بسم الله الرحمن الرحيم

مهسان
تکیه‌گاه شما
در دنیای هوشمند

CMake

An extremely brief to introduction of CMake

mahsan.co

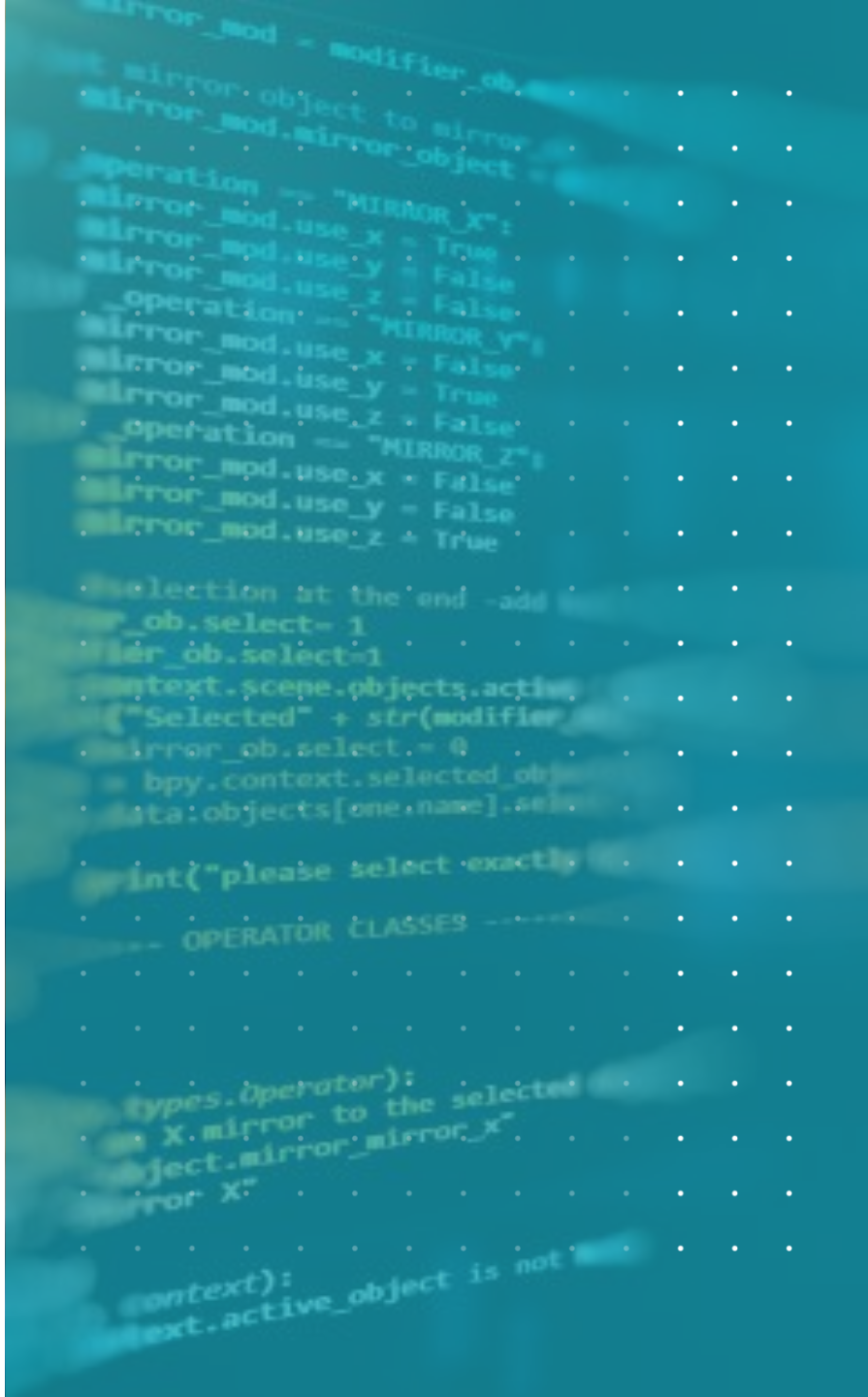
