XCode Setup Guide

XCode is an IDE created by Apple to ...

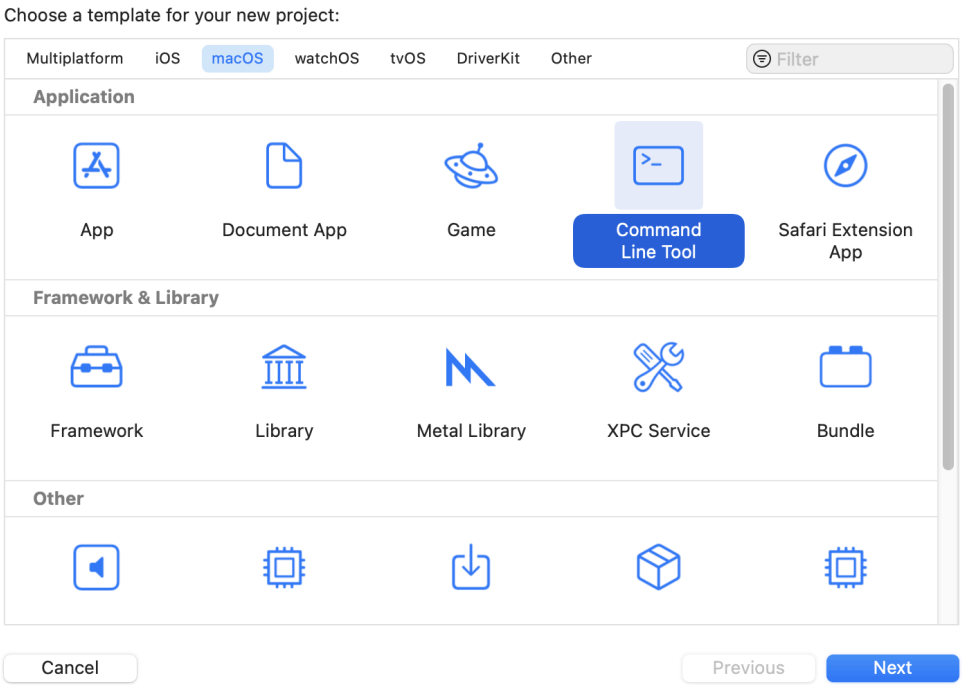
Installing XCode

Visit the apple app store on a mac and perform a search for XCode. If you don’t have a mac, please consider using one in the Groundworks laboratory (1st floor Duderstadt Center across from the VisStudio). Alternatively, ask a friend for temporary use of their mac, or purchase a “cheap” mac mini.

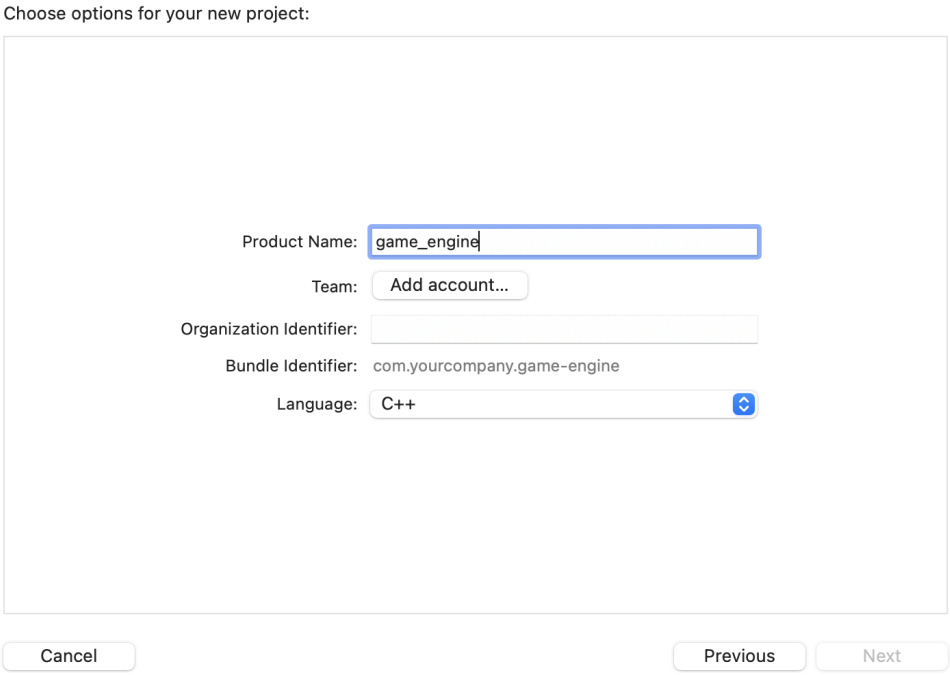
Create a New Project

When you open XCode, you will most likely see the following UI:

Click Create New Project and you will get the following options:

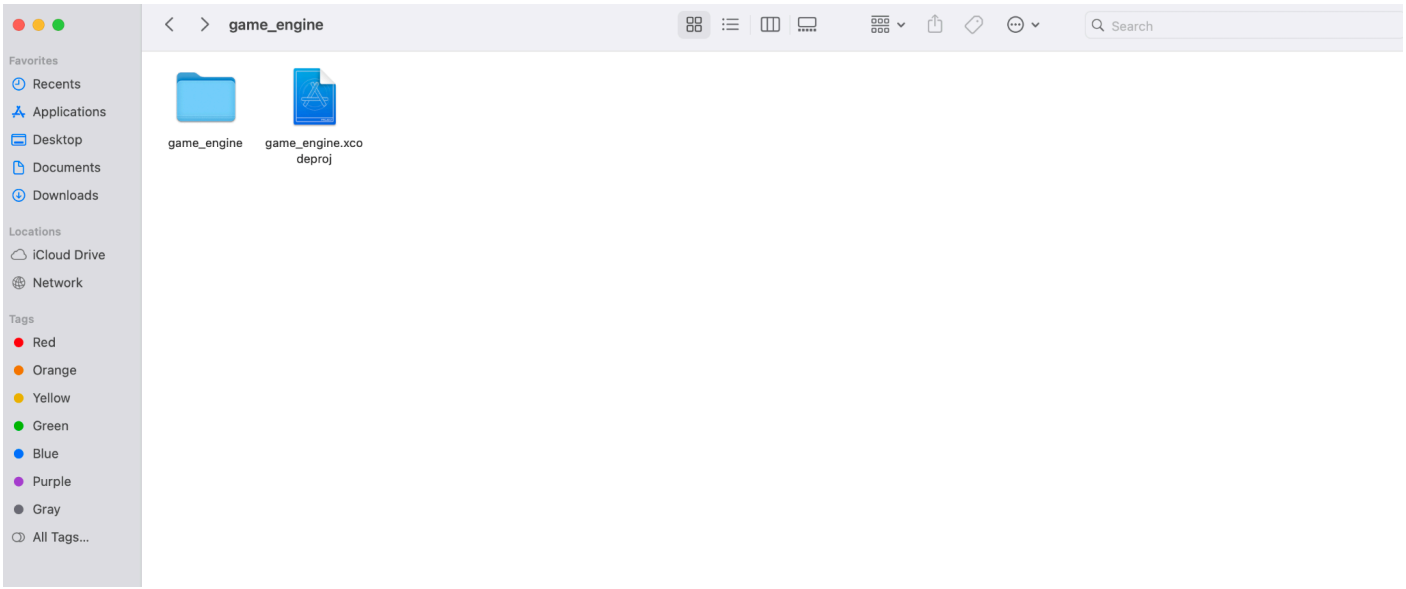


Go to the macOS tab and select Command Line Tool. You will now have to set up the following:



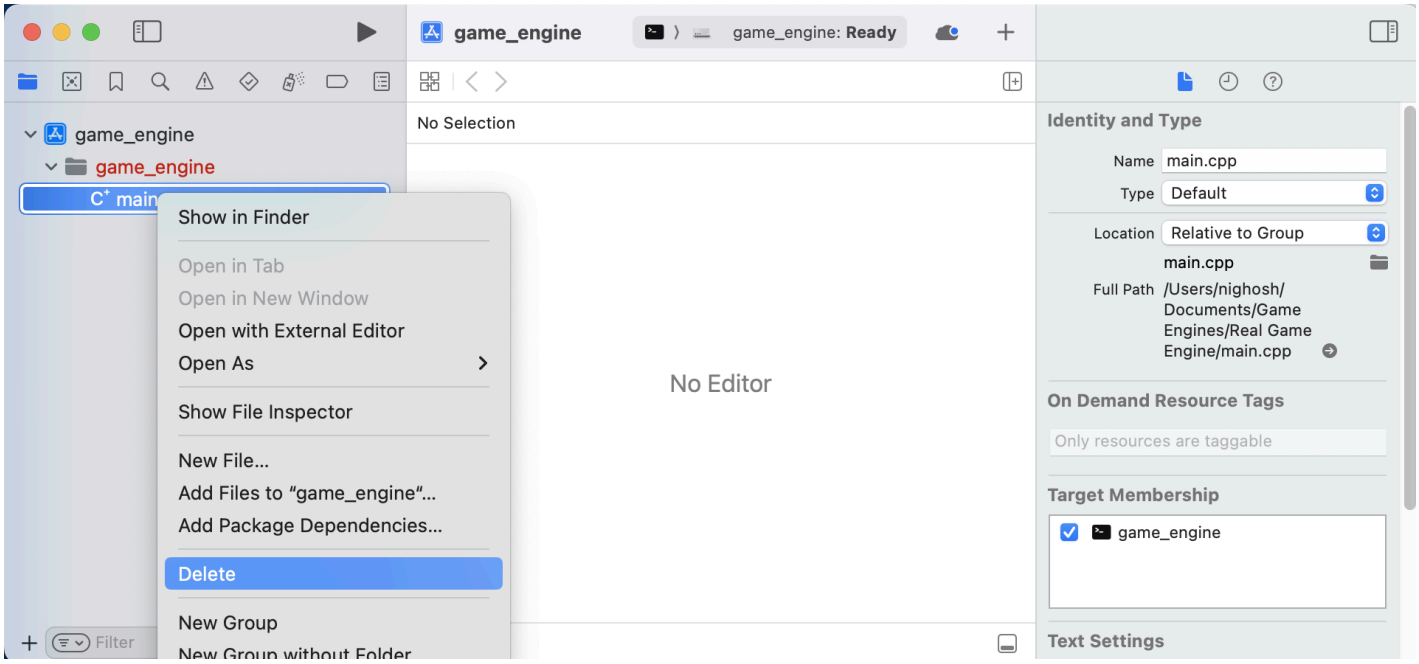
Product Name: Your product name must be game_engine, spelled exactly as shown.
Team: XCode requires you to have a “team” in order to use it. Select your personal team in order to continue. You may need to add an account / sign in with your apple ID during this process.
Organization Identifier: ?
Language: Choose C++

Once you fill out the following, you will be asked to choose a folder (pick any folder, as you will move this later). Uncheck the “Create a git repository” box because you already have one. You will see the following:



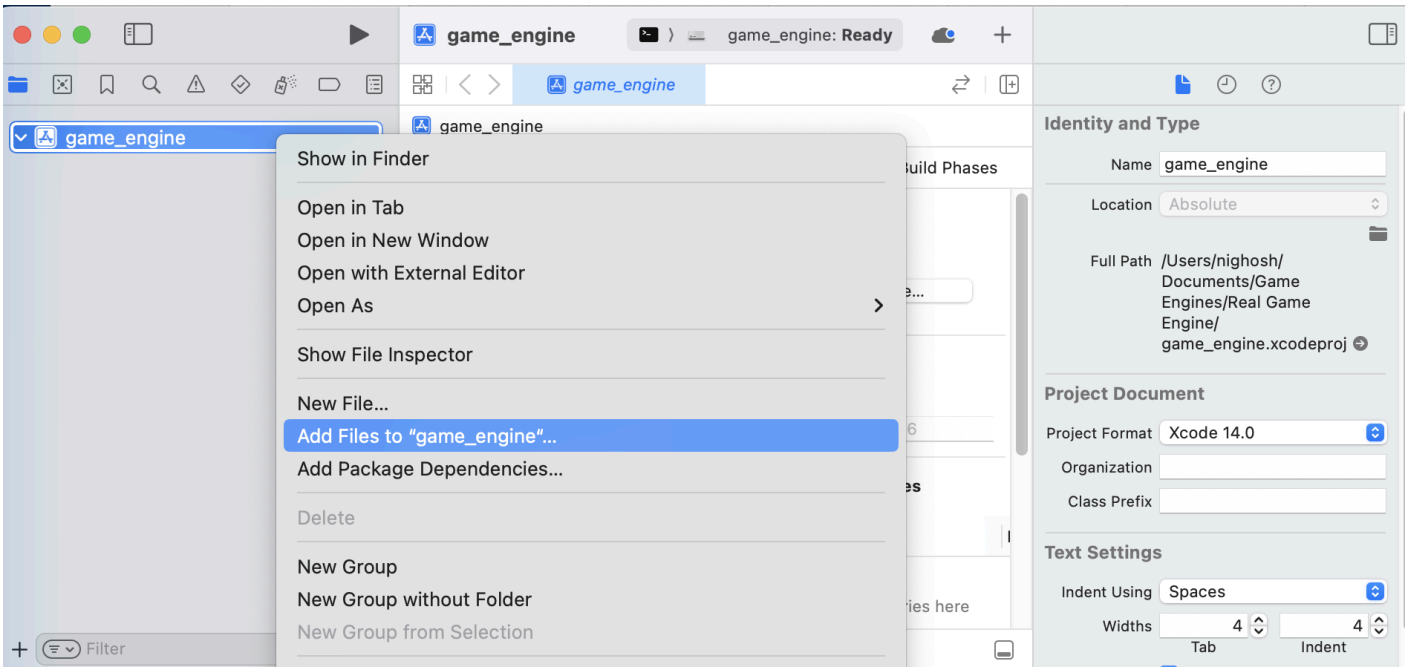
Delete the game_engine folder next to your .xcodeproj file. Move your .xcodeproj from its current location to the root folder of your repository. You can now reopen the project.

