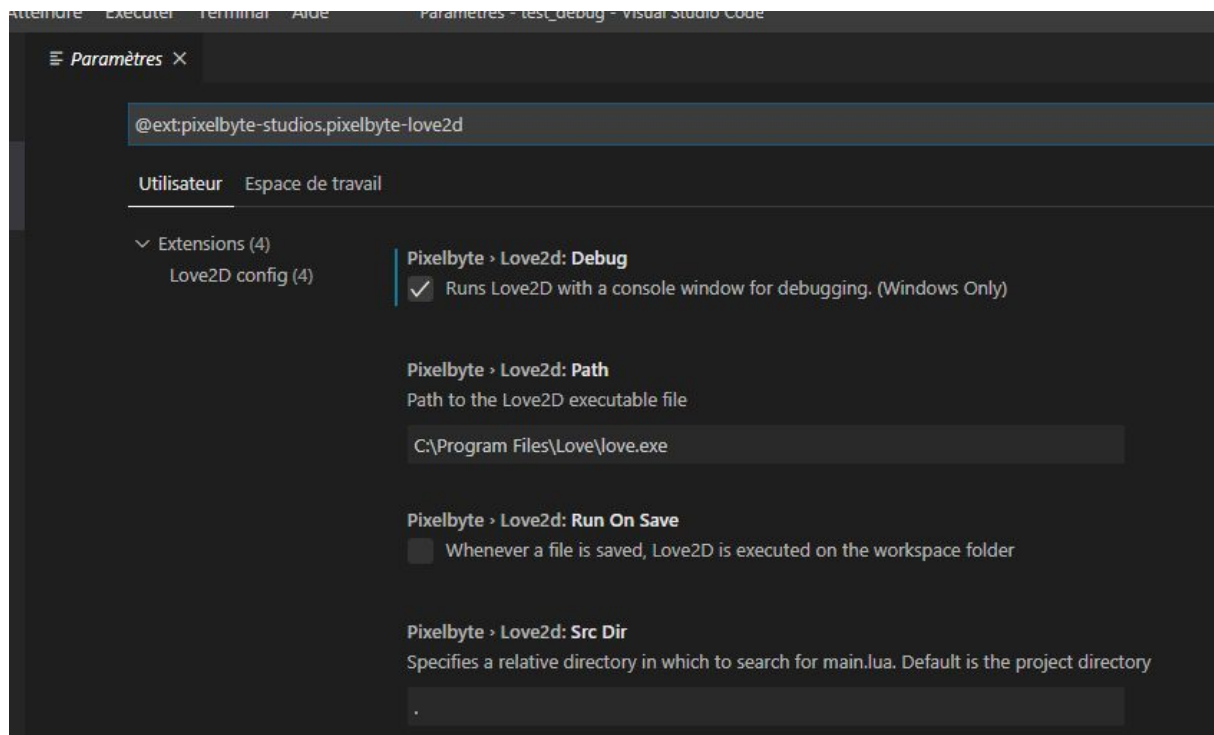


Une fois que l'extension est installée, vérifier dans son paramétrage qu'elle pointe bien vers love.exe.

