ENIGMA

A FREE, OPEN SOURCE & EDUCATIONAL GAME DEVELOPMENT KIT

Google Summer of Code

Improve Build System's Project Proposal

Contents

Contact Information (including discord username)

- Full Name: Saif Salah EL-Deen Yahya Mostafa Kandil
- Primary Email:
- Secondary Email:
- Phone Number:
- Country (City):
- LinkedIn:
- GitHub:
- Discord Username:

Affiliations

Computer and System Engineering, Ain Shams University, Faculty of Engineering.

Title

Software Engineer

Motivation

Build systems like Make, Ninja, Gradle, and MSVS were found to make our life easier because at some point your project files will go off-limits, then you won't be able to track all the dependencies beside this, the compilation process will take much longer as you will need to compile all your files not only files that changed which makes this a time-consuming job.

CMake is a meta-build system or build system generator. The CMake tool can write the Makefile file for us, while in GNU Make this job is on the developer himself.

CMake is a generator of build systems. It can produce *Makefile* files, it can produce Ninja build files, it can produce KDEvelop or Xcode projects, and it can produce Visual Studio solutions.

Building C++ projects is not standard across the platforms. This means that even if you have the source codes and *Makefile* file of a C++ project from your Linux machine, you cannot directly build that code inside Windows OS. Similarly, if you have a Visual Studio solution from Windows machine, you cannot directly compile it in Linux environment. CMake can solve this problem by creating the platform-based build system files by setting some flags.


This means that once you have shared your project codes with anyone that can be built using CMake, you don't need to worry about their development environment. CMake will take care of that, and this is what makes CMake a better choice than GNU Make build system.

CMake itself also provides some nice features like dependency detection, library interface management, or integration with CTest, CDash, and CPack.

Problem Description

Project URL: https://enigma-dev.org/tracker/gsoc.php

About ENIGMA

the Extensible Non-Interpreted Game Maker Augmentation, is an open-source cross-platform game development environment derived from that of the popular software Game Maker. Its intention is to provide you with a quality game creation tool and a bridge between high- and low-level programming languages. It can be used either through an IDE, namely, its sister project, LateralGM, or through a Command-line interface.

Improve The Build System (milestones)

- Replacement of the GNU Make System on your OS:
    1. Understand the current GNU Make system's organization.
    2. Reorganize it to build systems as modules.
    3. Write CMakeLists for the engine and various systems.
    4. Generalize the compiler YAML files to call more generic build commands. (pre-parse, pre-build, build, package, etc).
    5. Add support to ENIGMA's compiler for this change in YAML format.
- Work out any new build system bugs in the CI and other OSes you don't primarily develop on.
- Add packaging routines.
- Add support for interchanging systems at launch.
- Add/refine releases of enigma with pre-built systems.

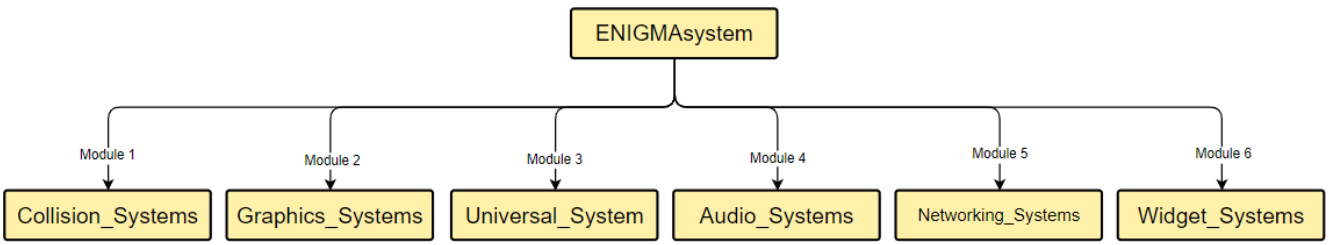
Implementation Plan (My Idea)

CMake makes reorganising build system much easier, so I'm gonna explain more about how this will be done before parsing the milestones into multiple tasks.