سید سروش
حسین‌علی‌پور

CMake

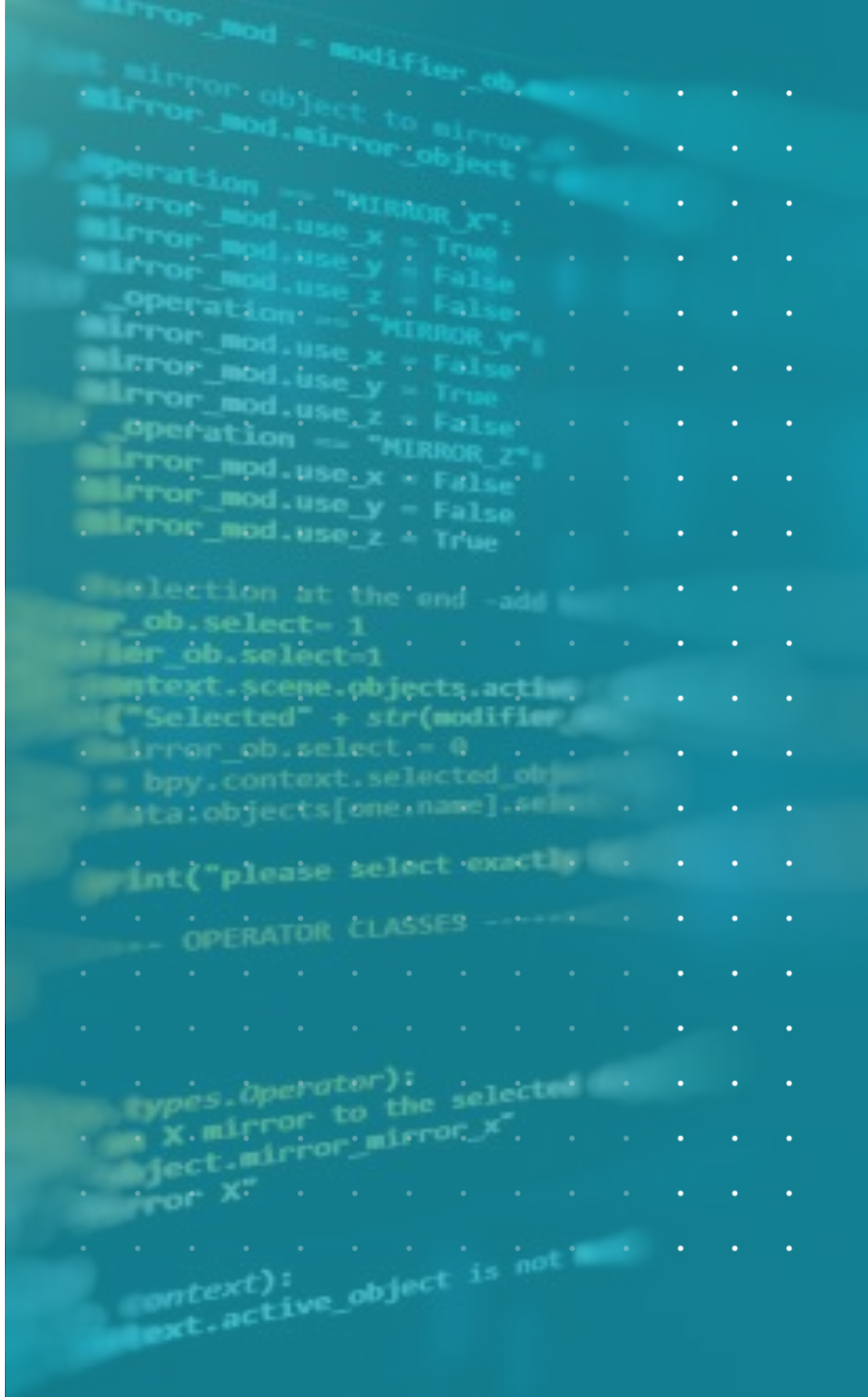
CMake is a collection of open-source and cross-platform tools used to build and distribute software.