Note: After moving your .xcodeproj project and reopening it, you may notice that the files that existed in your previous project are marked in red, as shown below. To fix this, right click on those files and press delete



Adding Existing Files

Throughout the project, you will need to add files that already exist. You will need to do this each time you need to add header files to your project. In addition, if you create files on another platform, you'll need to add them in XCode.

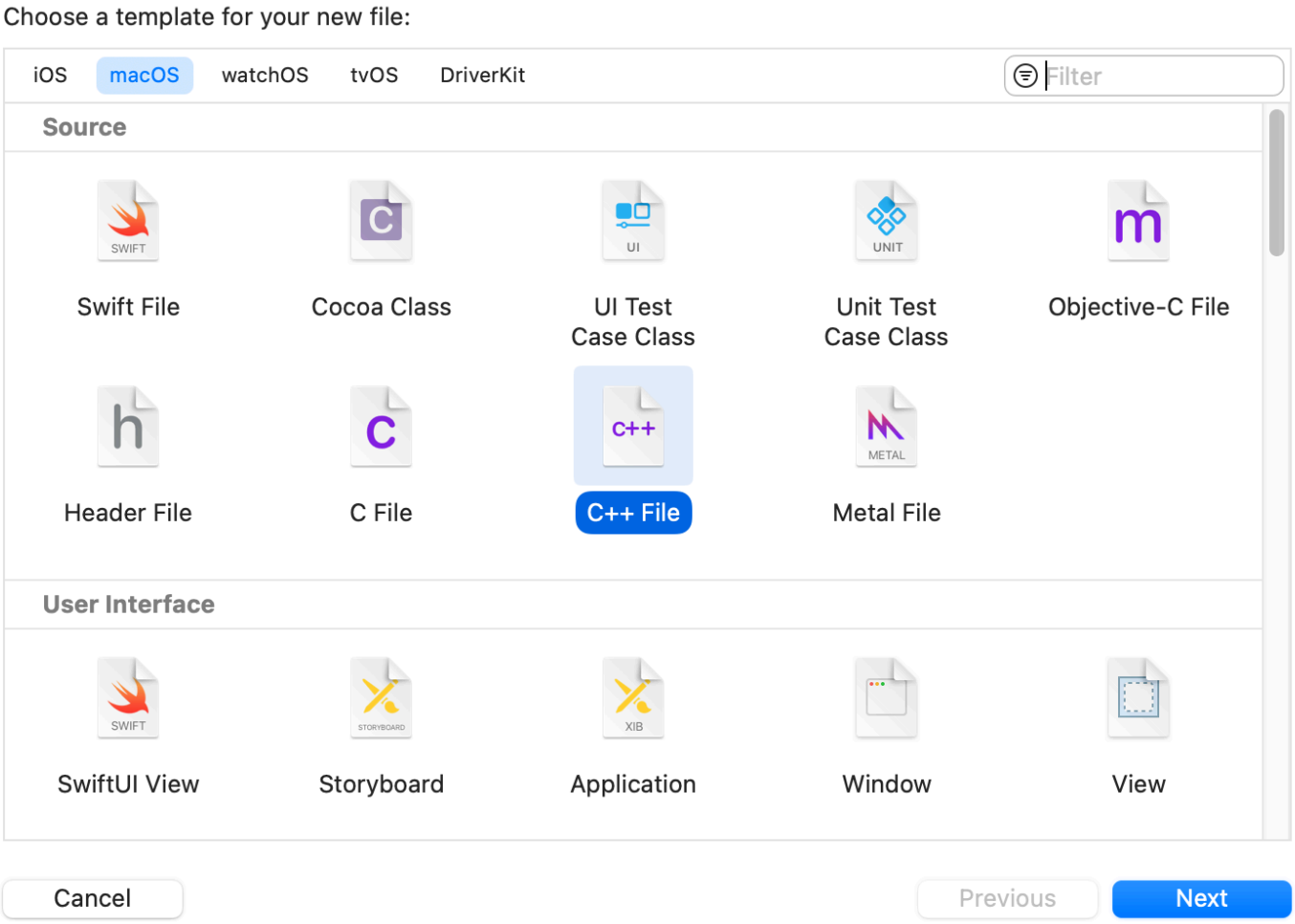


Right click on your game_engine xcode project and click “Add Files to game_engine”, as shown above. Select the existing files and add them to your xcode project.

NOTE: Many students have had an issue where adding an entire existing folder to their project causes XCode to not recognize their code, leading to a cascading set of unintelligible errors. To fix this, delete your added folder and add each file individually (this will work regardless of your file structure).

