ICS 161 – Winter, 2016 – Homework 2

Follow the steps below. Start in your Wednesday, Jan. 13 lab session, and continue on your own time. As specified below, take several screen shots and code fragments that show your progress and paste them (in order) into a single Word document. Upload that Word doc to the EEE Dropbox "ICS161-Hw2" before 11:55pm on Wednesday, Jan. 20. You will also upload a ".h" file to the same Dropbox, with the same due date but an earlier due time: 1:00pm.

- You can take a screen shot of the active window by holding down the Alt key and pressing the PrtScn key. This puts an image in the Windows clipboard, which you can paste into a Word document with Ctrl-V. Sometimes Alt-PrtScn does not work. If not, use plain PrtScn instead with no maximized windows, and scale the copied image to fit on a single Word page. With plain PrtScn you may need to make sure that the focus is on the program you are running, by clicking on the command prompt window.
- The desktops in the COGS lab are not shared. Files you store on the desktop or in My Documents will not be available on other computers, and may disappear from the computer you created them on. You should always copy your files to a USB stick or a network drive.

Lesson 4 – Handling Events

1. Lesson 4 is at http://www.willusher.io/sdl2%20tutorials/2013/08/20/lesson-4-handling-events/. This lesson is based on lesson 3, but unfortunately Visual Studio does not provide a straightforward way to copy a solution and all of its files and folders. Instead, create a Lesson4 project as you did for the last few lessons, based on SDL_Template and renaming the source code file. Then copy and paste your Lesson3.cpp code into Lesson4.cpp.

2. Download the lesson's image.png to res\Lesson4. Also copy background.png into this folder.

3. In your source code, change "Lesson3" to "Lesson4" in two places. Run the code to make sure everything is working OK. As usual, you'll need to copy in the four .dll files – you can conveniently find them in the Lesson3 Debug folder.

4. Do you see the background image? Why not? If you aren't sure, comment out the line that renders the "image" texture and run the program again. Does that provide a good enough clue? Since the background doesn't display, it's a good idea to remove from Lesson4 the code that handles it.

5. Read through the rest of the lesson before you copy anything into your source code file. You'll see that the lesson's third code block is a combination of the first two. Copy this third code block into your program's main(), replacing three source code lines that are there from lesson 3 (hint: one of these three replaced lines calls SDL_Delay()). Rebuild and run the program.

6. Let's follow the spirit of the lesson's "Extra fun" section, and add two features to the program:

   a. Decrement x each time through the outer while loop. Build and run and observe the results.

   b. Increment x each time the user presses the right arrow key. You'll need to add a test for `if (e.key.keysym.sym == SDLK_RIGHT)` inside the test for SDL_KEYDOWN. Make sure if the right arrow is pressed that this *doesn't* cause the program to quit. You will probably find that incrementing x by 1 isn't enough to get the expected results; if so, try incrementing x by 3 or 4.

7. You can see all SDL's codes for keyboard keys at https://wiki.libsdl.org/SDL_Keycode.

8. Copy and paste the source code lines in Lesson4.cpp from `//Our event structure` to the end of the outer while loop into a Word document. Clearly label these lines "From Lesson4.cpp".

Lesson 5 – Clipping Sprite Sheets

1. Lesson 5 is at http://www.willusher.io/sdl2%20tutorials/2013/08/27/lesson-5-clipping-sprite-sheets/ . The lesson follows the pattern of previous lessons and should be straightforward. You should end up with three versions of renderTexture(). The provided blocks of code for main() should replace existing (from lesson 4) blocks of code, so make sure to comment out or delete the superseded blocks.

2. Copy and paste your entire Lesson5.cpp main() function code into the same Word document. Clearly label these lines "From Lesson5.cpp".

Lesson 6 – True Type Fonts

1. Lesson 6 is at http://www.willusher.io/sdl2%20tutorials/2013/12/18/lesson-6-true-type-fonts-with-sdl_ttf/ .

2. Download SDL2_ttf_devel-2.0.13-VC.zip, expand the zip, and copy the .h, .lib, and .dll files into the usual places. Make sure to download the lesson's sample.ttf file into a resource folder for Lesson6.