History

- CMake development began in ۱۹۹۹
- Bill Hoffman
- Pcmaker Ken Martin
- Brad King

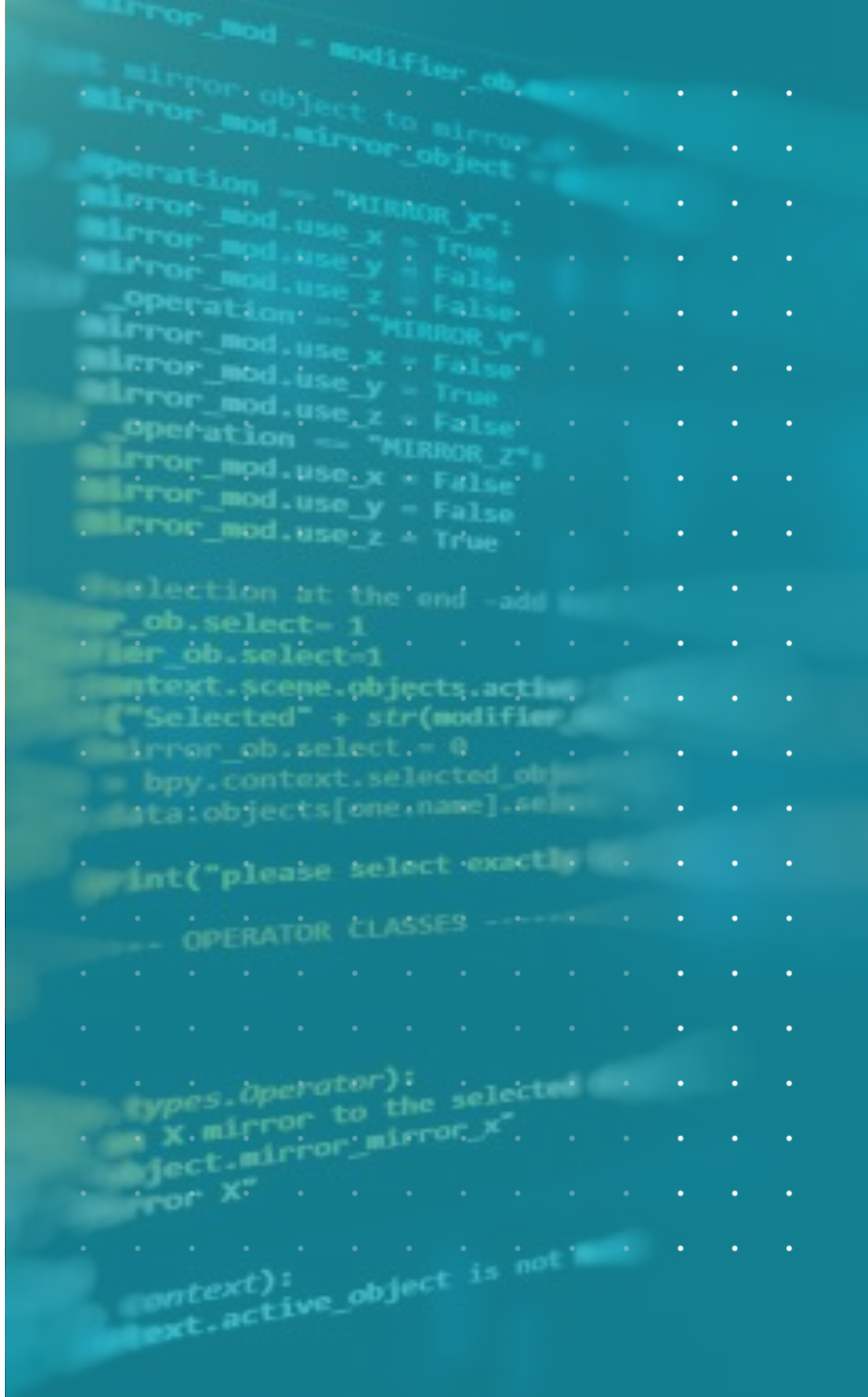


- CMake is known as a meta build system.
- It doesn't actually build your source code
- instead, it generates native project files for the target platform



CMake builds build systems.

