Software

This book goes over the different engines and tools needed for each programming course, and how to use them on Linux.

- Kevgine (Win32)
- Solution-Runner
- Linux IDEs
- SideFX Houdini

Kevgine (Win32)

This is the Win32 engine created by Kevin, with a strong emphasis on its Win32 backend. There are multiple ways to use this engine.

There is a student-maintained version of this framework that has a switchable backend, meaning you can seamlessly switch between Win32 and SDL2. Using the SDL2 backend, the engine can run natively on Linux (and Windows, of course). You can also always switch it to the WIN32 backend to get perfect compatibility, and test it via WINE.

The project is available in the Solution-Runner Templates directory.

> NOTE: The SDL2 backend is provided as best-effort and will not perfectly match the behavior of the Win32 version. Most parts of the engine are faithfully recreated, including the Win32 quirks that come with it, but it is not perfect. For example, sound is not implemented with the SDL2 backend. Please **make sure to always test a Win32 build with WINE (via Solution-Runner) before hand-in!**

WINE

Using LLVM, you can cross-compile the engine as provided to a Windows .exe file, and run it inside of WINE. **You should always do this at least once before hand-in!** The teachers will grade your code based on how it runs for them on Windows!

There are two ways to run your code on WINE:

Solution-Runner (recommended)

Kevgine is fully supported by the Solution-Runner project. With it, you can compile and run the program via WINE using the Win32 backend.

Manually cross-compile (not recommended)

See this in-depth blog post explaining the process for Kevin's engine: Cross Compiling a Windows Game Engine

Natively

To use the SDL2 backend, simply open the Kevgine template from the Solution-Runner repository in a CMake-enabled IDE.

> NOTE: KDevelop is the recommended IDE for C++ on Linux. It has great CMake support and is tested with these projects. Simply open the folder, and press Execute to run it. The first time, you will have to add a Launch Configuration. Simply select `game` from the `Add` dropdown. (TODO: make this a page)

Please make sure you have CMake, fontconfig, and base development tools installed. On Arch Linux, you can use this command:

```
sudo pacman -S base-devel cmake fontconfig
```

The game should run natively on Linux, and you can use any usual development tools like gdb as usual.

Hello world!

Solution-Runner

Solution-Runner is an open source project that allows running projects as Visual Studio "solutions" (`.sln` and `.vcxproj` formats) on Linux, via cross-compilation.

It has been tested with basic Win32 projects, the SDL-enabled Kevgine version (using its unchanged Win32 backend), and console projects.

> ## Source code.

Dependencies:

For Arch Linux, you can use this command to get the necessary packages:

```
sudo pacman -S python clang wine mingw-w64-crt mingw-w64-binutils mingw-w64-gcc mingw-w64-headers mingw-w64-winpthreads
```

Compile and run:

1. Clone the Solution-Runner project to your computer.
2. Navigate to your solution directory, and run the `run.py` script. It will parse the Visual Studio (.sln and .vcxproj) project and attempt to compile it.
3. If it succeeds, it will proceed to run the .exe file via WINE.

> NOTE: you can ignore the usual WINE output, as well as these lines:
>
> ```
> Warning: corrupt .drectve at end of def file
> ```

>

> You may set up a shell alias like `runsln` that points to the absolute path of the script, as a hackish way to "install" it globally. This way, you can call it from anywhere.

Usage with IDEs

This is not a replacement for a proper CMake project with Linux support! However, you can get a functional IDE experience if using clangd as a language server. Solution-Runner will generate a `compile-flags.txt` file, which clangd should pick up. This file allows it to provide errors and diagnostics as if you were on Windows. Magic!

