AdamLibSDL3

Introduction

AdamLib is a game development library written with the intention of allowing fast development with no supplimentary scripting language. An easy interface is provided allowing fast production of reuseable scenes. Built-in cross-compatibility for most desktop, mobile, console, and web platforms is provided.

This engine is not intended for use/installation of beginners to programming or C++, as a moderate amount of knowledge is needed to set-up and use this library properly.

Pre-requisites to use

A C++ compiler, CMake, and git are required to build, compile, and use this library. The MSYS2 Package Manager is required for Windows installation and the apt or yum/dnf packagage managers are required for installation on Linux. An IDE is strongly recommended but not required.

Step 1: Installing dependencies

Currently, the library requires three external libraries to be installed: SDL3, SDL3_image and SDL3_ttf. Different instructions are given for the currently supported platforms, Windows and Linux.

Windows

Ensure you have the MSYS2 package manager installed. Using the command-line interface, enter the following commands to install the required dependencies:

```
pacman -S mingw-w64-x86_64-sdl3
pacman -S mingw-w64-x86_64-sdl3-image
pacman -S mingw-w64-x86_64-sdl3-ttf
```

Once completed, you may proceed directly to Step 2.

Linux

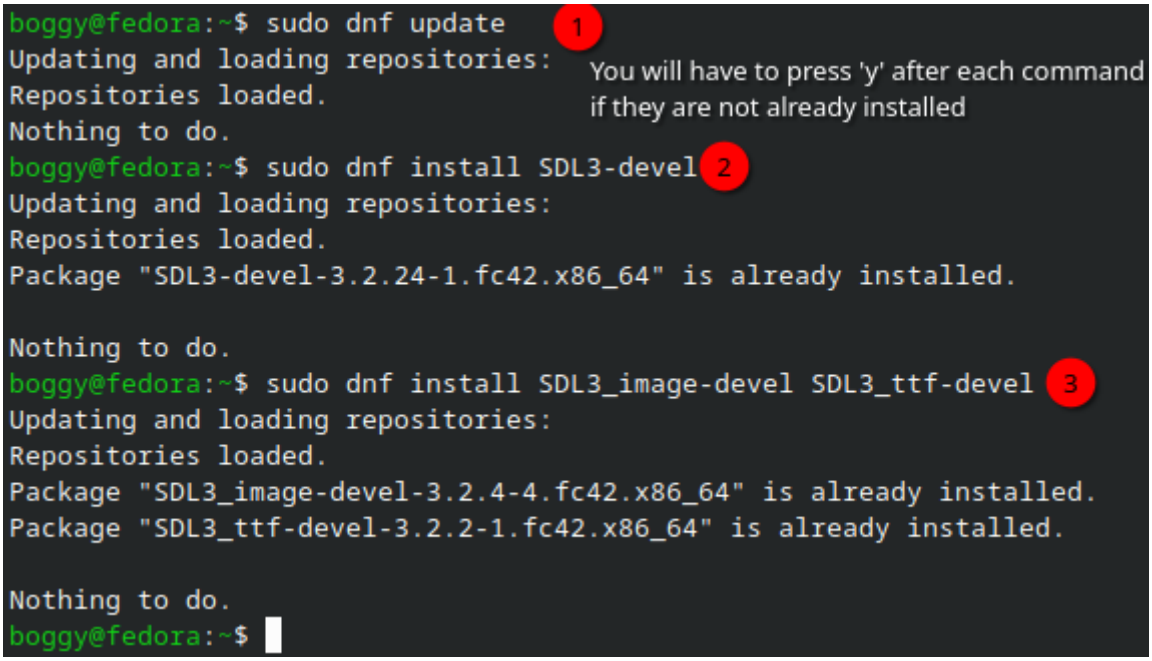
For the apt package manager, type the commands as shown into the command line

```
sudo apt update
sudo apt install libsdl3-dev
sudo apt install libsdl3-image-dev libsdl3-ttf-dev
```

For the yum/dnf package manager, type the commands as shown into the command line

```
sudo dnf update
sudo dnf install SDL3-devel
sudo dnf install SDL3_image-devel SDL3_ttf-devel
```

Example for the dnf manager:



Step 2: Building and Compiling

Find your preferred location to install the AdamLib library. Create a new folder in your desktop or documents folders, and open the command line in that location. Enter the following command:

```
git clone https://github.com/MadamAdamAddem/AdamLibSDL3
```

Then build and compile by typing the following commands into the command line:

```
cd AdamLibSDL3
mkdir build && cd build
cmake ..
make
```

