

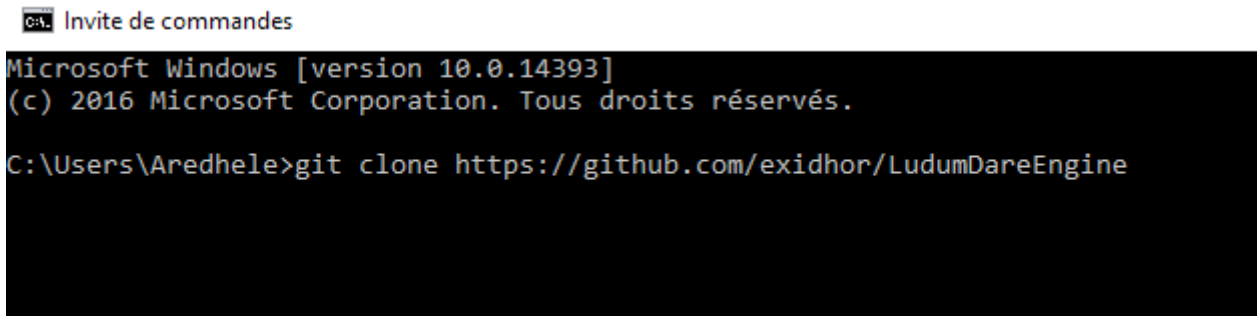
Tutoriel de création de projet « Ludm Dare Ready »

© 2016 Windfall



I) Récupérer le projet sur Github

Pour commencer, on clone le dépôt : <https://github.com/exidhor/LudumDareEngine>
Ce n'est normalement pas très dur, voici tout de même une capture d'écran :



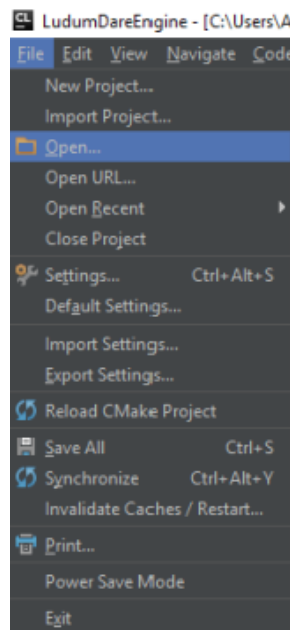
```
Invite de commandes

Microsoft Windows [version 10.0.14393]
(c) 2016 Microsoft Corporation. Tous droits réservés.

C:\Users\Aredhele>git clone https://github.com/exidhor/LudumDareEngine
```

II) Ouvrir le projet avec CLion

Une fois le projet cloné, il suffit d'ouvrir CLion (pour la grande compatibilité avec CMake) et de faire « File » puis « Open ». On sélectionne le dossier contenant le projet précédemment cloné, puis « Ok ».



Une fois le projet ouvert, CLion va charger les CMake files et automatiquement vérifier le système d'exploitation et le compilateur. CMake vérifiera aussi la présence du fichier de configuration des dépendances externes comme la SFML. Si ce fichier n'existe pas, il sera créé et devra être complété. Voici un exemple :

```

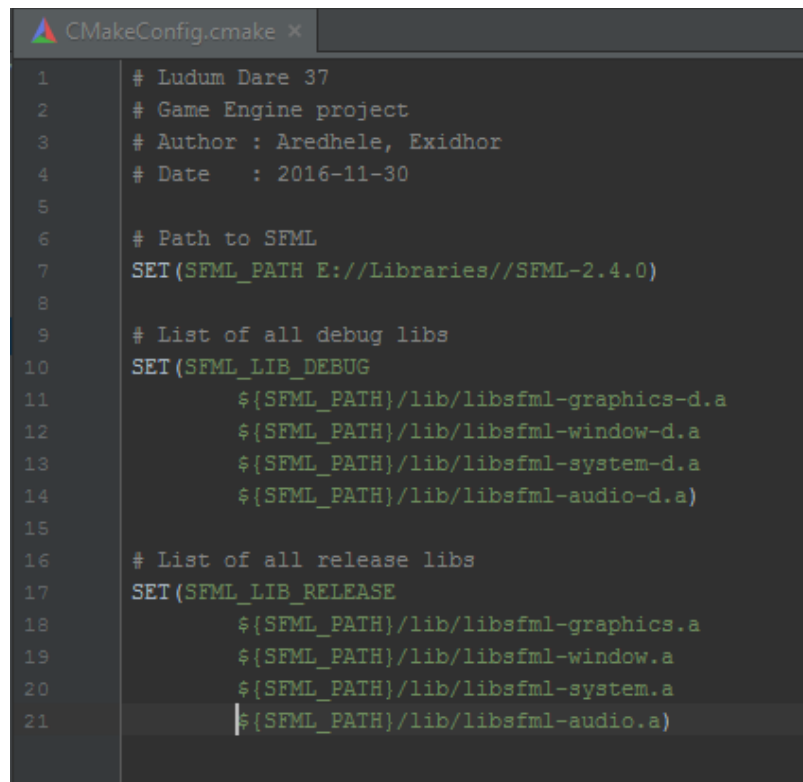
-- The C compiler identification is GNU 4.9.2
-- The CXX compiler identification is GNU 4.9.2
-- Check for working C compiler: E:/Libraries/mingw32/bin/gcc.exe
-- Check for working C compiler: E:/Libraries/mingw32/bin/gcc.exe -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: E:/Libraries/mingw32/bin/g++.exe
-- Check for working CXX compiler: E:/Libraries/mingw32/bin/g++.exe -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Engine : LudumDareEngine 0.1.0
-- Compiler : GCC
-- Platform : Windows
-- Unable to find CMakeConfig.cmake
-- Creating CMakeConfig.cmake
CMake Warning at CMakeLists.txt:92 (MESSAGE):
  Please set the cmake config

-- Compilation mode : Debug
-- Configuring done
-- Generating done