3. The lesson 6 code provided uses some functions and main() code developed earlier, so copy them in or maybe start from Lesson5.cpp and delete what isn't needed. You will probably like to have a call to SDL_Delay(). You'll need to add SDL2_ttf.lib as a linker input in the project Properties.

4. Once the program is working, try a different TrueType font. A google search for "downloadable truetype fonts" will provide plenty of leads. Download and copy one to your Lesson6 resource folder, and modify the program's source code to reference the new font's name. Run the program and **take a screen shot showing the output window.** (Alt-PrtScn probably will not

work, so use plain PrtScn instead with no maximized windows, and scale the image to fit on a single Word page.) Copy and paste the image into your Word document. Clearly label the image “From Lesson6”. Include in the Word doc the name of the true type font and the web site your borrowed it from.

5. Following the procedure described in Homework 1 for Lesson 3, update SDL_Template to contain an include for SDL_ttf.h and a linker input for SDL2_ttf.lib.

Lazy Foo’s Lesson 22 – Timing

Another good set of SDL2 tutorials with which you should be familiar are from Lazy Foo. The complete list is at http://lazyfoo.net/tutorials/SDL/. You may find these useful to refer to in the future. For now, we’ll just do number 22, Timing.

1. Create a new project based on your updated SDL_Template (that is, including the header and .lib for True Type fonts), and call it LazyFoo22. Read through the complete lesson, but don’t enter any code yet. To get the code, down load the 22_timing.zip file referenced at the end of the lesson. Copy the code from 22_timing.cpp into your LazyFoo22.cpp file, but for now don’t make a copy of lazy.tff (which will cause an error…).

2. Build and try to run the project. You’ll have to copy DLLs over, as usual. You should get an exit code of 0, but you only see the program window and the command prompt window appear momentarily. The command prompt window may have an error message in it, but it disappears too soon to be read.

3. A solution is to make the command prompt window wait for user input, which will give you time to read the messages. Add the line `getchar();` right before `return 0;` at the end of the program. A good design aspect of Lazy Foo’s code is that there is a single point of exit no matter what kind of error is encountered, which makes it easy to have some code run right before the program ends. The getchar function pauses until the user presses the Enter key. Run the program again, and you should see the messages.

4. OK, now fix the problem by copying lazy.tff from the zip file to res\LazyFoo22 and adding code to call getResourcePath (don’t forget its header file). You may get an error message saying that TTF_OpenFont wants its first argument to be const char*, not something long with templates and sub-templates. In other words, you are passing a `const std::string` to a function that is expecting an “old style” C-string (which is a pointer to char). Fortunately, it’s easy to do the conversion (and it was done in the previous lessons in renderText()). Call the .c_str() method of the object created by the concatentation of resPath and “lazy.tff”.

5. Now it should be working. Note that you need to press Enter to close the console window, even when there isn’t an error message. How could you make the getchar() call happen only when there is a message displayed?

6. **Take a screen shot showing the output window** (the one displaying milliseconds), and paste it into your Word file, labeled "From Lazy Foo 22".

Designing a game engine – first step

The capability of writing text on the screen or elsewhere is definitely something we want a game engine to provide. For now, let's concentrate on a single use-case: writing text on the window, on top of whatever else is on the window. Your task is to design (but not implement) a text-on-screen class that works with SDL2 and SDL_tff. Your design should have the following characteristics and features:

- It is completely encapsulated in a C++ class. For now you will only write the header file, which should not include any method implementations.
- It gives the user control over the font, size, color, and other aspects of the text.
- It is relatively efficient, cleans up allocated resources, and does not leak memory.
- It handles errors gracefully.
- One supported option for positioning text is that the caller supplies the x and y coordinates of the upper left hand corner of the text on the window.
- Two additional options for positioning text are "continue from the end of the previous text" and "start on the next line".
- It allows the user to "turn off" or suppress all text output by setting a single flag or value at compile time. If the class's output is used for debugging, this will make it easy to remove the debug output in a release build of the project (while leaving the code in place).

Much useful information is at http://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf.html.

Create a single .h file that is syntactically correct and defines the C++ class. Upload this to the same EEE Dropbox "ICS161-Hw2" before 1:00pm (note the earlier time) on Wednesday, Jan. 20.