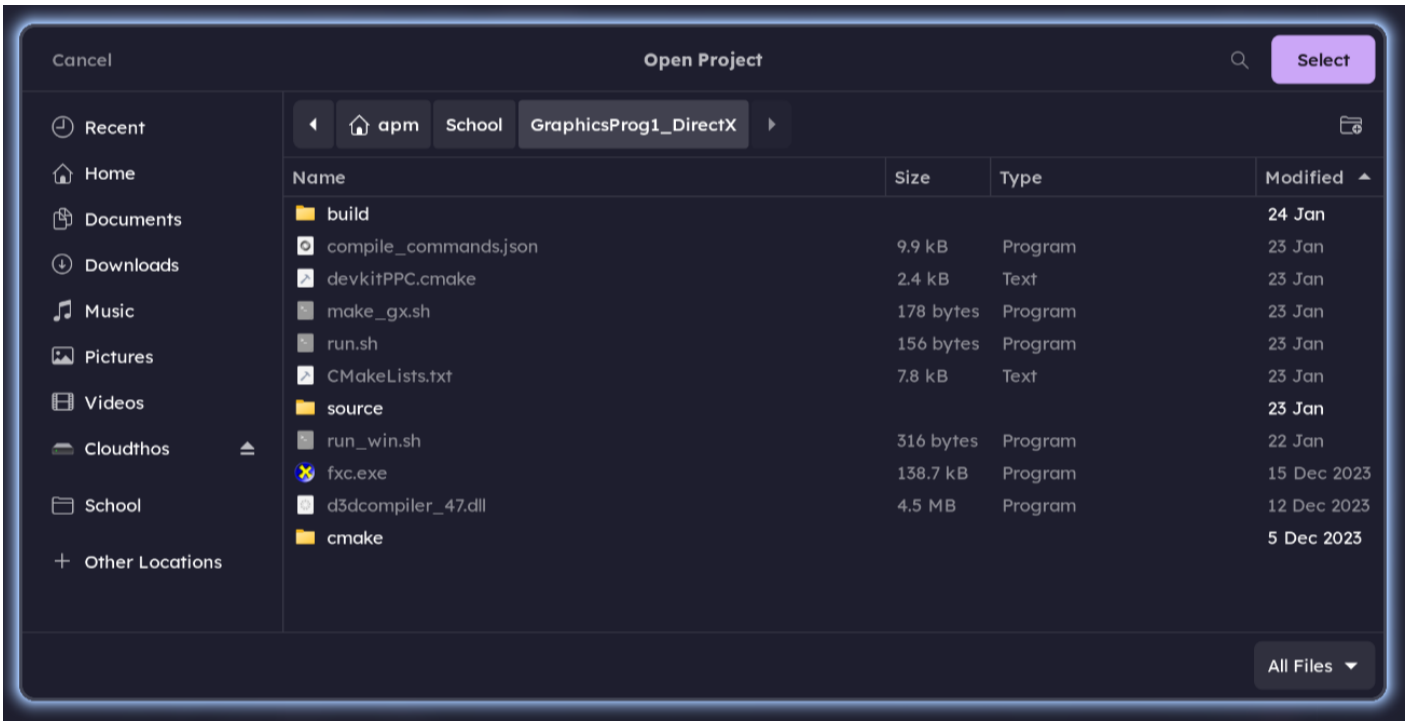
Linux IDEs

Documenting IDEs that work well with DAE Linux engines and tools.