- A project based on CMake always contains the CMakeLists.txt file
- **Generators**
- out-of-source build



Understanding the CMakeLists.txt file

- A modern CMake's CMakeLists.txt is a collection of targets and properties.
- A target is a job of the building process. (desired outcome)
- In our example, we want to build the source code into a binary executable: that's a target.
- Targets have properties such as:
  - source files required to compile the executable
  - the compiler options
  - dependencies

In CMake you define targets and then add the necessary properties to them.

```
myApp/
    src/
        engine.hpp
        engine.cpp
        utils.hpp
        utils.cpp
        main.cpp
    CMakeLists.txt
```

Define the CMake version

- A CMakeLists.txt file always starts with the cmake_minimum_required() command

```
cmake_minimum_required(VERSION <version-number>)
```

- Modern CMake starts from version ۳.۰.۰
- We use

```
soroosh@ssoroosh-pc:/mnt/c/Users/ssoroosh$ cmake --version
cmake version 3.16.3

CMake suite maintained and supported by Kitware (kitware.com/cmake).
```

Set the project name

- The second instruction a CMakeLists.txt file must contain is the project name

```
project(myApp
    VERSION 1.0
    DESCRIPTION "A brief CMake experiment"
    LANGUAGES CXX)
```

Define the executable target

- We are about to add our first CMake target: the executable.

```
add_executable(myApp
    src/engine.hpp
    src/engine.cpp
    src/utils.hpp
    src/utils.cpp
    src/main.cpp)
```

Set some target properties

- They are set by a bunch of commands that start with the target_ suffix
- These commands also require you to define the scope
  - how properties should propagate when you include the project into other CMake-based parent projects.
- Since we are working on a binary executable (not a library), nobody will include it anywhere so we can stick to the default scope called PRIVATE

```
target_compile_features(myApp PRIVATE cxx_std_20)
```

Some other properties

```
target_compile_definitions(myApp PRIVATE USE_NEW_AUDIO_ENGINE)
```

```
target_compile_options(myApp PRIVATE -Wall -Wextra -Wpedantic)
```

```
CC = g++
LOADLIBES = -lm
CFLAGS = -Wall -O2


SRC1 = Agent.cpp Breeder.cpp CandidateSolution.cpp \
    Cupid.cpp FateAgent.cpp Grid.cpp Reaper.cpp \
    fitness.cpp

SRC2 = main.cpp

SRC  = $(SRC1) $(SRC2)

OBJS = $(SRC1:.cpp = .o)

AUX = $(SRC1:.c = .h)


main: $(OBJS)
#   $(CC) $(CFLAGS) -o $(SRC) $(AUX)

.PHONY: clean
clean:
    rm -f *.o main
```

add_library

```
add_library(<name> [STATIC | SHARED | MODULE]
            [EXCLUDE_FROM_ALL]
            [source1] [source2 ...])
```

- The CMake variable BUILD_SHARED_LIBS controls whenever to build an static (OFF) or an shared (ON) library

```
add_library(my_shared_lib SHARED lib.cpp) # Builds an shared library
add_library(my_static_lib STATIC lib.cpp) # Builds an static library
```

Running CMake to build the project

Add multiplatform support: Linux, Windows and macOS

```
if (CMAKE_SYSTEM_NAME STREQUAL "Windows")
    target_compile_options(myApp PRIVATE /W4)
elseif (CMAKE_SYSTEM_NAME STREQUAL "Linux")
    target_compile_options(myApp PRIVATE -Wall -Wextra -Wpedantic)
elseif (CMAKE_SYSTEM_NAME STREQUAL "Darwin")
    # other macOS-specific flags for Clang
endif()
```

Passing command line variables to CMake

```
option(<variable> "<help_text>" [value])
```

```
option(USE_NEW_AUDIO_ENGINE "Enable new experimental audio engine" OFF)
```

```
cmake -DUSE_NEW_AUDIO_ENGINE=ON ..
```

```
cmake [options and flags here] <path to CMakeLists.txt>
```

Debug versus release builds

- Debug — debugging information, no optimization;
- Release — no debugging information and full optimization;
- RelWithDebInfo — same as Release, but with debugging information;
- MinSizeRel — a special Release build optimized for size.

