Warmup
Due: Feb. 6, 2018

Introduction

Welcome to CS1950U! In this assignment you’ll be creating the basic framework of the game engine you will be developing for the rest of the semester. It will introduce first and third-person movement in a 3D world, basic application and game world organization, and some important graphics concepts. By the end of these two weeks, you’ll also have a simple 3D game built on top of your engine!

All assignments in this course will follow a similar format of requiring both engine features and a game built on the engine. They have three sets of requirements: playtesting, primary, and secondary. Primary and secondary requirements are further divided into engine and game requirements. See the Course Missive on the Docs and Resources page of the website for more information about the grading, and the point breakdown for these different types of requirements.

To make sure you fulfill the requirements for a given checkpoint, you should copy the rubric for the checkpoint into your README. The rubric will contain the same requirements as this document, but should be easier to copy directly into your README. Rubrics can be found at /course/cs195u/rubrics/<asgn>/grade.txt.

Before moving on, we recommend reading the Course Missive on the Docs and Resources page to make sure you understand the grading policy, and structure of the course.

If you want to see an example of what Warmup might look like, feel free to run cs195u_demo warmup{1,2} on any department machine.

Support Files and IDE

Getting Started

If you haven’t yet, read the first few pages of the CS1950U Setup Guide on the Docs and Resources page. It contains instructions for setting up a work environment on your personal computer as well as more detailed descriptions of the support code.

To get the support code for this and all future assignments, copy the contents of /course/cs195u/asgn/warmup to your project directory for Warmup (probably something like ~/course/cs195u/engine). This should give you a basic Qt project including a Qt pro file that configures and helps build your project, as well as directories containing some starter code and resources. This code should compile and run right away, give you a black window and a framerate counter.

You’ll use the Qt Creator IDE for this course. Run the cs195u_qtcreator command and open warmup.pro to load the project. Files ending in pro are text files containing the project configuration (a list of sources, compiler flags, and platform-specific build commands). Here are several shortcuts you can use in Qt Creator:

- Ctrl+R: Build and run your project (standard out appears in the Application Output pane)
• Ctrl+K: Quickly open any file in the project by name (in addition to any class or function)
• Ctrl+Click: Jump to the definition of any symbol (variable, function, macro, etc.)
• F4: Switch between *.h and *.cpp files with the same name

Support Code

You will need to implement most projects from scratch, without support code, so make sure to allocate time for design.

However, we realize you have limited time and want you to focus on what’s interesting, so we have provided some support code to get you started on Warmup 1:

• **view.{h,cpp}**: Defines a View widget extending QGLWidget. This is a starting point for your game engine; it sets up a full screen window with mouse capture and a variable-update game loop. Every update of the game loop calls tick() to handle game updates and triggers paintGL() to redraw the view. You will want to fill in these methods when implementing Warmup.

• **mainwindow.{h,cpp,ui}**: Initializes a main window containing a View widget. You should not need to modify these files.

• **main.cpp**: Starts the program.

• **engine/util/CommonIncludes.h**: Contains include statements for universally needed classes and libraries, such as glm. You may add whatever you want to it, however it will be included in many, many files, so try to keep it as small as possible.

• **engine/graphics/Graphics.h**: Contains an implementation of a graphics object, which includes various functions for drawing, and OpenGL state management. It also manages graphics resources such as Textures, Shaders, Shapes, Materials, Fonts, and Framebuffers.

• **engine/graphics/Camera.h**: A default implementation of a Camera object, which describes a view on a 3D world.

• **engine/graphics/*/**: Contains other graphics helper classes which you’ll hopefully find useful!

Note: when you create new folders, you may want to add them to INCLUDEPATH and DEPENDPATH in warmup.pro so you can #include files inside them directly.

Resources

In addition to the support code, we may also provide resources such as textures and models. In this assignment, the only resource we provide is a grass texture entitled **grass.png** (from opengameart.org). Feel free to find other textures or models while doing your projects, but keep in mind that you cannot use any associated code.
Week 1 - Due Jan. 30

Warmup1 will get you used to working in 3D space. Though there won’t always be a ton of requirements related to gameplay, this is ultimately a class about making games, so get creative and have fun with these assignments!

Design Check

- How will you define an application? What about a screen?
- List the steps involved with setting up a first person camera.
- Describe the steps necessary to render the floor.
- How will you implement gravity, the floor, and jumping?

Playtesting Requirements

Playtesting requirements describe what you need in order to have a basic, playable demo for a checkpoint. For Warmup 1, they are as follows:

- Your handin only crashes on edge cases
- Your game renders a quad in 3D
- Moving the mouse pans the camera
- Pressing certain keys moves the camera

Primary requirements

Primary requirements describe the major engine and game features of this assignment.

