EECS 211 Lab 4

C++ Toolchain Setup

Winter 2019

Today we begin programming in C++ and the GE211 game engine in a minimal example game. The game is quite simple: You control two circles on the screen—one with the mouse and one with the keyboard—and when the two circles overlap, one changes color. As you will see, however, it comes with a bug.

Before we can get started we need to install our C++ and GE211 development environment. You'll need a C++ compiler, the CLion IDE, and the SDL2 graphics libraries. Read on...

Software registration & downloads

First, register for a student JetBrains account on their website. You will receive an email that you need later in this process.

https://www.jetbrains.com/shop/eform/students

For all platforms, you will need to download the CLion installer. Additional downloads vary by platform:

https://www.jetbrains.com/clion/download

Windows You will need to download our custom installer for *MinGW-w64* with SDL2. This comes with C and C++ compilers as well as the SDL2 graphics library.

https://users.eecs.northwestern.edu/~jesse/course/eecs211/files/MinGW-SDL2.exe

Mac You will need to download our custom disk image containing the SDL2 graphics library.

https://users.eecs.northwestern.edu/~jesse/course/eecs211/files/SDL2-all.dmg

Linux Make sure you have a working C++14 toolchain installed. You should also install the development packages for SDL2, SDL2_image, SDL2_ttf, and SDL2_mixer.

Toolchain setup

Windows

On Windows, you need to install *MinGW-w64* (the C++ compiler):

1. Run the MinGW-SDL2.exe installer and follow the prompts to install *MinGW-w64*. You should usually install it to C:\MinGW, but wherever you install it, take note, as you will have to configure CLion to find it later.

2. Follow the instructions in your JetBrains registration email to activate your account.

3. Run the CLion installer. Most defaults should be fine, but you should check all of the "Create associations" boxes when they appear.

   Set the toolchain in CLion to the location where you installed *MinGW*. The folder you select should contain subfolders with names like bin and lib. Ignore any warnings about version numbers.

Mac

Mac OS automatically installs its toolchain when you attempt to use it from the command line for the first time; you will still have to install the SDL2 libraries yourself.

1. Thus, to install developer tools, run the Terminal.app program (from /Applications/Utilities) to get a command prompt. At the prompt, type

   ```
   $ clang
   ```

   and press return. If it prints `clang: error: no input files` then you have it installed already. Otherwise, a dialog box will pop up and offer to install the command-line developer tools for you. Say yes.

   (Alternatively, you can install the latest version of Command Line Tools for OS X manually from Apple, or install XCode from the App Store.)

   https://developer.apple.com/downloads/

2. Once you have the developer tools installed, you need to install the SDL2 libraries. Open the SDL2-all.dmg disk image and drag the four frameworks into the linked /Library/Frameworks directory. You may have to authenticate as an administrator.

3. Follow the instructions in your JetBrains registration email to activate your account.

4. Run the CLion installer—defaults should be fine.

The game

Getting the starter code

For this lab, the starter code is provided as a ZIP file here: http://users.eecs.northwestern.edu/~jesse/course/eecs211/lab/eecs211-lab04.zip. Extract the archive file into a directory in the location of your choosing. Once you have your new directory containing the starter files, you can open it in CLion.

Be careful, as CLion will only work correctly if you open the *main project directory* with the CMakeLists.txt in it. If you open any other directory, CLion may create a CMakeLists.txt for you, but it won't work properly.

Inverted control

Currently, there is an bug in this code. Run the program and try to control the circle with your left and right arrow keys, and you will move in the opposite direction of what you intend. Locate the code for this—hint: it's in the model—and fix it.

There are test cases for checking the model's movement, so when you are done try running your code against the tests.

Up and down

As you have seen, the circle that is controlled by the keyboard only moves horizontally right now. Add two member functions to the `Model` struct, `Model::move_large_circle_up()` and `Model::move_large_circle_down()`, and connect them to the keyboard by modifying the `Game::on_key(Key)` member function in game.cpp.

Click, not hover

Currently, the position of the smaller circle tracks the position of the mouse. However, what if we want the game to only update the position of smaller circle when we click? To detect mouse clicks, you will have to override the `Abstract_game::on_mouse_down(Mouse_button, Position)` function in the `Game` struct. See the documentation here.

Testing

The current tests include a few examples that should pass for your code. Add two new test cases for the new functions you added to the model, and verify that they do what you expect.

There is a test case that checks that the state will sometimes be `Collision_state::touching`. Fill in the final test for checking when the circles aren't touching.