Add New Files

Right click on your game_engine xcode project and click “New File”, XCode will open up a menu that looks like this:

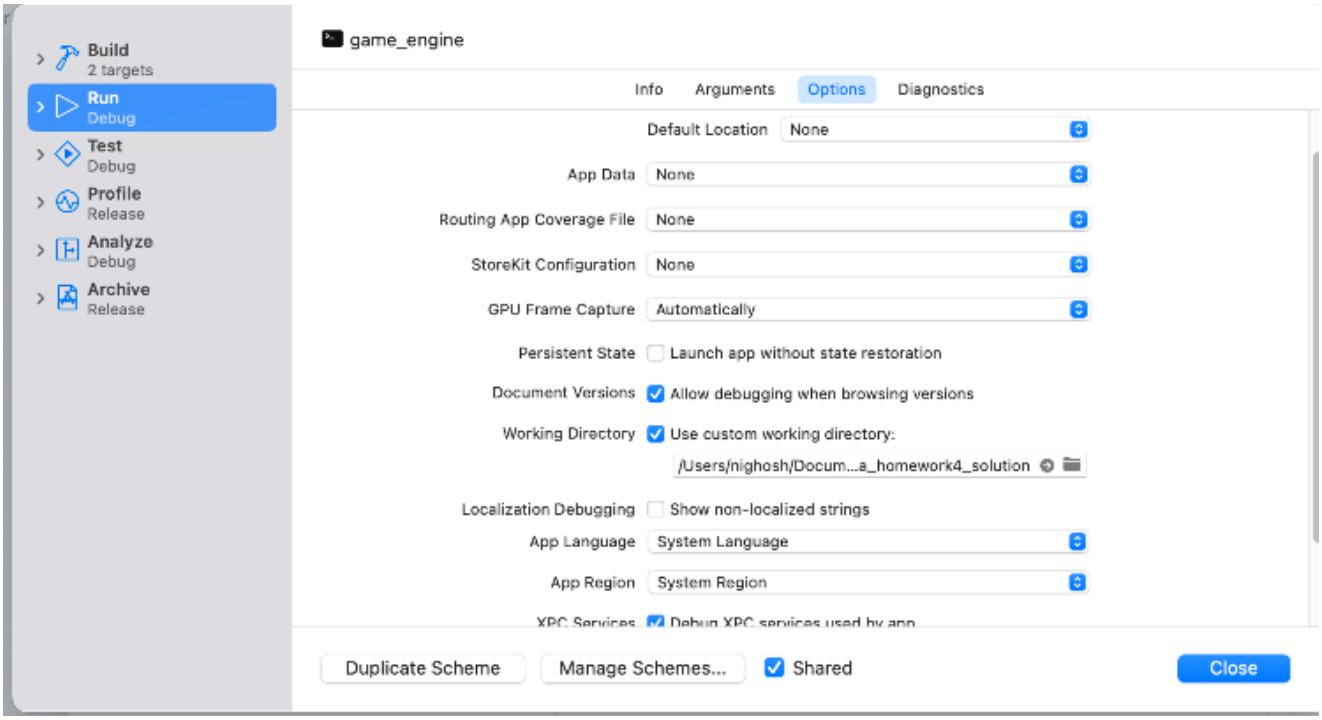


Pick C++ File. Click Next, and then name your file (if you are creating a class, you should select the checkbox that says “Also create a header file”). Click Next, and then choose a location for your new file.

Changing Your Working Directory

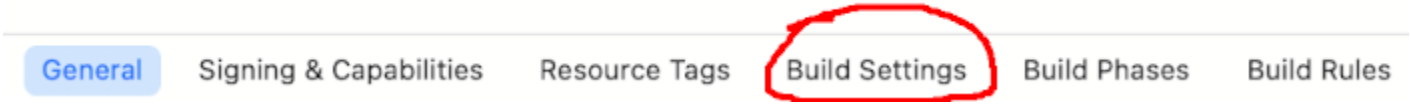
Your working directory is the directory from which the project is run. All filepaths you access by std::filesystem will be relative to your working directory. While the autograder will ensure that it runs your submissions from the correct working directory, you may wish to change it while developing on your mac (this can be relevant if Mac is not your primary development environment, and your project on visual studio uses a different working directory).

To change your working directory, go to Products -> Scheme -> Edit Scheme. Go to the Run tab and the Options tab and you will see a checkbox titled “working directory”. Check the checkbox and set your working directory to a custom path, as shown below:



Adding Additional Include Directories

Double click on your game_engine xcode project in the left bar to open up the project settings. At the top of the tab, you will see the following:

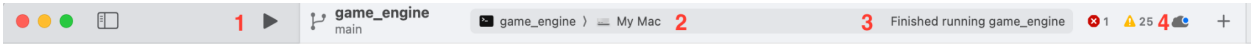


Go to the Build Settings tab and scroll down to see the following (if you are unable to find it, you may need to change your setting to All):

Double click on Header Search Paths, and add the additional include directories you need one by one. Note that you want the **relative** paths of the libraries from the xcode project. Consider using the $(PROJECT_DIR)/ macro to keep paths relative to your repository's root directory.

Running Your Project

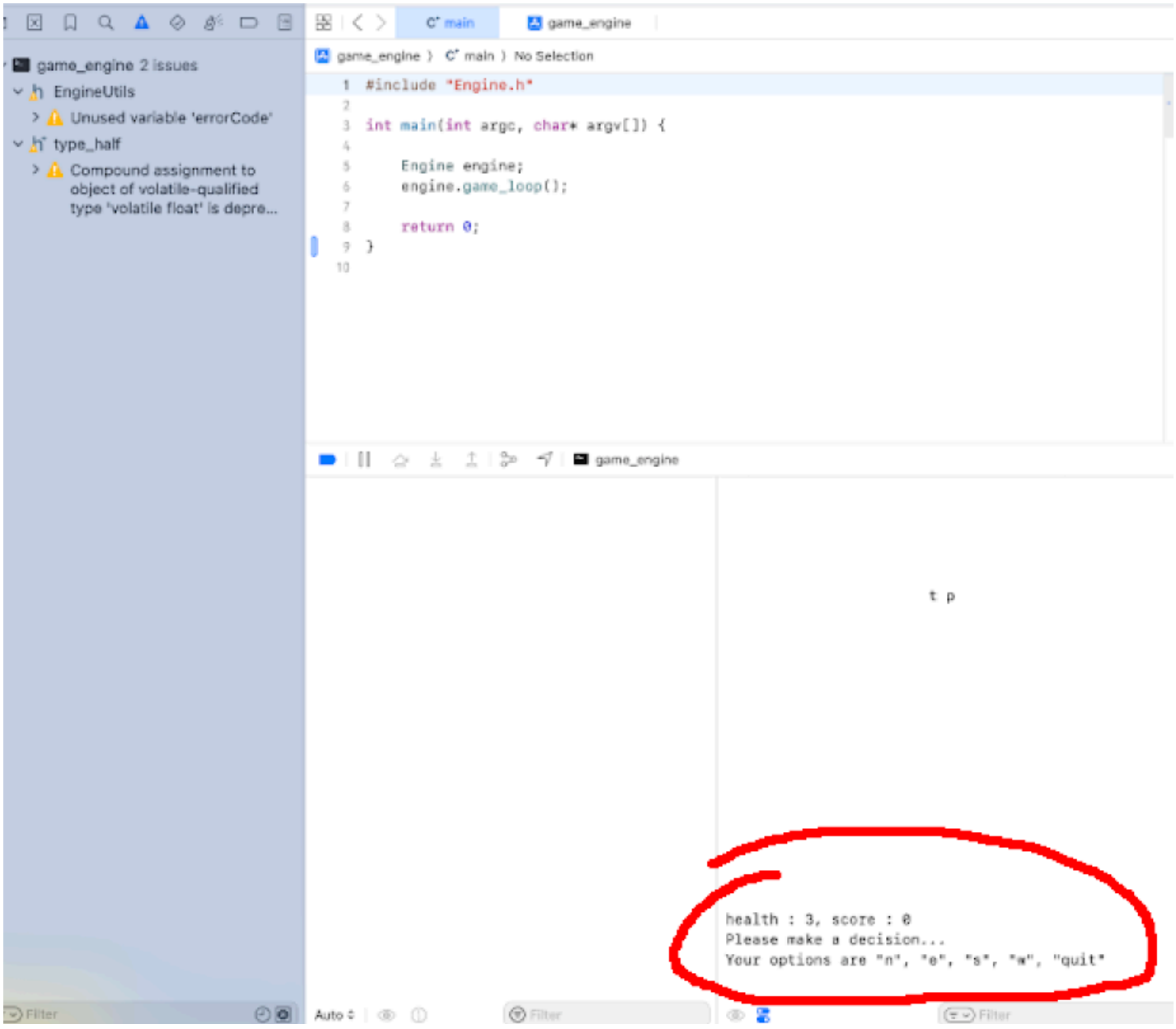
In the top bar, you will see the following:



1. Build Button - Click this button to start building your game engine

2. Project and Destination - This specifies the project you're building and the destination you're building to. If the destination is not "My Mac", you may not be able to see your project. To fix the destination, go to Product -> Destination -> Manage Run Destination.
3. Status - Text that shows you the status of your project. If you are currently running a project, the text will say "running game_engine"
4. Errors and Warnings - This box will show the number of errors (things that are preventing your code from compiling) and warnings (things that XCode does not like but are probably fine). Double click this box to view your errors and warnings.

If your project runs successfully, you will see a console appear in the bottom of your screen, as shown below:

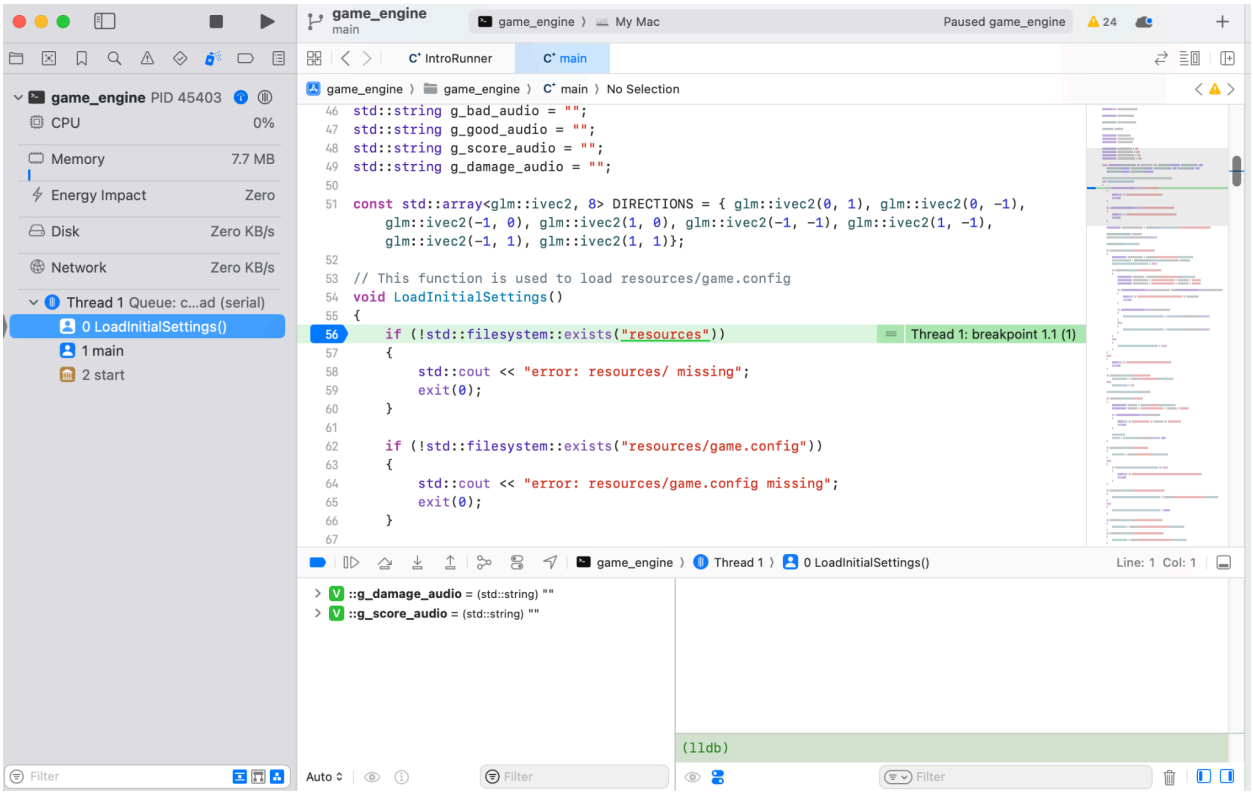


Debugging Your Project

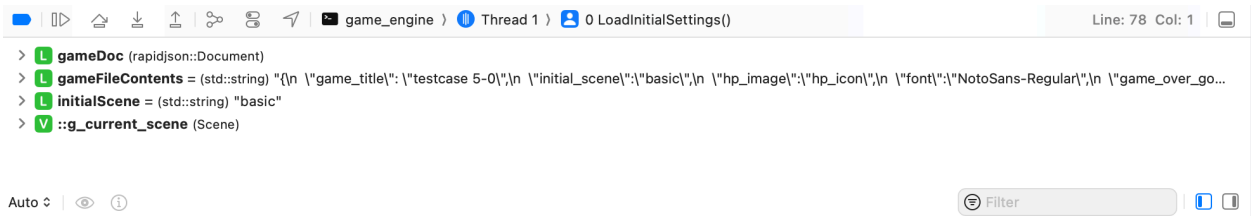
One of the most important skills you'll need in this class is the ability to debug your program. Most students use std::cout to debug their programs in classes like EECS 280 and 281. Although this may work in certain classes, many students find this method to be inadequate for debugging their game engines.

