# Project Configuration

## Introducción

Hasta ahora, todos los proyectos que habíamos realizado en C++ han utilizado el sistema de construcción de Microsoft llamado **MSBuild.**

Pero existen alternativas a este sistema que pueden ser más interesantes según las necesidades del proyecto que estés realizando.

## ¿Qué es un build system?

Es una o un conjunto de herramientas que automatizan el proceso de construcción de un proyecto. Te permiten gestionar directorios (source files, include files, etc.), bibliotecas externas, configuraciones del compilador, tests, etc.

Suelen ser muy flexibles y útiles para grandes proyectos.

Algunos ejemplos de build systems:

- **CMake**
- **MSBuild**
- Build2 (dependency manager incorporado)
- Scons (configuración con scripts de python)
- Ninja
- Make
- Premake (configuración con scripts de lua)

## ¿Qué es un compilador y como funciona?

El compilador es una pieza de software que transforma código en C++ a un binario que la máquina pueda entender.

Este funciona de la siguiente manera:

1. El build system entrega al compilador los archivos C++ deseados por el programador + ciertas configuraciones específicas para el proyecto.
2. El compilador convierte cada .cpp en un .o / . obj (ensamblador). Da por hecho que las referencias externas se cumplen.
3. El linker intenta resolver esas referencias externas y unifica el código en ensamblador en un único binario.

Gracias a **Jason Gregory** por la explicación.

## Como funciona MSBuild

MSBuild es una solución all-in-one desarrollada por Microsoft para su IDE Visual Studio. Esta, gestiona los proyectos con soluciones (.sln) que a su vez internamente contienen uno o varios subproyectos (.vcproj). Toda esta información es almacenada en forma de xml que se pueden editar manualmente o con una UI.

Además, este sistema utiliza un compilador propio llamado **MSVC.**

La configuración de este sistema se basa en 3 partes esenciales:

- Definir donde puede encontrar los **source files.**
- Definir en que directorio de trabajo quieres trabajar (desde donde partirán todas las direcciones relativas).
- Por último, decirle al linker donde puede encontrar las librerías necesarias para tu proyecto (.dll / .so para las librerías dinámicas o .lib / .a para las estáticas).

## Como funciona CMake

Es un build system open-source y multiplataforma creado por kitware. Este se ha convertido en el estándar de la industria para proyectos multiplataforma en C++.

Entre las características que lo caracterizan, nos encontramos con que es muy flexible y organizado, que es compatible con la mayoría de IDE's, que es más complejo de configurar que otros tipos de build systems, y por último, que utiliza un lenguaje propio basado en macros para configurar los proyectos.

La configuración de este sistema requiere de las siguientes características:

- La creación de una archivo CmakeLists.txt, en el que se define la configuración del proyecto y como CMake debe construirlo.

Un ejemplo de CmakeLists.txt utilizado en el proyecto de la asignatura de Desarrollo de Videojuegos:

```
cmake_minimum_required(VERSION 3.5.0)
project(Gato-Juego VERSION 0.1.0 LANGUAGES C CXX)

include(CTest)
enable_testing()

set(SOURCE_DIR "${CMAKE_CURRENT_SOURCE_DIR}/Game/Source")
set(EXECUTABLE_OUTPUT_PATH "${CMAKE_CURRENT_SOURCE_DIR}/Output_Linux")

set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fpermissive -std=c++2a  -l Box2D")

file(GLOB SOURCES "${SOURCE_DIR}/*.cpp" "${SOURCE_DIR}/*.h")
list(APPEND SOURCES "${SOURCE_DIR}/External/Optick/include/optick.h")
file(GLOB_RECURSE PLAYER_SOURCES "${SOURCE_DIR}/States/*.cpp" "${SOURCE_DIR}/States/*.h")
list(APPEND SOURCES ${PLAYER_SOURCES})

include_directories(${SOURCE_DIR})

find_package(SDL2 REQUIRED)
find_package(SDL2_mixer REQUIRED)
find_package(SDL2_image REQUIRED)
find_package(SDL2_ttf REQUIRED)
find_package(pugixml REQUIRED)

add_executable(Gato-Juego ${SOURCES})

include_directories(Gato-Juego ${SDL2_INCLUDE_DIRS} ${SDL2_MIXER_INCLUDE_DIRS} ${SDL2_IMAGE_INCLUDE_DIRS} ${SDL2_TTF_INCLUDE_DIRS} ${PUGIXML_INCLUDE_DIRS})

#target_compile_options(${PROJECT_NAME} PRIVATE $<$<C_COMPILER_ID:MSVC>:/W4 /WX>)
#target_compile_options(${PROJECT_NAME} PRIVATE $<$<NOT:$<C_COMPILER_ID:MSVC>>:-Wall -Wextra -pedantic -Werror>)
add_compile_options(-Winconsistent-missing-override)
target_link_libraries(Gato-Juego PRIVATE SDL2::SDL2 SDL2_image::SDL2_image SDL2_mixer::SDL2_mixer SDL2_ttf::SDL2_ttf pugixml::pugixml ${CMAKE_SOURCE_DIR}/Output_Linux/lib/libOptickCore.so)
```

## Vcpkg Dependency Manager

Vcpkg es una solución creada por Microsoft para gestionar dependencias para C++. Este es multiplataforma, de código abierto y está soportado por los IDE's más populares.