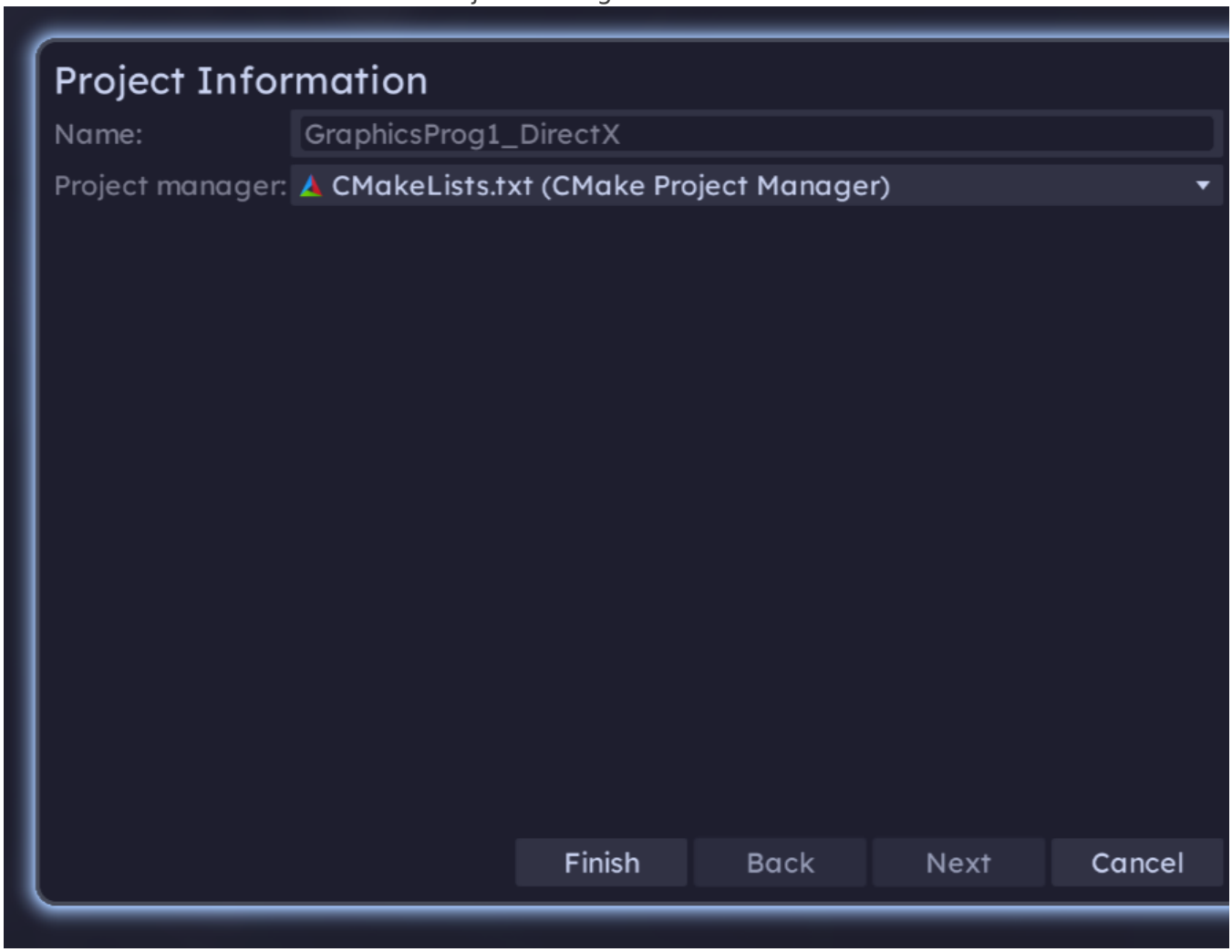
KDevelop

KDevelop is an awesome free and open source IDE for Linux (and Windows + macOS) made in Qt by the KDE team. It provides an experience closest to Visual Studio. Once installed, you can follow these instructions to set it up with any DAE Linux engine:

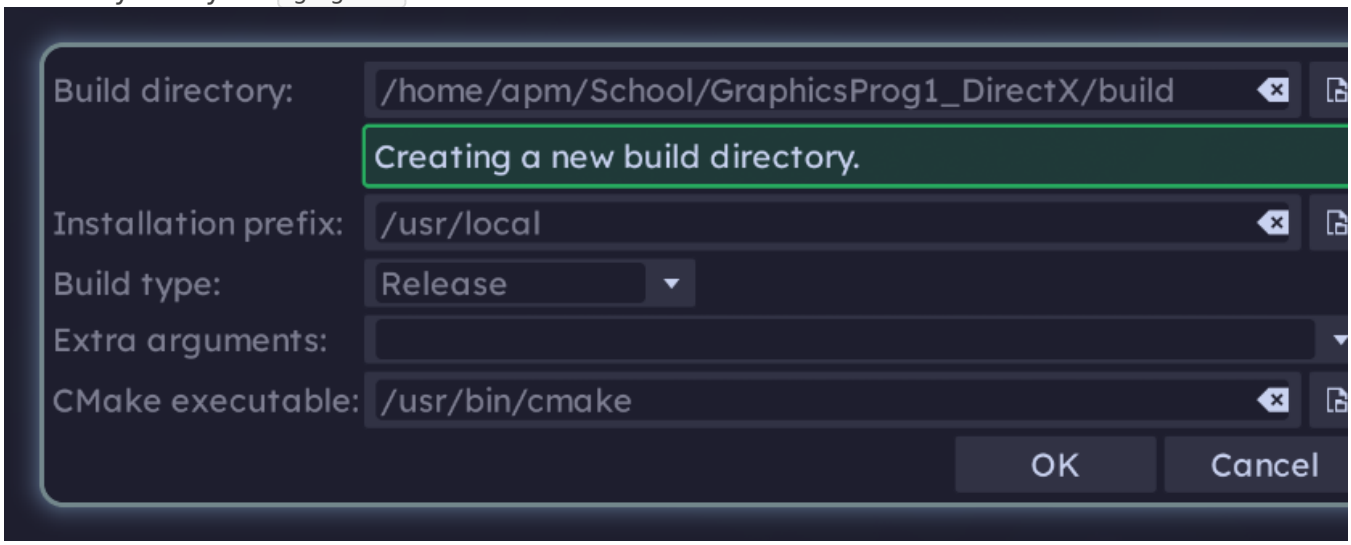
1. Use the `Project` -> `Import Project` button, and select the directory containing `CMakeLists.txt`:



2. Hit finish with the default CMake Project Manager selected:



3. Accept the default settings, ensuring it will use an empty `build` directory. Make sure this directory is in your `.gitignore`!



4. Wait for CMake to run. You can see this in the panel at the bottom. After some time, it should finish successfully:
   The Build window showing successful CMake output. It ends with *** Finished *** and no errors

> NOTE: Some projects may produce CMake warnings, namely older ones using SDL. These can be safely ignored, as long as the configure itself finishes.

5. Now you can hit `Execute` at the top (or `Shift+F9`). The first time, you will be asked to select a launch configuration. Simply click `Add`, then select your target (at the end of the list). For most DAE Linux frameworks, this will be called `game`. You can leave the settings as default: The Launch Configurations list, showing a target called game after being added

SideFX Houdini

Houdini is a nonfree proprietary program developed by SideFX. It allows defining procedural assets via various nodes, in a workflow akin to Blender's geometry nodes.

It is used in DAE's 3D for Games course (fourth semester).

Despite being proprietary, Houdini works very well on Linux, likely thanks to it packaging a semi-recent version of Qt. It does not, however, run natively on Wayland.

Setup

TODO!!