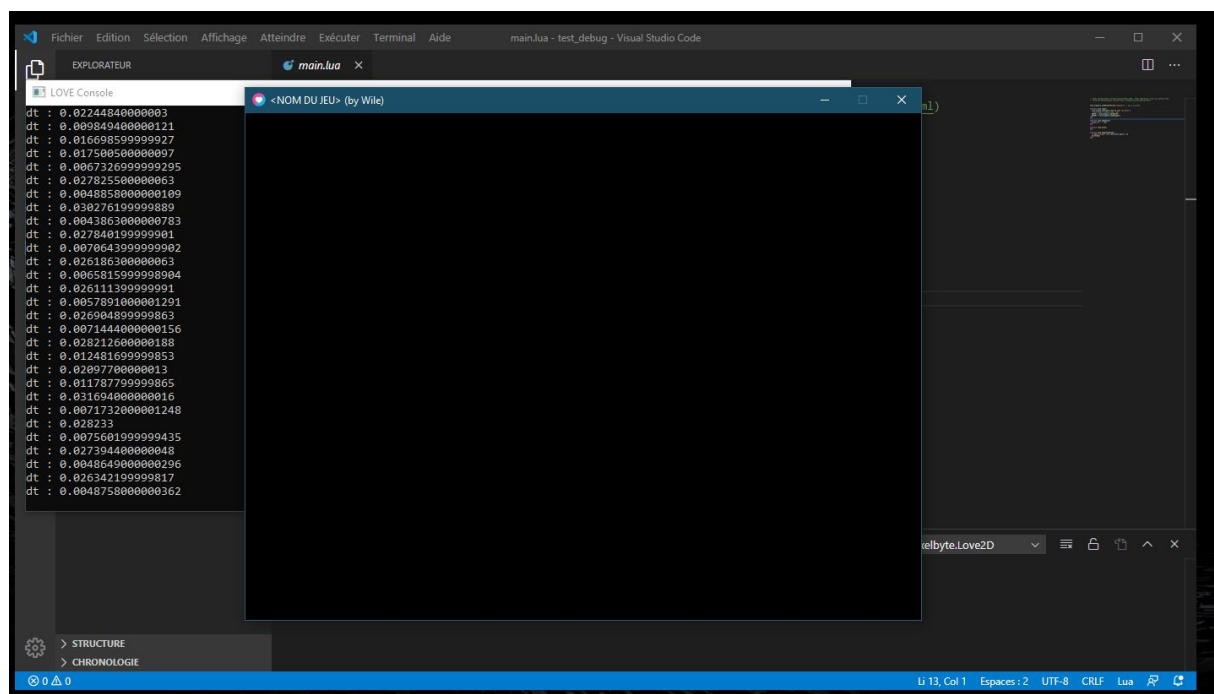
Pour ouvrir le paramétrage d'une extension, cliquer sur la petite roue-dentée de l'extension :



Cocher aussi la case "Run Love2D with a console..." :



Fermer l'onglet du paramétrage et revenons à notre projet. La fenêtre "main.lua" est affichée, nous pouvons faire "ALT + L" et si tout va bien, notre programme se lance !!!



Parfait ! La console affiche bien le "dt" de la fonction update, ça permet de vérifier que la gameloop le lua tourne correctement.