Example output:

```
boggy@fedora:~/Desktop/New Folder$ git clone https://github.com/MadamAdamAddem/AdamLibSDL3
Cloning into 'AdamLibSDL3'...
remote: Enumerating objects: 451, done.
remote: Counting objects: 100% (451/451), done.
remote: Compressing objects: 100% (300/300), done.
remote: Total 451 (delta 223), reused 341 (delta 119), pack-reused 0 (from 0)
Receiving objects: 100% (451/451), 233.83 KiB | 942.00 KiB/s, done.
Resolving deltas: 100% (223/223), done.
boggy@fedora:~/Desktop/New Folder$ cd AdamLibSDL3
boggy@fedora:~/Desktop/New Folder/AdamLibSDL3$ mkdir build && cd build
boggy@fedora:~/Desktop/New Folder/AdamLibSDL3/build$ cmake ..
-- The C compiler identification is GNU 15.2.1
-- The CXX compiler identification is GNU 15.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (0.8s)
-- Generating done (0.0s)
-- Build files have been written to: /home/boggy/Desktop/New Folder/AdamLibSDL3/build
boggy@fedora:~/Desktop/New Folder/AdamLibSDL3/build$ make
```

After some time compiling, the library will be ready for use!

Step 3: Setting up your project with AdamLib

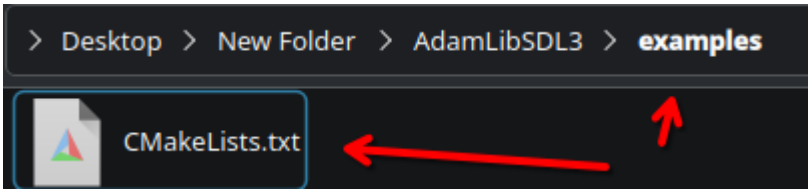
The following steps show how to set up your project within the AdamLib directory. Other set-ups are possible, but are not within the scope of this tutorial.

Configuring CMake

Let **<new_name>** represent the name of your project.
Let **<executable_name>** represent the name of your executable.
Let **<list_of_cpp_files>** represent a space-seperated list of the c++ files you wish to compile.

In the following steps, use the above <> statements to represent your choice
*Ex: If your project name is 'Video Game', replace all instances of **<new_name>** with 'Video Game'*

Locate and open the CMakeLists.txt file found at *AdamLibSDL3/examples/CMakeLists.txt*



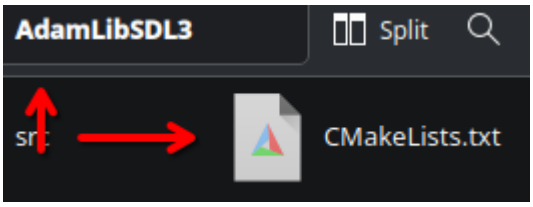
Modify the file such that it looks like this image, with your project's specifics replacing the <representations>.

```
add_executable(<executable_name> <list_of_cpp_files>)

target_include_directories(<executable_name>
  PUBLIC
  ${PROJECT_SOURCE_DIR}/include
)

target_link_libraries(<executable_name>
  PRIVATE
    adam::lib
)
```

Once completed, replace the name of the *examples* folder with <new_name>, then locate and open the CMakeLists.txt file found at *AdamLibSDL3/CMakeLists.txt*



Modify the file such that it looks like this image replacing <new_name>.

```
cmake_minimum_required(VERSION 3.10)
project(adam_engine)

set(CMAKE_CXX_STANDARD 20)

find_package(SDL3 CONFIG REQUIRED)
find_package(SDL3_image CONFIG REQUIRED)
find_package(SDL3_ttf CONFIG REQUIRED)


include_directories(${PROJECT_SOURCE_DIR}/include)

add_subdirectory(src)
add_subdirectory(<new_name>)
```

Configuring your IDE

After configuring the CMake files, open the AdamLibSDL3 folder in your IDE of choice.
Enter the IDE's settings and set the *include/AdamLib* directory to be a registered include directory.
Since there are many different IDE's, no specifics can be provided; there should be instructions available online.

Compiling your project

Once done configuring, simply repeat the Building and Compiling steps mentioned in Step 2.
If you'd like to recompile without rebuilding, type *make* into the build directory.