First, the project tree structure will be organised:

```
└── enigma-dev
    └── ENIGMAsystem
        └── SHELL
            ├── build # for binary files
            ├── CMakeLists.txt # main CMakeLists.txt file
            ├── *.cpp # SHELLmain.cpp
            ├── *.h # libEGMstd.h
            ├── Networking_Systems
            │   ├── CMakeLists.txt
            │   ├── Asynchronous
            │   │   ├── CMakeLists.txt
            │   │   ├── *.cpp
            │   │   ├── *.h
            │   │   └── test
            │   │       └── ...
            │   ├── *.cpp
            │   ├── *.h
            │   └── test
            │       └── ...
            ├── ... # other systems
            │   ├── CMakeLists.txt
            │   ├── ... # different system modules
            │   │   ├── CMakeLists.txt
            │   │   ├── *.cpp
            │   │   ├── *.h
            │   │   └── test
            │   │       └── ...
            │   ├── *.cpp
            │   ├── *.h
            │   └── test
            │       └── ...
            ├── ... # other directories
            └── test
                ├── CMakeLists.txt
                └── src
                    └── *.cpp # cpp files for testing
```

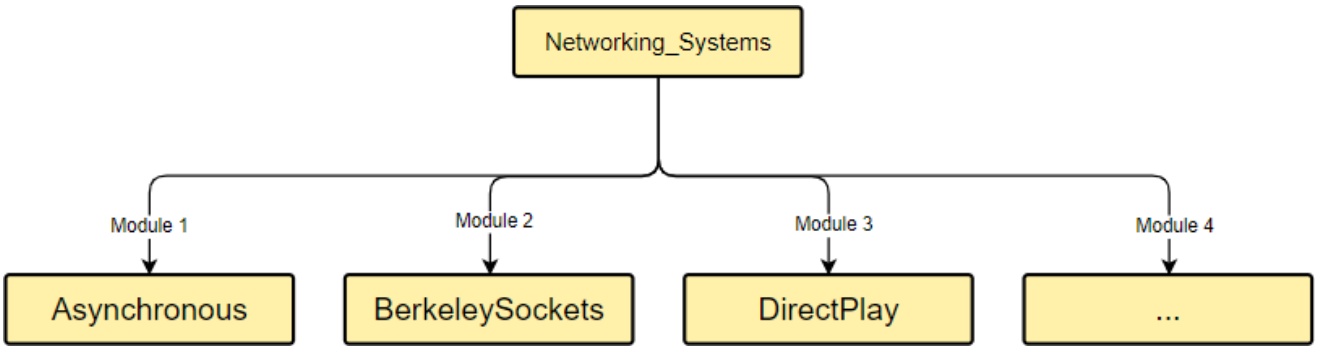
1. The project consists of multiple sub-projects (called build modules).




All these modules are prerequisites of target *ENIGMAsystem*, and may some of these modules have smaller modules, maybe it's not needed to create a *CMakeLists.txt* file for all these smaller modules.

2. Each build module can consist of multiple parts that create either static or dynamic libs, executables, or tests.

Let's look at *Networking_Systems* as an example:



Each of these modules will be linked together to create *Networking_Systems'* target, so I will add these modules as libraries inside *CMakeLists.txt* file:

```
...
add_library(asynchronous
*.cpp)
... # for any other commands like including more directories
```

Then for all other modules:

```
...
add_library(berkeley_sockets
*.cpp)
...
```

```
...
add_library(direct_play
*.cpp)
...
```

```
...
add_library(none
*.cpp)
...
```

Then at the top-level directory of *Networking_Systems*, *CMakeLists.txt* file will be created to link all modules for creating *Networking_Systems* target:

```
cmake_minimum_required(VERSION 3.14)

set(ENGINE_LIB "NETWORKINGsystem")
set(TARGET_DESCRIPTION "Engine sub system")
project(${ENGINE_LIB} DESCRIPTION ${TARGET_DESCRIPTION})
add_subdirectory(Asynchronous) # for CMakeLists.txt file in that dir
add_subdirectory(BerkeleySockets)
add_subdirectory(DirectPlay)
add_subdirectory(None)
target_include_directories(${ENGINE_LIB} PUBLIC General) # for headers in Gen-
eral directory
target_link_libraries(${ENGINE_LIB} asynchronous berkeley_sockets direct_play
None)
...
```

3. Build modules may depend on the public interface of other build modules.
4. For fast build times, each build module should be built on a separate CI-server. The dependencies should not be built each time but fetched from some binary repository.

There are 5 milestones in the project, I will go over them one by one and parse them into multiple tasks:

Replacement of the GNU Make System on your OS

This milestone will include 4 different tasks:

1. Writing *CMakeLists.txt* file for modules (as libraries) of subsystems (if needed).
2. Writing *CMakeLists.txt* file for all various systems in ENIGMAsystem\SHELL\.