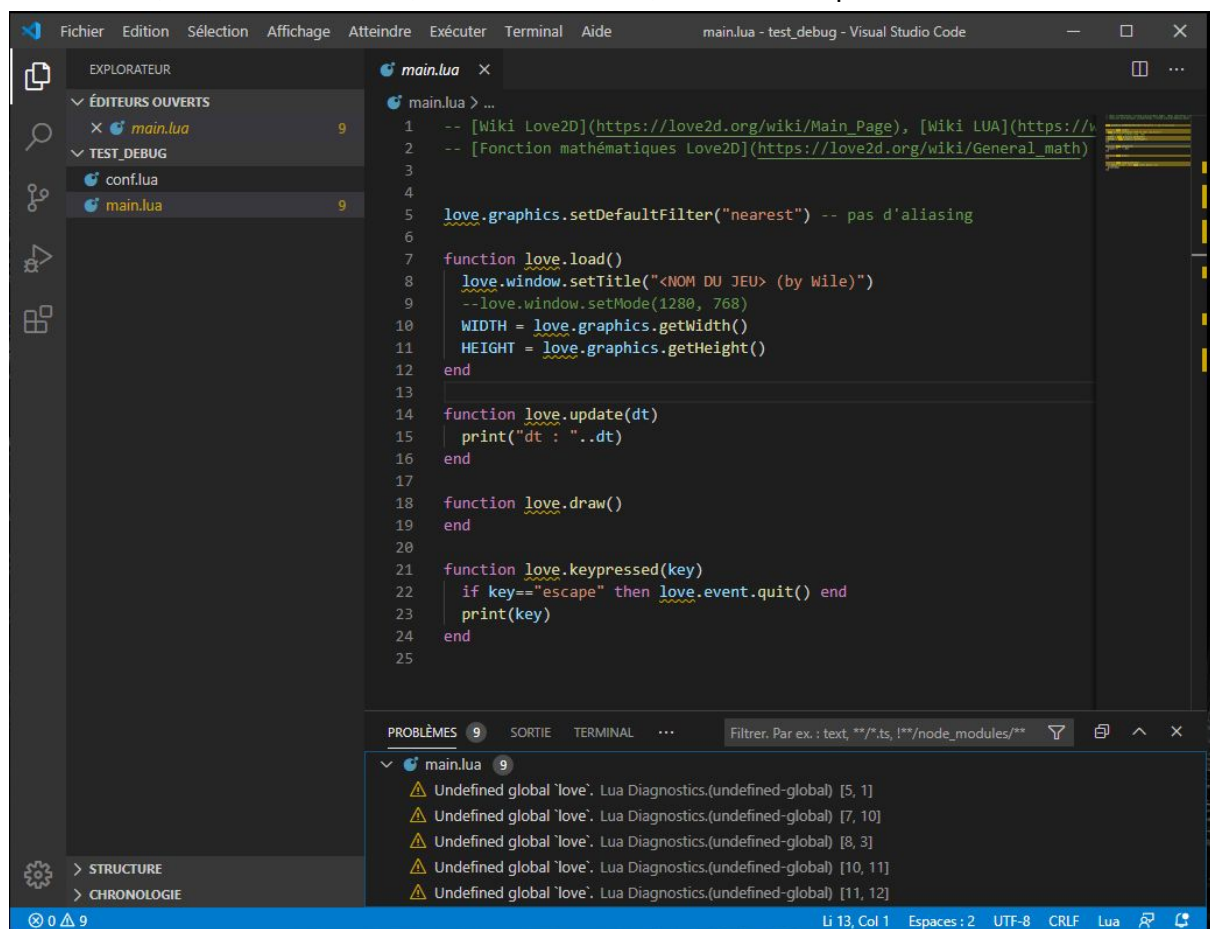
“Lua” de sumneko

Cette extension permet énormément de chose pour Lua. Une auto-complétion, de l’analyse de code, bref, je rentre pas dans les détails, mais elle est absolument indispensable.

Rien que la page l’onglet de l’extension fait bavé avec toutes les captures d’écran qu’il affiche.

Cette extension installée, elle nous dit qu’il y a des problèmes dans notre code. Tiens, c’est étrange, il fonctionne pourtant !

On voit déjà comment fonctionne cette extension : elle indique en jaune dans la liste des fichiers ceux ayant des erreurs, nous l’affiche aussi à droite pour trouver les lignes concernées, et dans la console, en bas de l’écran, liste tous les problèmes :



L’erreur que notre projet rencontre n’est pas réellement un problème. C’est simplement que pour l’extension, elle détecte l’utilisation d’une “variable” dont elle ne détecte pas la déclaration. Excellent !! Par contre, cette variable, c’est “love”... c’est normal qu’elle ne soit pas déclarée celle-là !

En fait, on comprend ce qu’il se passe. Le framework, “Löve2D” étant implicitement déclaré dans tout projet, cette variable ne sera jamais déclarée. L’extension ne le sait pas, il va falloir lui expliquer que si elle voit une variable nommé “love”, c’est normal.

Pour cela, en passant la souris sur un terme “love” souligné en erreur dans le code, une popup s’affiche. Impressionnante d’ailleurs cette popup, elle donne pas mal d’information sur notre code !

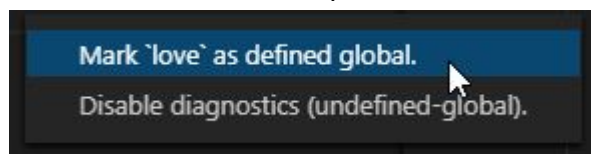


```
5 love.graphics.setDefaultFilter("nearest") -- pas d'aliasing
6
7 function
8   love.wi
9   --love.
10  WIDTH =
11  HEIGHT
12 end
13
14 function
15   print("
16 end
17
18 function love.draw()
19 end
```

global love: {
draw: function,
event: table,
graphics: table,
keypressed: function,
load: function,
update: function,
window: table,
}

Undefined global `love`. Lua Diagnostics.(undefined-global)
Aperçu du problème (Alt+F8) Correction rapide... (Ctrl+;)

Je vous laisse cliquer sur “Aperçu du problème” pour vous laisser entrevoir les possibilités de cette extension, mais pour le moment, nous allons utiliser la Correction rapide :



En réalisant cette action, nous indiquons à l’extension que “love” est une variable globale que l’on ne va jamais déclarer.