Primary Engine Requirements

The following will all be a part of your engine. The following features should be applicable to almost any game, as a result, should be logically separated from, but utilized by, your game code:

- Your handin meets all global requirements
- Screen management
  - Virtual Screen class representing a logical subscreen of a game which (minimally) supports:
    - Timed updates (tick)
    - Render events (draw)
    - Input events (mouse and keyboard)
Window size updates (resize)

Virtual Application class representing a whole game which supports:

- Timed updates (tick)
- Render events (draw)
- Input events (mouse and keyboard)
- Window size updates (resize)
- Adding and removing screens
- Switching screens

- Your engine uses the provided Camera object, or you’ve built your own Camera object
- Your engine uses the provided Graphics object, or you’ve built your own Graphics object

**Primary Game Requirements**

For this week, you won’t implement much gameplay. Your handin should allow the player to walk around a world using mouse and keyboard inputs to change the camera. The player will be able to jump and not fall through a textured floor, but there will be no “point” to the game. Remember to keep your game code logically separated from your engine code.

The primary game requirements are:

- Ground must consist of planar geometry with a grass texture
- The player must be able to move using standard first person camera controls, meaning:
  - Horizontal mouse movements change the yaw of the camera
  - Vertical mouse movements change the pitch of the camera
  - Standard WASD keyboard controls (W moves forwards, S moves backwards, A strafes left, D strafes right) change the eye of the camera

**Secondary requirements**

Secondary requirements describe the engine features which are willing to help build your games for a checkpoint, but are not strictly required for all other checkpoints.

**Secondary Engine Requirements**

There are no secondary engine requirements for this week.

**Secondary Game Requirements**

- Handin meets all primary game requirements
- The player must never fall through the ground at \( y = 0 \)
• The player must be able to jump off the ground using the spacebar or a mouse button

• Gravity must act downwards on the player

• The player can only jump when on the ground

• The game must have at least two screens, one of which requires player input to get to the other

• The ground must consist of planar geometry with a tiled grass texture

Keep in mind that there are also global requirements that apply to every checkpoint. Be sure to confirm that you meet these as well!
Week 2 - Due Feb. 6

Warmup2 will add world organization functionality to your game engine, as well as a basic collision system and a third-person camera. By the end of the week, you’ll have your first fully playable game!

Design Check

- How will your camera support both first and third person?
- Describe the steps to determine if, and by how much, two cylinders are colliding.
- Describe the design of your World/System/GameObject/Component hierarchy.
- Which components and systems will you add to your engine and game? What will they be used for?
- What kind of gameplay will you implement?

Playtesting Requirements

- Your handin only crashes on edge cases
- All Warmup1 primary and secondary game requirements must be fulfilled

Primary Engine Requirements

- Handin meets all global requirements
- World/System/GameObject/Component hierarchy
  - World class representing a collection of Systems and GameObjects
    * Timed updates (tick)
    * Render events (draw)
    * System management (adding and removing Systems)
    * GameObject management (adding and removing GameObjects from World and Systems)
  - System class
    * Timed updates (tick)
    * Object management (adding and removing GameObjects or Components)
  - GameObject class
    * Component management (adding and removing Component)
  - Component class
    * Timed updates (tick)
- Better input
Key presses and mouse presses recorded in a map or object of some sort in your engine (for example, a map of key presses and mouse presses in your World, updated on event)

Key press and mouse press information should be accessible to components

- TickSystem which ticks all game objects it owns
- DrawSystem which draws all game objects it owns
- Third person support built into your engine, with the ability to toggle between third person and first

Primary Game Requirements

- Playtesting requirements must be fulfilled
- Player movement should be continuous rather than jumpy (they should move as long as the keys are held down, and stop when they aren't)

Secondary Engine Requirements

- Handin meets all primary engine requirements
- CollisionSystem which handles all collisions between game objects
  - Cylinder-cylinder collision detection
  - Cylinder-cylinder collision resolution, and generation of a response callback
  - Resolution - translate the cylinders out of detection using the MTV
  - Response - dispatches collision callback to collided GameObjects

Secondary Game Requirements

- Handin meets all primary game requirements
- Cylinder-cylinder collision response must be used somewhere
- There must be at least 1 NPC (non-player character), such as an enemy or opponent
- The NPC must have some basic AI
- The game must have a non-arbitrary and reasonably achievable win or loss condition
- The game must be resettable upon win or loss without restarting the program
- The game can never enter an unwinnable or unloseable state
Handing In

Hand in the entire directory tree for your project, including both your engine and game code. You must also include a README file that describes how to verify each requirement, and an INSTRUCTIONS file that describes how to play your game, as specified in the Global Requirements. To hand in, run `cs195u_handin warmup n` from the top level directory of your project (which should be where your Qt pro file is), where `n` is the checkpoint you are handing in. Please do not hand in the build files from your project.