We have some variables in the main *CMakeLists.txt* file that calls a separate .mk files from another repository enigma-dev/enigma-android like *android_game* variable, so these .mk files may need to be ported also.

3. Writing *CMakeLists.txt* file for the Engine and gathering all these libraries created together.

```
cmake_minimum_required(VERSION 3.14)

set(TARGET "ENIGMAengine")
set(TARGET_DESCRIPTION "Enigma Engine")
project(${TARGET} DESCRIPTION ${TARGET_DESCRIPTION})
add_subdirectory(Netwoking_Systems) # for CMakeLists.txt file in that dir
add_subdirectory(Graphics_Systems)
...
target_link_libraries(${TARGET} Networking_Systems Graphics_Systems ...)
...
```

4. Generalize the compiler YAML files exists in \Compilers to call more generic build commands (like pre-parse, pre-build, build, package, etc), to make the information stored in YAML files fit any build system, not just GNU build system.
5. Add support to ENIGMA's compiler for this change in YAML format.

Testing

I plan to test the new build system by using a separate *CMakeLists.txt* file with .cpp and .h files as an example by using CMake GUI software to generate *Makefile* file, kindly check the project tree structure above.

Work out any new build system bugs in the CI and other OSes

This milestone will include only 1 task:

1. Fixing any new build system's bug generated from the CI system.

CI used by ENIGMA org:

- Azure Pipelines
- Travis-CI
- AppVeyor

Of course, I'm not relying on the CI system that's why testing will be done during coding.

Add packaging routines


This milestone will include 2 tasks:

1. Using CPack system inside all *CMakeLists.txt* files:

```
cmake_minimum_required(VERSION 3.14)
...
include(CPack)
...
```

When including *CPack* and building the project, two new targets are added to the *Makefile* file *package* and *package_source* and *PACKAGE* in VS and Xcode, so when the user runs CPack commands:

```
cpack -C CPackConfig.cmake
cpack -C CPackSourceConfig.cmake
```

A binary zip, source zip, and NSIS installer (.exe) will be created for the project (the game to be specific).

The table shows the output of CPack commands on different OSes.

Windows	Cygwin	Linux/UNIX	Mac OS X
NSIS	CYGWIN_BINARY	DEB	PACKAGEMAKER
ZIP	SOURCE_CYGWIN	RPM	DRAGANDROP
SOURCE_ZIP		STGZ	BUNDLE
		TBZ2	OSXX11
		TGZ	
		TZ	
		SOURCE_TGZ	
		SOURCE_TZ	

2. Testing *CMakeLists.txt* files for bugs by using a virtual machine setup (I may use docker instance)

Add support for interchanging systems at launch

This will include only 1 task:

1. Add support for interchanging systems (like OpenGL and DirectX) at launch without recompiling.

Add/refine releases of enigma with pre-built systems

Deliverables

- Completely new build system generator which is easier to use.
- Packaging routines for creating installers for users to use.
- Proper documentation and Tests for the above-mentioned build system.

Timeline

Before April 20:

- I know that it's out of working weeks but in this period, I will familiarise myself completely with the current GNU Make system's organization (for a better understanding of how the system works), that's in case I got accepted to start coding without any latency once the coding period starts.
- Study ENIGMA's repo for a better understanding of how the building works and what libraries are being used.
- I'm new to build systems so I will use this period for a better understanding of how CMake works.
- To familiarise myself with the other milestones for the project.
- During this period, I will keep in touch with the community and my mentors to boost the process of understanding GNU Make and CMake build systems.

April 20 – May 20 (Before the official coding time):

During this period, I will be giving 25 hours per week (minimum) for interacting with my mentors and community to discuss my idea and to finalise the brainstorming of the idea with my mentors if they have any suggestions.

- Getting more familiar with the current GNU make build system.
- To familiarise myself more with the dependencies tree of the current GNU Make build system for ENIGMA's engine.

- Revisiting my plan for any error-prone ideas and updating them.
- Get more familiar with all modules created for both subsystems and systems for the engine.
- To do self-coding of *CMakeLists.txt* file to improve my further understanding of CMake build system.
- To get more familiar with the current CI system for testing new build system.
- To get more familiar with all kinds of packages that will be built and how they will be tested (this is for packaging routines).
- To get more familiar with CPack for packaging routines.
- To get more familiar with interchanging systems at launch milestone.
- To get more familiar with ENIGMA by solving issues related to it.
- During this period, I will remain in constant touch with my mentor and the ENIGMA community. I will remain active on the discord channel to discuss and finalise the modifications (if any) that need to be added to my implementation plan.
- Thus, with the help of my mentor, I will become clear about my future goals, and completely understand the current GNU build system to start coding once the official coding period starts.