XCode has a debugger, a program that allows you to view the inner workings of your program. The main purpose of a debugger is to enable you to set breakpoints, where your code stops executing at a particular line. When you set a breakpoint, you can view information about the internal workings of the program.

To use the Xcode debugger, ensure that at least one breakpoint is set. To set a breakpoint, click on the leftmost column in your code view, as shown below (in the screenshot below, a breakpoint is set on Line 56.

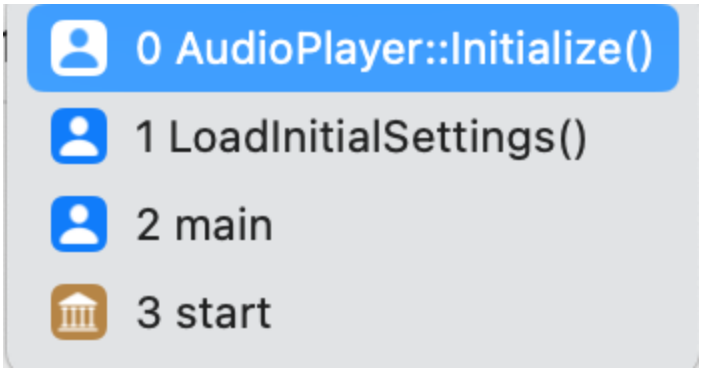


When your program reaches that breakpoint, it will stop. You will see the following tab;



This tab contains a few parts:

- **Variables:** This area contains all the variables at your breakpoint. Clicking on the arrows next to the variable names will allow you to see the member variables inside (useful for debugging complex structures). The mode can be set to Auto, showing relevant variables (such as variables that were changed recently), Local, showing local variables, or All, which shows every viewable variable.
- **Call Stack:** This is viewable by clicking on the current function call at the top of the debugging tab. When clicked on, you will see the call stack (shown below), which shows the active functions being called at this breakpoint. It can be very useful for figuring out where your code is being called.



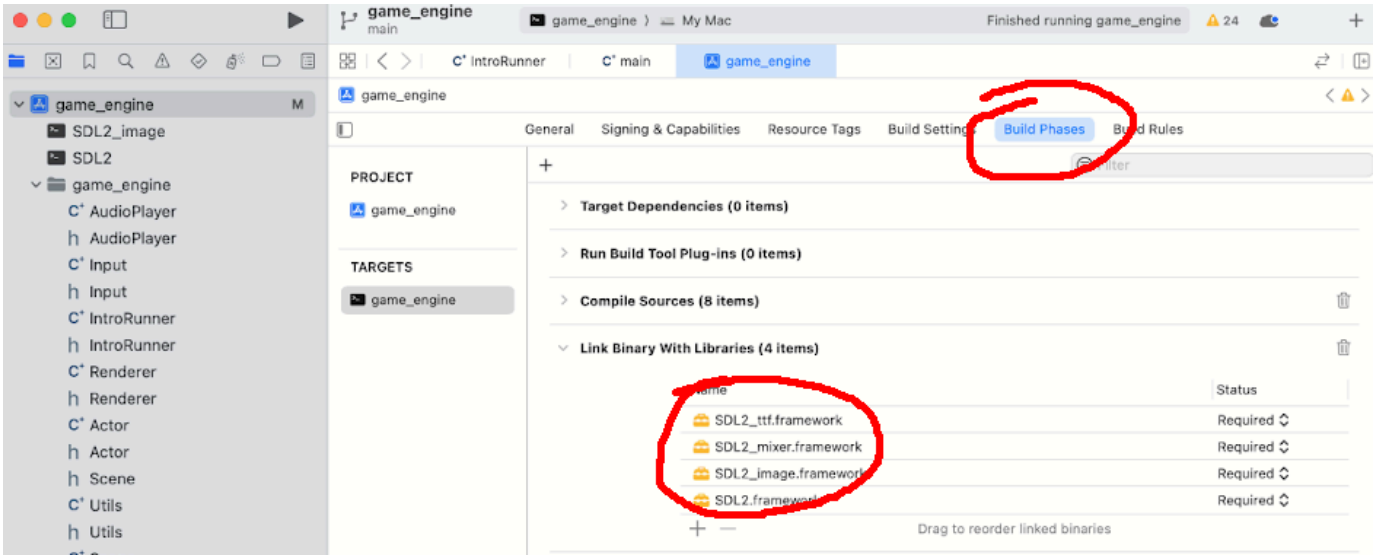
- **Debugging Controls:** The debugging controls are at the top of the tab (shown below), which allows you to step through your code. These controls are (from left to right).

  - **Deactivate Breakpoint:** Set the current breakpoint to be inactive (inactive breakpoints are marked by gray instead of blue)
  - **Continue Program Execution:** Continue running as normal until the next breakpoint hit
  - **Step Over:** Move to the next line in the current file (i.e. do not go into the function call)
  - **Step Into:** Move into the function currently being called
  - **Step Out:** Move out of the current function call and into the caller of this function
  - **Debug Memory Graph:** Not useful for this class (and we have no idea what this does)
  - **Simulate Location:** Not useful for this class (and we have no idea why this exists)

