Introduction to Software Development – CS 6010
Lecture 18 – Final Project

Master of Software Development (MSD) Program

Varun Shankar

Fall 2022

Miscellaneous

- Midterm grades have been posted.
  - You can see the specific grading of your exam via Gradescope (see link on Canvas page)
  - Grades are on Canvas.
  - We will have a final exam this Friday that covers all topics from this class.

Lecture 18 – Final Project

- Topics
  - Libaries
    - SFML – Simple Fast Multimedia Library

SFML

- Simple and Fast Multimedia Library
  - Open source on github.
- Drawing, mouse input, timers, audio
- Classes, Structs, Functions needed to make a game.
- Note, these classes/functions use libraries provided by the standard library
  - Image manipulation
    - libPNG, libJPEG
  - Audio
    - libVorbis, libFlac
- https://www.sfml-dev.org
  - API Documentation (Application Programmer Interface)
  - Make sure to use version 2.5 (google searches may give old / out of date information that will not work.

What do I need to use a Library?

- Header Files
  - Need signatures of functions
  - Need to see data in structs
  - Need to see what data is in classes?
    - Why or why not?
    - No – purpose of a class is to hide these details from the user.
- CPP files
  - Maybe, maybe not. Lots of creators of libraries want you to use their library, but don't want to provide you their "trade secrets".
- Compiled code.
  - .o files
  - .a / .so files.

Types of Libraries

- Header Only – includes:
  - .h files
- Source Libraries – includes:
  - .h, .cpp files
    - Have to compile the library yourself.
- Compiled Libraries – includes:
  - .h files
  - Binaries
    - Max OS – .a, .dylib
    - Windows – .lib, .dll,
    - Linux – .a, .so

Package Manager

- Installs libraries for us
- MacOs uses *homebrew*

Homebrew

- https://brew.sh
- Runs a single command to install – via the command line.
  - In general this is not a good idea, but homebrew is pretty standard.
- Once this is done, you will have a command line program named *brew.*

> *brew* # will list all commands available.

- Finding a library

> brew search jpeg

- More info on a library:

> brew info jpeg

- Tree

> brew install tree

- tree <folder>

- brew install cowsay

> brew install sfml

> brew info sfml

- brew update
- brew upgrade

- Homebrew is installed in:

/usr/local/

/opt/homebrew

- If homebrew fails to install:
  - Reboot your Mac and try again.

Make

- Compiling on the command line can get tedious.
- One of the most basic and powerful tools for assisting with this is called Make.
- Instead of calling clang++ with all the parameters, we put everything (commands, paths to headers and libs, etc.) into a *makefile*
  - then simply run “make” on the terminal.

Makefile

```
SRCDIR=../Game/Game
INCLUDE=-I/opt/homebrew/include/

# Compilation Flags
CFLAGS=-std=c++11 $(INCLUDE)
# Linking Flags
LFLAGS=-L/opt/homebrew/lib -lsfml-graphics
-lsfml-system -lsfml-window

CC=g++

.PHONY : all
all : game
```

```
# List of all object files to link together.
OBJS=main.o Ball.o

Ball.o : $(SRCDIR)/Ball.cpp $(SRCDIR)/Ball.h
        $(CC) $(CFLAGS) -c $<

main.o : $(SRCDIR)/main.cpp $(SRCDIR)/Ball.h
        $(CC) $(CFLAGS) -c $<

game : $(OBJS)
        $(CC) $(LFLAGS) -o game $(OBJS)
```

CMake

- Makefiles can get large and tedious
- Fortunately, there are tools that allow us to automatically generate makefiles or even IDE projects.
  - These are called build systems.
- CMake is an extremely popular build system that makes Makefiles, Xcode projects, Visual Studio Projects.
- We'll use Cmake to help generate an Xcode project that knows where SFML libraries live.

Workflow

- Install any libs we need with package manager (aka homebew)
- Create build system project file to incorporate lib into your project
  - In our case, use CMake
- Write and run C++ code
- Working with a partner via GIT
  - Add your partner to your git repo!

Final Project

- With your partner, decide on the project you wish to create.
- Discuss the data and functionality that you will need for your project.
- Sketch out the C++ classes you will need to implement.
  - Any object that you will simulate in your project (e.g.: a car, a space ship, a ball) must be implemented in its own class.
  - What data will they contain?
  - What functions will be needed for those classes?

Final Project – Goals

- Must have several objects on-screen
- Interaction
  - Objects – You must have at least two separate objects.
    - At least 2 objects must interact with each other in some way
  - User – You must allow the user to interact with your game in some way through the keyboard or mouse
- Must create at least one object dynamically

Assignment(s)

- Code Review – Make your own Vector
- Lab – Draw Something
- Homework (Group) – Final Project
- Catch up on all missing HW and lab (if any).
  - This is your last week, and therefore your last chance to catch up.
  - No extensions will be allowed past Thursday for any work.