Step 4: Creating a main file

There should exist three files within what was previously the examples directory:

> main.cpp, test.hpp, and testgame.cpp

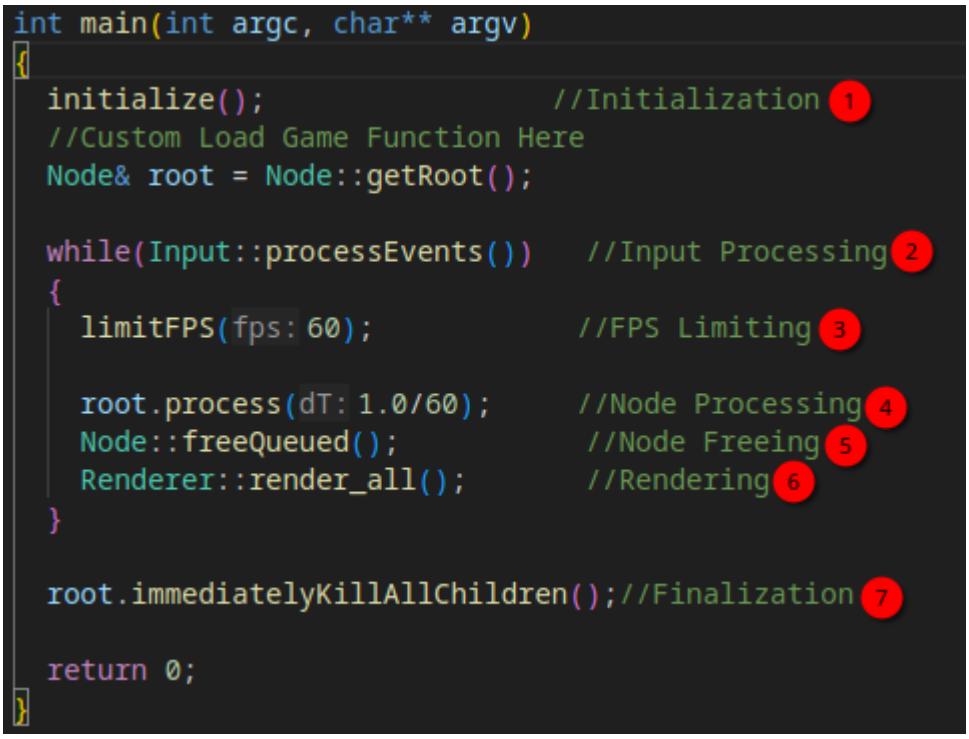
You may read or use these files, but for the sake of this tutorial we will consider them to be deleted.

Setting up the main loop

Create your file that will contain the main function, and define it.
There are seven parts to a main function with AdamLib:

> Initialization (1)
> Input processing (2)
> FPS Limiting (3)
> Node processing (4)
> Node freeing (5)
> Rendering (6)
> Finalization (7)

Each of these parts has a corresponding function. The following image shows how these can be used simply.



Most main loops will look exactly like this, with the game loading function being the only variation point.
Ensure the headers are included like so:

```cpp
#include <AdamLib/Core/AdamLib.hpp>
#include <AdamLib/Core/Rendering.hpp>
#include <AdamLib/Core/Input.hpp>
#include <AdamLib/Nodes/Node.hpp>

using namespace AdamLib;
```

Setting up your game skeleton

AdamLib functions using a **Node Tree**. Put simply, Nodes are the objects that exist within the world.

These objects can have children, which move as the parent object moves, while maintaining their relative position.

To create a Node, first create a **Node Template** to describe the qualities of the node, such as its name, default position, and what child nodes it will have. Then, when ready, create an instance of the node by placing it as a child of the Root node.

Here is an example of just that:

```cpp
NodeTemplate testnode(default_name: "Test_Node");
NodeTemplate childnode(default_name: "Child_Node");

void loadgame()
{
  Node& root = Node::getRoot();
  testnode.default_pos_ = {x: 200,y: 200};
  testnode.registerChildTemplate(child: &childnode);

  root.addChild(node: testnode.createInstance());

}
```

There exist multiple types of Node with different properties. For example, the SpriteNode contains an image which will be displayed on screen at the SpriteNode's position. To create one, use the corresponding NodeTemplate type and add the relative path to your image.

```cpp
NodeTemplate testnode(default_name: "Test_Node");
SpriteNodeTemplate childnode(name: "Child_Node", img_path: "square144.png");

void loadgame()
{
  Node& root = Node::getRoot();
  testnode.default_pos_ = {x: 200,y: 200};
  testnode.registerChildTemplate(child: &childnode);

  root.addChild(node: testnode.createInstance());

}
```

Finishing Remarks

When run, you should see your image display with the top left corner at x: 200, y: 200.

Much more functionality is offered by the library, but this tutorial intends to teach you the basics. Use the intellisense offered by your IDE to scour for functions that may suit your interests, or check out the headers within the */include* Directory for more informaton.