- multi-configuration generators
- single-configuration generators

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

Dependency management

the find_library() command

- The idea here is to instruct CMake to search the system for the required library and then link it to the executable if found.

```
find_library(LIBRARY_SDL sdl)
```

- it takes the name of the library to look for and a variable that will be filled with the library path, if found.

```
if (LIBRARY_SDL)
    target_link_libraries(myApp PRIVATE ${LIBRARY_SDL})
else()
    # throw an error or enable compilation without the library
endif()
```

```
# A simple Makefile for compiling small SDL projects

# set the compiler
CC := clang

# set the compiler flags
CFLAGS := `sdl2-config --libs --cflags` -ggdb3 -O0 --std=c99 -Wall -lSDL2_image -lm
# add header files here
HDRS :=

# add source files here
SRCS := #file-name.c

# generate names of object files
OBJS := $(SRCS:.c=.o)

# name of executable
EXEC := #name your executable file

# default recipe
all: $(EXEC)

showfont: showfont.c Makefile
    $(CC) -o $@ $@.c $(CFLAGS) $(LIBS)

glfont: glfont.c Makefile
    $(CC) -o $@ $@.c $(CFLAGS) $(LIBS)

# recipe for building the final executable
$(EXEC): $(OBJS) $(HDRS) Makefile
    $(CC) -o $@ $(OBJS) $(CFLAGS)

# recipe for building object files
#$(OBJS): $(@:.o=.c) $(HDRS) Makefile
#    $(CC) -o $@ $(@:.o=.c) -c $(CFLAGS)

# recipe to clean the workspace
clean:
    rm -f $(EXEC) $(OBJS)

.PHONY: all clean
```

the find_package() command

- You are using special CMake modules that help in finding various well-known libraries and packages.
- Such modules are provided by the library authors
- cmake --help-module-list
- Modules that start with the Find suffix are used by the find_package()

```
find_package(SDL)
```

```
cmake_minimum_required(VERSION 3.10)
project(MyExeProject VERSION 1.0.0)

find_package(SomePackage REQUIRED)
add_executable(MyExe main.cpp)
target_link_libraries(MyExe PRIVATE SomePrefix::LibName)
```

```
cmake_minimum_required(VERSION 3.10)
project(MyExeProject VERSION 1.0.0)

find_package(PNG REQUIRED)

# Add path to a FindSomePackage.cmake file
list(APPEND CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/cmake")
find_package(SomePackage REQUIRED)

add_executable(MyExe main.cpp)
target_link_libraries(MyExe PRIVATE
    PNG::PNG
    SomePrefix::LibName
)
```

the ExternalProject module

- it downloads, builds and prepares the library for use in your CMake project
- By default it assumes the dependency to be a CMake project but you can easily pass custom build instructions if necessary.
- Using this module is about calling the ExternalProject_Add(<name> [<option>...]) command

```
include(ExternalProject) # Needs to be included first
ExternalProject_Add(sdl
    GIT_REPOSITORY https://github.com/SDL-mirror/SDL.git
)
```

- One thing to keep in mind: the download step is performed when you build the project

the FetchContent module (CMake ۳.۱۴+)

- The difference here is that FetchContent downloads the source code in advance while generating the project.
- This lets CMake know that the dependency exists and to treat it as a child project.

```
include(FetchContent) # Needs to be included first
FetchContent_Declare(sdl
  GIT_REPOSITORY https://github.com/SDL-mirror/SDL.git
)
FetchContent_MakeAvailable(sdl)
```

```
project(testProj)

include(FetchContent)

FetchContent_Declare(
  Catch2
  GIT_REPOSITORY "https://github.com/catchorg/Catch2"
  )

FetchContent_GetProperties(Catch2) #mispelled name in original post
if(NOT Catch2_POPULATED)
  FetchContent_Populate(Catch2)
  message(STATUS "Catch source dir: ${catch2_SOURCE_DIR}")
  message(STATUS "Catch binary dir: ${catch2_BINARY_DIR}")
  add_subdirectory(${catch2_SOURCE_DIR} ${catch2_BINARY_DIR}) #can be case insensitive
endif()

add_executable(testExe
  main.cpp
  )

message(STATUS "Catch include dir: ${catch2_SOURCE_DIR}/include")

target_link_libraries(testExe Catch) #name of library to link is case sensitive!
```

از توجه شما
سپاسگزاریم

مهسان
تکیه‌گاه شما
در دنیای هوشمند