Vcpkg tiene dos modos: Classic y Manifest. El que nos interesa es el modo Manifest, ya que este gestiona la instalación de las dependencias automáticamente.

Para utilizar el modo Manifest se necesita un archivo vcpkg.json (que se puede crear manualmente o por comandos).

Un ejemplo de vcpkg.json podría ser el siguiente:

```
{
  "name": "opengl-glfw",
  "version": "0.1",
  "builtin-baseline": "2c401863dd54a640aeb26ed736c55489c079323b",
  "dependencies": [
    "glfw3",
    "assimp"
  ]
}
```

Por último, se puede utilizar con muchos build systems, pero nativamente está soportado por CMake y MSBuild.

Para configurar **vcpkg** en **MSBuild** hay que seguir los siguientes pasos:

- Abrir la **Developer Command Prompt** o **Developer PowerShell** y introducir ***vcpkg integrate install***
- Activar en la config del proyecto **"Use Vcpkg Manifest"** a "**Yes**"

Para configurar **vcpkg** en **CMake** hay que seguir los siguientes pasos:

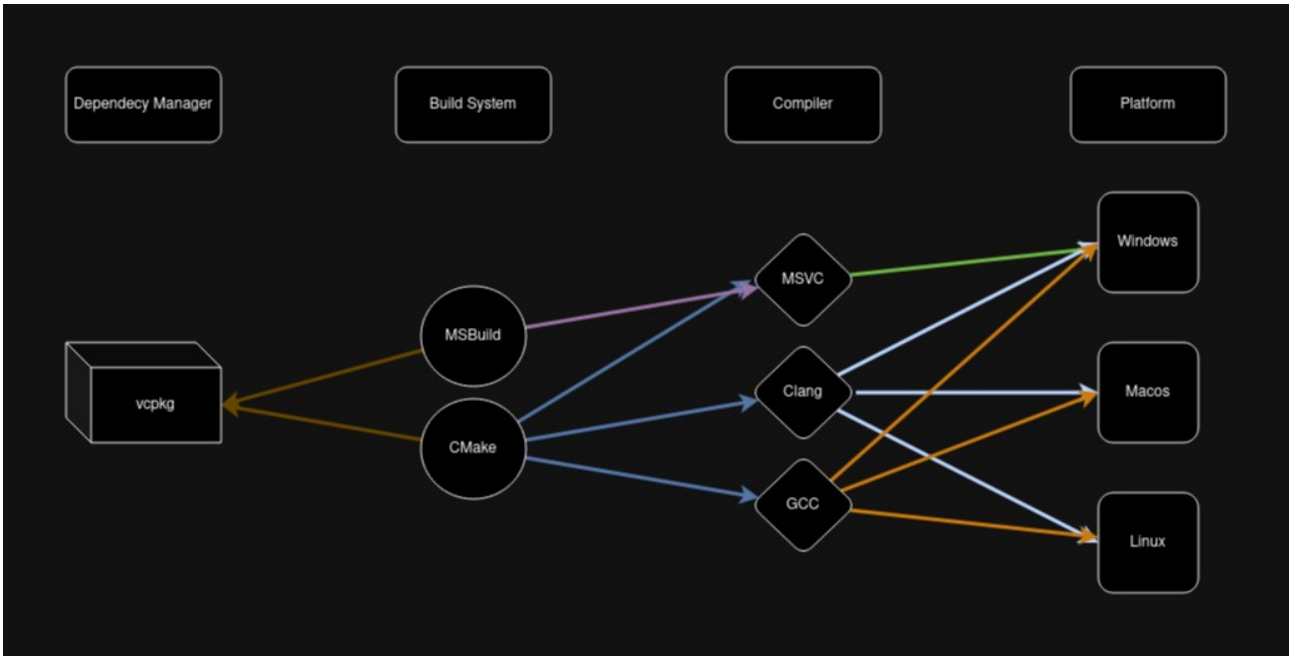
- **Visual Studio 2022**
  - Abrir o crear un proyecto CMake
  - Añadir el vcpkg.json
- **Clion**
  - Abrir o crear un proyecto
  - Abrir la ventana de vcpkg (nativo)
  - Añadir el vcpkg.json en la pentaña usando el "+"
  - Refrescar proyecto CMake
- **Visual Studio Code**

- Tener instalada la extensión de CMake
- Añadir un CMakePresets.json con la toolchain (más info en el pie de página)
- Configurar el preset
- Refrescar el proyecto de CMake

Para más información, consulta https://learn.microsoft.com/ca-es/vcpkg/.

Existen otros Dependecy managers como Conan.io y build2.

## Esquema




## Repositorio de ejemplo:

https://github.com/HugoPlacer/CMake-Vcpkg-Example