May 21 – June 10 (Official coding period starts) (Phase 1):

Note that I have final exams starting from 04/06/2022 till 23/06/2022 (not final) (please check my commitments below).

- Porting each module of each system one by one (subsystems).
- Porting sub Makefile files (systems).
- Porting other .mk files that exist in enigma-dev/enigma-android repo (may be needed).
- Porting main Makefile file (enigma-dev\ENIGMAsystem\SHELL\Makefile).
- Generalising the compiler YAML files to make the information fits with any build system like Ninja, Gradle, and MSVS not just GNU Make.
- Add support to ENIGMA's compiler for this change in YAML format.
- Testing the *CMakeLists.txt* file for bugs (during coding).
- Discuss the codebase with mentors for any comments (during coding).


For me, I might expect issues with platform detection while writing *CMakeLists.txt* file as well as issues with sub-Makefile files which hold .cpp, and .h files but I intend to solve these issues by studying other repositories that use CMake as their build system and studying how they handle this platform detection part and by contacting my great mentors for help or ask ENIGMA's community.

Of course, I have no intention to face these issues in the future that's why there's enough period for that before the official coding time starts.

June 11 – June 30:

- Work out any new build system bugs in the CI and other OSes.
- Add packaging routines.
- Testing system for bugs.

JULY 6th MID TERM EVALUATION

July 1 – July 2 (Phase 1 Evaluation)

- Discuss the drawbacks of Phase 1 with the mentor and ask for any improvements.
- Reviewing strategies for the final phase with mentors.

July 3 – July 10 (Final Phase):

- Add/refine releases of enigma with pre-built systems.
- Making further changes in the code to improve the Functionality, Readability, Finding Bugs easily, and Bug Removal.

July 11 – July 18:

- Add support for interchanging systems at launch (if time allows).
- To be in constant touch with the ENIGMA's developers and to let them know about my progress.
- Most of the time will be consumed for rigorous testing and bug fixes, of course, testing will be done during work so maybe this time won't be so large.
- This period might be needed as a backup for any unexpected latency.

July 19 – July 24 (Final Phase Evaluation):

- For Documentation

A buffer of three weeks has been kept for any unpredictable delay.

Additional Commitments

As listed in my Academic Calendar, exam dates start from **04/06/2022** till **23/06/2022** so I will presume that it's final.

Maybe my interaction during this period will be less than usual but I can provide 12 hours per week as a **maximum** for the project and I will keep in touch with my mentors for any updates as well as be online on discord all the time for any important messages from my mentors.

Why This Project?

I'm new to build systems so I chose this project to learn something new and important to improve my skills for future interviews, and I hope I can learn as much as I can in those 12 weeks.

I hope for a great summer this year with the ENIGMA team.

Why Me?

Goal-Orientated Software Engineer, bringing 1 year of experience. Communicating effectively and working tirelessly to achieve impeccable quality standards and results. Expertly skilled in C++, well-versed in high-pressure, deadline-driven, Fast learning, target-focused environments.

My Familiarity with ENIGMA

I'm new to ENIGMA and LateralGM but I'm not new to C++ and Qt so I believe that I can achieve outstanding results.

I plan to extend my knowledge by reading other repositories that use CMake for their build system and how they handle stuff like platform detection, I'll ask the ENIGMA's community for ideas, duplicate existing ideas with my implementation or think of ideas myself.


As a help to the community (myself included), I intend to build a great CMake build system and complete the rest of the milestones perfectly, I will work tirelessly for that.

My Involvement with The ENIGMA Community

As I mentioned above, I'm new to ENIGMA, so my contribution to ENIGMA's community is too small.

I worked with my great mentors a little bit with the current build system, understood more about it and I intend to work more with it to understand stuff like how ENIGMA handles platform detection.

I intend to solve more issues related to ENIGMA before the official coding period.

My Participation in the Google Summer of Code Project with the ENIGMA Project

- I agree to be in contact with my mentor or project administrator at least once a week.
- I agree to attend the meetings, webinars, and other possible online events for students and mentors as many of these as possible given my time zone.
- I agree to comply with the code of conduct of ENIGMA.

GSoC 2022