Nous remarquons alors qu’un nouveau dossier apparaît : le dossier “.vscode” dans lequel VSCode a créé un fichier. C’est dans ce fichier que VSCode stock les paramètres propres à notre projet.

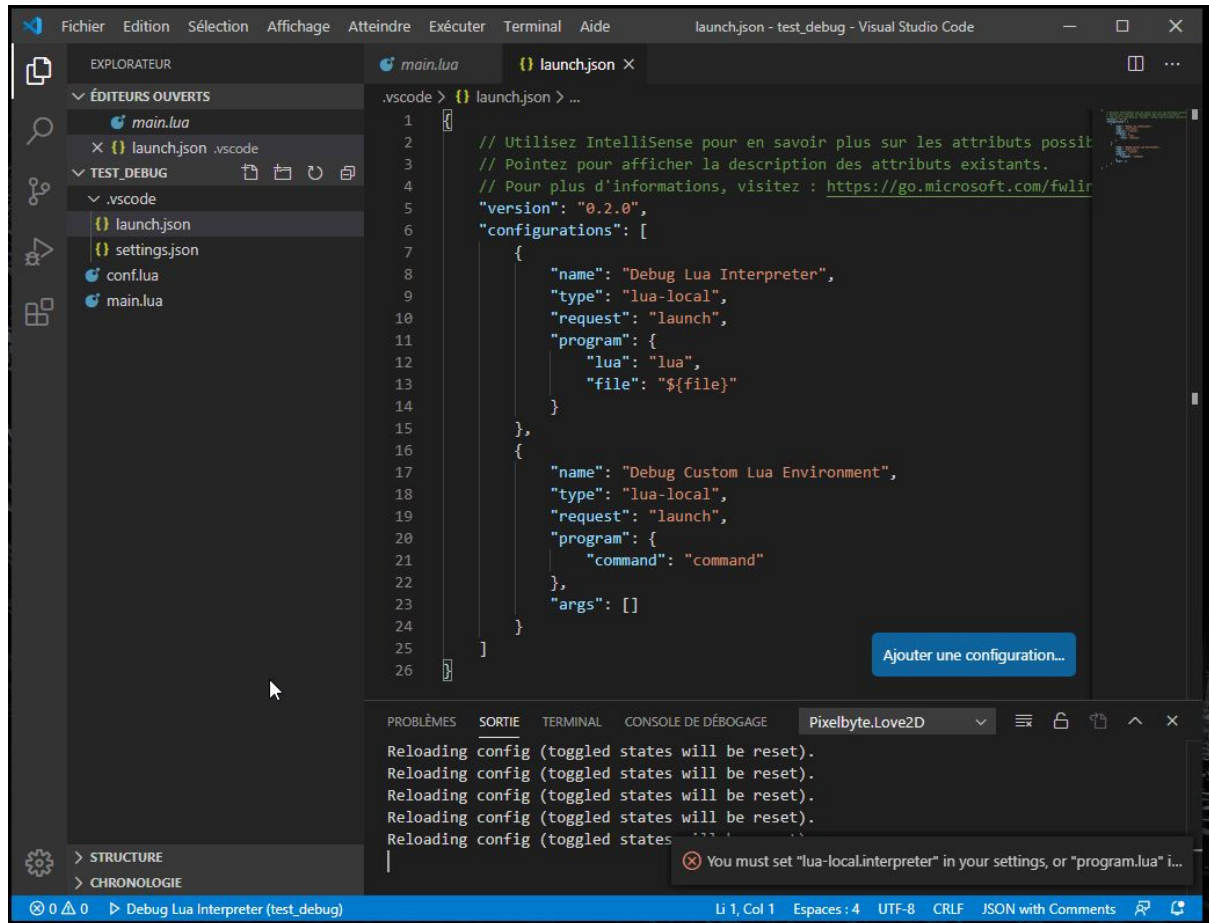
Une fois cette opération faite, notre code est totalement propre, plus d’erreurs, cool ! Un petit “ALT + L” permet de vérifier, ça tourne toujours nickel.

“Local Lua Debugger” de Tom Blind

Nous y voici, c’est cette extension qui va nous permettre de faire du pas à pas dans notre code.