```

III) Paramétrer les chemins vers la SFML

Sur la dernière capture d'écran, CMake nous indique clairement qu'il faut paramétrer le fichier de configuration. Voici un exemple typique de configuration :



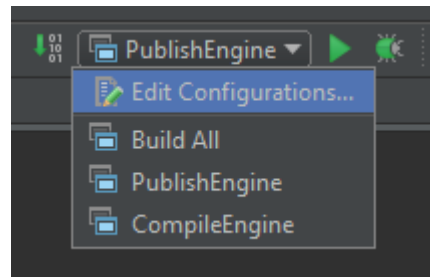
```

CMakeConfig.cmake x
1  # Ludum Dare 37
2  # Game Engine project
3  # Author : Aredhele, Exidhor
4  # Date   : 2016-11-30
5
6  # Path to SFML
7  SET(SFML_PATH E://Libraries//SFML-2.4.0)
8
9  # List of all debug libs
10 SET(SFML_LIB_DEBUG
11     ${SFML_PATH}/lib/libsfml-graphics-d.a
12     ${SFML_PATH}/lib/libsfml-window-d.a
13     ${SFML_PATH}/lib/libsfml-system-d.a
14     ${SFML_PATH}/lib/libsfml-audio-d.a)
15
16 # List of all release libs
17 SET(SFML_LIB_RELEASE
18     ${SFML_PATH}/lib/libsfml-graphics.a
19     ${SFML_PATH}/lib/libsfml-window.a
20     ${SFML_PATH}/lib/libsfml-system.a
21     ${SFML_PATH}/lib/libsfml-audio.a)

```

IV) Build & Publish

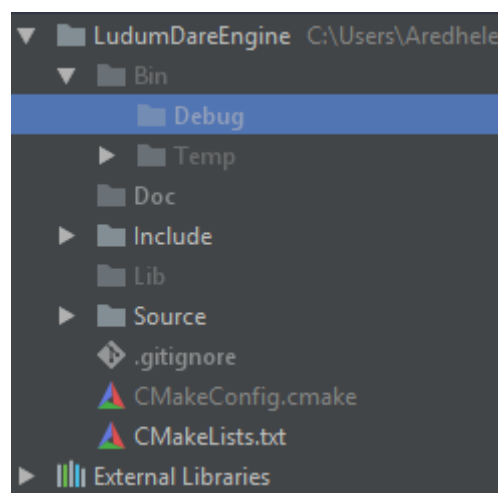
Une fois cela effectué, c'est déjà presque fini. Arrivé à ce stage, l'IDE devrait proposer deux cibles CMake différentes : CompileEngine et PublishEngine.