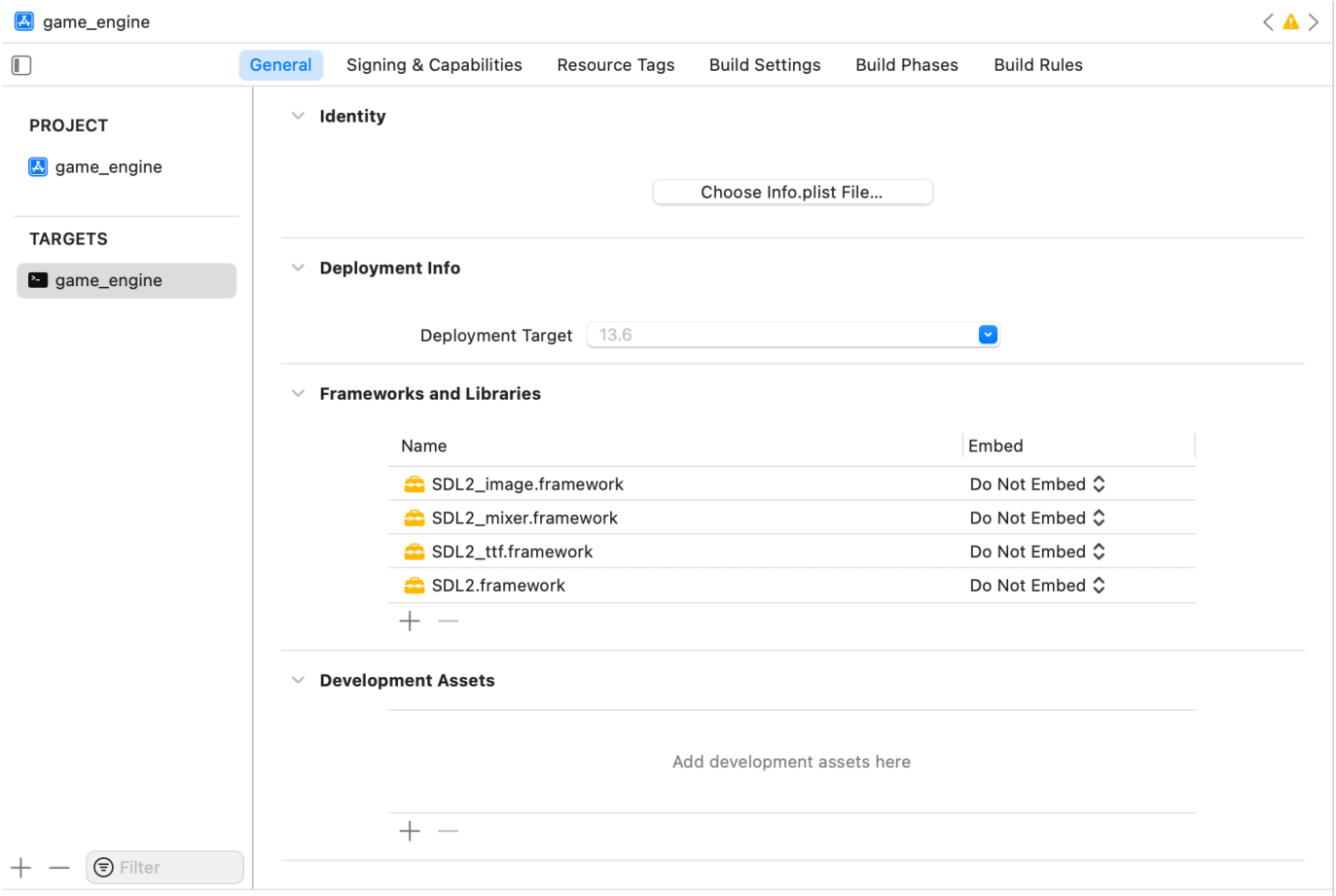
Linking Libraries

NOTE : Are all of your frameworks suddenly failing to build when they used to be working? See the troubleshooting section below!

You may add .frameworks to your project via the “Build Phases” tab--



Please also add them in the General tab--



Setting up Your RPath for Dynamic Libraries

If your engine successfully builds in xcode, but fails once it begins running, this is telltale sign of a dynamic linker problem (known as the “rpath” or “runtime search path” on xcode). Here is an example of what you might see when your rpath fails.

To fix your rpath, you need to go to your project’s Build Settings, find the “runtime search path” property, and add a project-relative line leading to the folder where your desired .framework file exists. This could be something such as `$(PROJECT_DIR)/SDL_image/lib/` but the exact path you need will vary based on how you chose to structure your repo and its various folders.

Choosing C++17

By default, your xcode project may be set to use C++20. As the course and autograders use C++17, you may find yourself struggling to compile on the autograders if you use any C++20-exclusive features.

To make the change, visit your project’s Build Settings and search for “language”--