L’installation de l’extension est toute bête, c’est son paramétrage qui est un peu plus compliqué, quoique, une fois qu’on a compris, comme d’hab, c’est tout con ;)

Affichons notre main.lua et lançons une session de debug. Pour cela, faisons “F5”.
Aïe !!!



VSCode nous indique qu’il lui manque des informations pour pouvoir déboguer.

En effet, nous avons installé l’extension, mais nous n’avons pas dit à VSCode de l’utiliser pour déboguer. C’est certainement là que certains ont bloqué, car c’est à ce moment que j’ai fouillé partout ;)

VSCode nous a rajouté un fichier “launch.json” dans son dossier “.vscode”. Ce fichier lui permet de savoir quoi faire lorsqu’on “run” un projet, quoi faire qu’on on fait “F5”.
Le fichier json est à modifier pour lui indiquer qu’il faut utiliser l’extension “lua-local”.

Vider son contenu est remplacez-le par celui-ci :

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug LOVE2D",
      "type": "lua-local",
      "request": "launch",
      "program": {
        "command": "C:/Program Files/LOVE/love.exe"
      },
      "args": [
        "${workspaceFolder}"
      ]
    }
  ]
}
```

Vérifier bien que votre chemin vers “love.exe” est correct, en prenant garde à mettre des “/” et non des “\” (façon Linux).

La suite de la configuration est la présence de cette ligne de code dans le fichier **conf.lua** du projet :

```
if os.getenv("LOCAL_LUA_DEBUGGER_VSCODE") == "1" then
  require("lldebugger").start()
end
```

Elle permet de faire un *require* du débogueur et de le lancer *.start()*, seulement si *LOCAL_LUA_DEBUGGER_VSCODE* est positionné.

En fait, c’est simplement pour lancer le debug que s’il est exécuté depuis VSCode. Notre projet déployé ne l’utilisera pas et sera donc optimisé, pas de crainte de ce côté.

Car oui, le débogage pas à pas lancé, ça fait tourner une grosse machinerie qui ralentit l’exécution. C’est à utiliser avec parcimonie (ou un pote à lui s’il est pas dispo), mais le reste du temps, lancez votre projet avec ALT+L tout simplement.

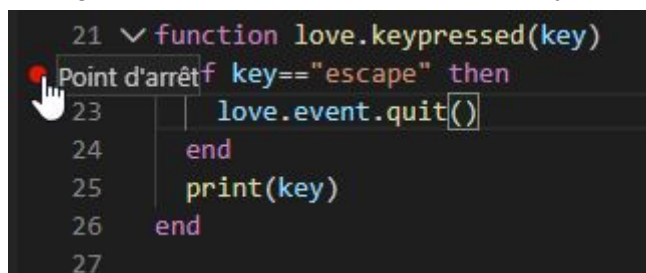
Le pas à pas est à utiliser quand on problème

Maintenant, testons avec “F5” : Yes, notre programme se lance toujours !

Vérifions le pas à pas.

Pour mettre un point d’arrêt, soit cliquer dans la marge, soit faire “F9” sur la ligne voulue.

Deboguons pour tester la fonction *love.keypressed* :



```
21 function love.keypressed(key)
22   if key=="escape" then
23     love.event.quit()
24   end
25   print(key)
26 end
27
```

The screenshot shows a code editor with a dark theme. A function `love.keypressed` is defined. A red dot, representing a breakpoint, is placed in the left margin next to line 22, which contains the line `if key=="escape" then`. A tooltip next to the red dot says "Point d'arrêt". The code continues with `love.event.quit()` on line 23, `end` on line 24, and `print(key)` on line 25, followed by another `end` on line 26.

Lançons en mode debug avec F5, et si on appuie sur une touche, BINGO !!! VSCode fige l'exécution sur notre point d'arrêt.

Le pas à pas se réalise avec les raccourcis F10, F11, etc... ou avec les boutons de débogage :



Remarques : lorsqu'on lance notre projet avec F5 (et non ALT+L), le projet n'est pas lancé de la même façon. Dans cette configuration, la console n'est pas affichée, tous nos `print("toto")` n'apparaissent pas. La console de VSCode sera alors utilisée et rafraîchie que lorsqu'un point d'arrêt sera rencontré.