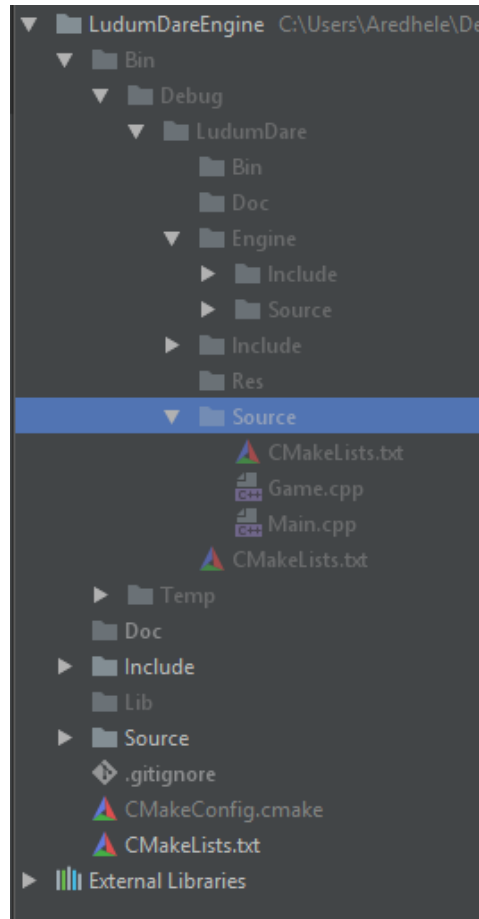
- CompileEngine :
 - Permet de compiler normalement le projet dans son état et de le tester
- PublishEngine :
 - Permet de créer une version prêt à l'emploi du moteur dans un contexte d'utilisation pré-construit. Cela crée un projet entier avec tous ses répertoires et le moteur est placé sous forme de sources dans un dossier à part pour ne pas polluer le projet et le projet CMake déjà prêt. Il suffira de prendre le projet généré et ... c'est tout. Il est directement possible de coder un jeu. CMake se charge de tout, de la compilation du jeu et du moteur ensemble, des répertoires de headers, du chemin de sorti, et il n'y a pas besoin d'ajouter les fichiers sources aux CMake files, CMake ira lui même les chercher en temps voulu.

Voici un aperçu :

- Avant la publication du moteur :



- Après publication :



De plus, la publication a pour dépendance la compilation du moteur. Si une publication est effectuée alors que le dernier build du moteur date, les sources qui ne sont pas à jour seront compilées pour que la version publiée du moteur soit la plus récente possible.

V) Un projet prêt à l'emploi immédiat

La conclusion est que le projet généré est directement utilisable ! En effet, il suffit d'ouvrir avec le projet généré, de simplement configurer ses chemins vers SFML, et de .. presser le petit triangle pour compiler. Ni plus ni moins. Il est prévu que l'utilisateur ajoute son code dans les méthodes de la classe Game prévues à cet effet. Des tutoriels d'utilisation du moteur sont prévus et seront aussi ajouter au projet utilisateur lors de la publication du moteur.

Voici une capture faisant preuve :

CL LudumDare - [C:\Users\Aredhele\Desktop\LudumDare] - ... \Source\Game.cpp - CLion 2016.3

File Edit View Navigate Code Refactor Run Tools VCS Window Help

LudumDare > Source > Game.cpp

Project

- LudumDare C:\Users\Aredhele\Desktop
- Bin
- Engine
 - Include
 - Source
- Include
 - Game.hpp
- Res
- Source
 - CMakeLists.txt
 - Game.cpp
 - Main.cpp
- CMakeConfig.cmake
- CMakeLists.txt
- External Libraries

1 #include "Game.hpp"

2

3 /// \brief Default constructor

4 /* explicit */ Game::Game ()

5 {

6 // Code

7 }

8

9 /// \brief Default destructor

10 Game::~Game (void)

11 {

12 // Code

13 }

14

15 /// \brief Called before engine initialization

16 void Game::OnPreInitialize (void)

17 {

18 // Code

19 }

20

21 /// \brief Called after engine initialization

22 void Game::OnPostInitialize (void)

23 {

24 // Code

25 }

26

27 /// \brief Called before engine update

28 void Game::OnPreUpdate (float dt)

29 {

30 // Code

31 }

32

33 /// \brief Called after engine update

34 /// \brief dt The elapsed time since the last update

35 void Game::OnPostUpdate (float dt)

36 {

37 // Code

38 // On peut coder le jeu ici sans crainte ;)

39 }

40

41 /// \brief Called before the engine release

42 void Game::OnPreRelease (void)

43 {

44 // Code

45 }

46

47 /// \Brief Called after the engine release

48 void Game::OnPostRelease (void)

49 {

50 // Code

51 }

Voilà, c'est fini, j'espère que ça t'aura plu ! Pour plus d'informations, merci de me contacter au 06 46 ... Euh ok, envoie moi un message sur Discord !

Je suis resté sur un truc super simple, j'ai hésité à faire construire à CMake le moteur sous forme d'archive statique (.a) mais j'ai pensé que tu penserais que cela serait une mauvaise idée car il est préférable d'avoir le moteur sous forme de sources pour pouvoir le hotfix ou rajouter une feature sans refaire toute la production du moteur et le redonner à tout le monde ...