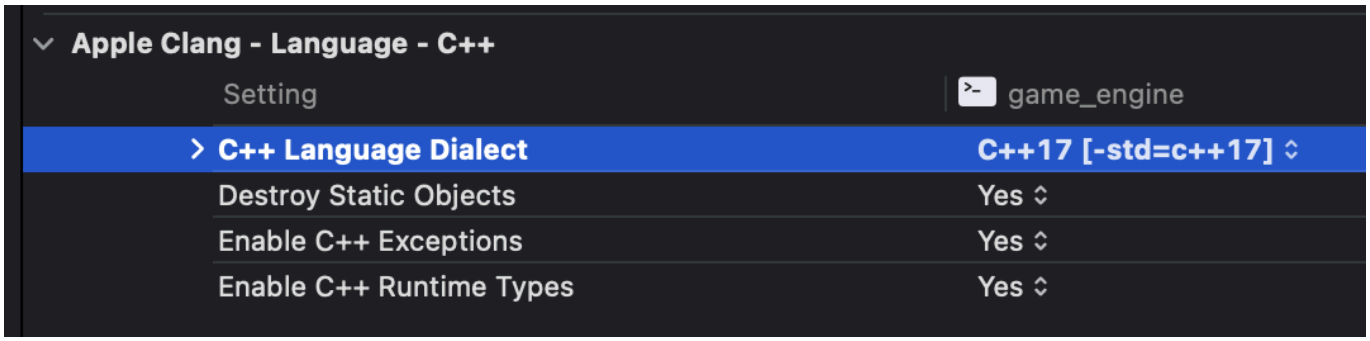
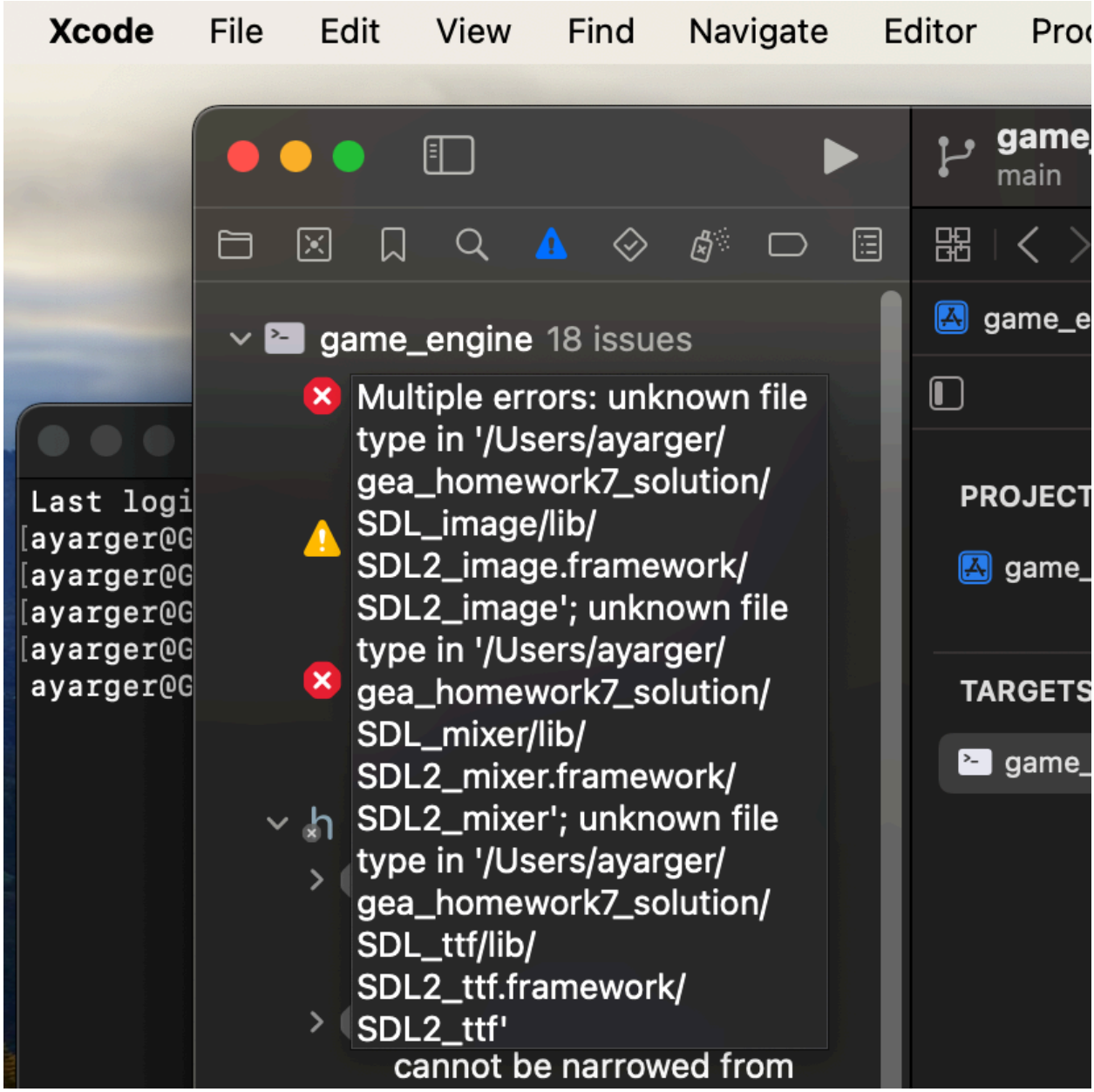


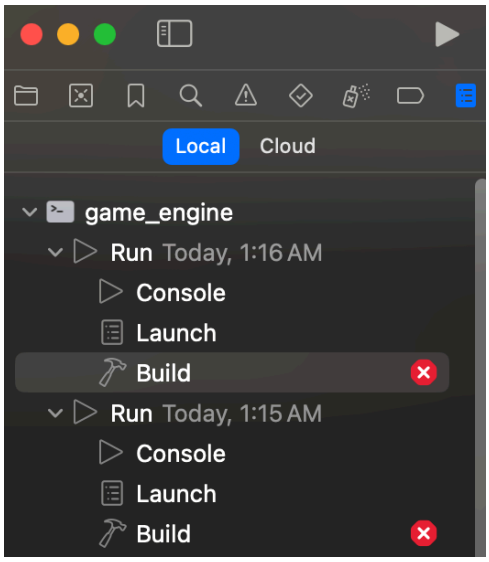
Image courtesy of rahulu

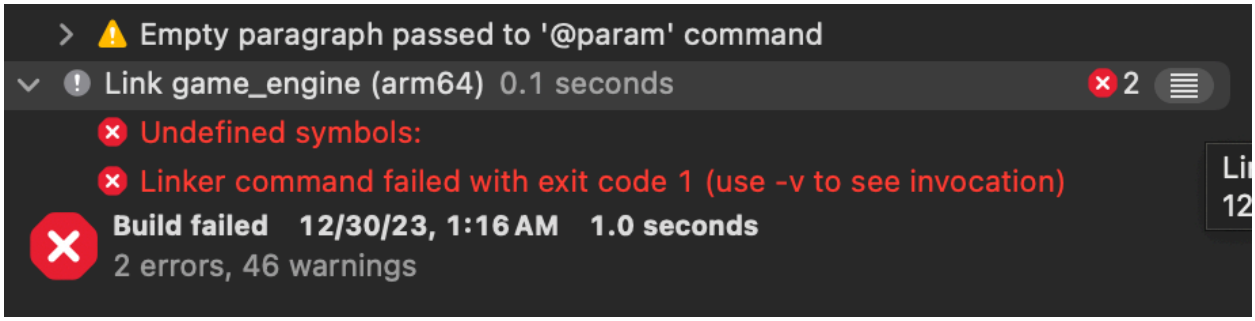
General Troubleshooting

- Are your previously-fine .framework files no longer working in XCode? You might see the error below. If so, re-download the frameworks from github and replace the ones in your repo with them. This seems to fix some weird metadata / filesystem-related corruption that can happen.



- XCode showing weird linker errors without much detail (such as “Undefined Symbols: “)? You can go into the xcode logs to get more information about why the build is failing, and which symbols might be missing. On the left side panel, click the furthest-right icon (in blue below), then click “Build” and then click the small “sandwich” button to the right of the error.





- Having the “The local path is “(null)”. LLDB reports that it doesn’t exist. Check that the executable remains on disk” error? This is one of the unintelligible errors caused by XCode being unable to include your main file in its project. See the “Adding Existing Files” sections for tips on how to add your main file.
- Having trouble getting xcode to load your .framework libraries at runtime? If you see an error like this (“code signature invalid”) after a successful build, please place this script in the top of your repository and launch it via bash in the OSX terminal (`bash fix_frameworks.sh`).
    - The script works by searching your repository for any .framework folders, then signs their inner file, allowing it to be used / loaded on your current machine. It uses the following command–

```
codesign --force --deep --sign - "<library_name>.framework/Versions/A/<library_name>"
```

- Turn off recursive headers
- Enable the console
- Disable