WELCOME TO CS1950U!
Introduction

STAFF
A word from our sponsor
Class Goals

- Build your own 3D game engine, from scratch!
- Build games on top of your game engine!
- Improve your software engineering and design skills!
Useful Skills

• C++
• Graphics/OpenGL
• Basic vector math
Introduction

ASSIGNMENTS
Projects

- Three projects split up into weekly checkpoints
  - One checkpoint every week
- One open-ended final group project
Warmup

- Startup assignment to get familiar with working in 3D space
- 2 week project
- Basic engine architecture, graphics, controls
Dungeon Crawler

• 3 week project
• Topics:
  – Procedural generation
  – AI
  – Frustum culling
  – Raycasting
Platformer

• 3 week project
• Topics:
  – Simple physics/collisions
  – Pathfinding
  – Spatial acceleration
  – Model loading
  – UI/HUD
Final

- ~5 week project
- Your choice of engine features
- Your choice of game features
- Groups required
- Public playtesting
- More details later
Introduction

GRADING
Grading

- Only projects
- Weekly checkpoints
  - Handin due every Tuesday at 11:59 PM
- Each checkpoint worth 3 points
Grading

• For each checkpoint, you have...

• Minimum requirements (1 pt)
  – Constitute a basic, playable demo

• Full requirements (2 pts)
  – Major features

• Extra credit (up to 1 pt)
  – Additional features
Final Grades

• No curve!
  – Do the work, get an A
  – Can get 1 point extra credit each week

• Need to complete all primary engine requirements and a final project

<table>
<thead>
<tr>
<th>Points</th>
<th>Missing</th>
<th>Percent</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>40+</td>
<td>0</td>
<td>100+</td>
<td>A</td>
</tr>
<tr>
<td>34-39</td>
<td>1-6</td>
<td>85-99</td>
<td>B</td>
</tr>
<tr>
<td>33-</td>
<td>7+</td>
<td>84-</td>
<td>C</td>
</tr>
</tbody>
</table>
Design Checks

• Sign up with cs195u_signup <project>

• High-level conceptual questions

• Gives one standard retry, which bring us to ...

Incomplete Handins

- **Standard Retry**
  - As long as you complete a design check, you are allowed to re-hand in a checkpoint

- **Extra retries**
  - You have two for the whole class
  - Can use to retry a checkpoint that you already retried

- You have a week to use each retry
Incomplete Handins

• Minimum requirements cannot be retried
• Extra credit can be retried
• No extra credit until all requirements are met
• Only your best handin will count (retries never hurt your grade)
Out of Retries

• Used the standard retry, out of extra retries, now what?
• You can still do well in the class
  – Don’t have to get credit for all requirements
• You can still pass the class
  – Hand in working version of all engine requirements by the end of the semester
Please hand in on time!

- Falling behind causes a “snowball of death”
- If your handin is playable, hand it in even if you’re missing some reqs so you can be playtested
- If it isn’t, you have another week to retry
Introduction

QUESTIONS?
Introduction

CLASS TIMES
Class Times

• Class: Wednesday 12:00pm-1:00pm
  – Lecture 1 hour (CIT 316)
  – Playtesting ~30 minutes (Sunlab)

• Lecture Review: Thursday 4pm - 6:30pm
  – Location TBD
  – Optional

• Design Checks: Friday-Saturday
  – Sign up using cs195u_signup <project>

• TA hours: Sunday-Tuesday

• Website: http://cs.brown.edu/courses/cs195u/
Introduction

OTHER COURSE POLICIES
Collaboration Policy

• Full version is on our website

• Short version:
  – Can discuss lectures and assignments
  – Can play each other’s games
  – Cannot look at or give any code
  – Can cooperate with other students during TA hours (at TA discretion)
Piazza

• We are setting up a course Piazza
• More info coming soon
Style Guide

• We expect you to have a reasonable style, but don’t require any specific style guide

• If you’re unsure of what counts as reasonable style, pick your favorite style guide from a course you’ve taken and follow it
Test Your Code

• Your code needs to compile and run on department machines
• We can’t grade it if we can’t run it
• Should run at 20+ FPS
Introduction

ABOUT REGISTRATION
Registering for CS1950U

• If you can’t register for CS1950U because the Thursday lecture review overlaps with another course:
  – Don’t panic
  – Fill out this form
  – In description of overlap, indicate that you can make up the cs1950u lecture review during TA hours
  – Talk to Christopher Dennis (Deputy Dean of the College)
Registering for CS1950U

• If you can’t register for CS1950U because you don’t meet the prerequisites
  – Don’t panic
  – Email us for an override code
Registering for CS1950U

• If you can’t register for CS1950U because you’re a RISD student
  – Don’t panic
  – Email our professor (Daniel Ritchie)
Introduction

QUESTIONS?
LECTURE 1
Basic Engine Architecture
Basic Engine Architecture

WHAT IS A GAME ENGINE?
What is a game engine?

- The things that games are built on
- Games tend to have a ton of functionality in common
- Create engines that abstract out common functionality
What is a game engine?

• Usable by many games
  – It should be able to easily create a game without modifying engine code

• Should be general
  – No game-specific logic!

Warmup
Dungeon Crawler
Platformer

Engine
What does this look like?

- Sample hierarchy
  - src/
    - engine/
      - Screen.cpp
      - Screen.h
    - warmup/
      - WarmupScreen.cpp
      - WarmupScreen.h
What does this look like?

- Engine code should never `#include` game files
Basic Engine Architecture

AN ESSENTIAL INTERFACE
A game generally needs...

- Timed updates (ticks)
- To render to the screen (draws)
- Input events
- Resize events
Ticks

• General contract:
  – `void tick(float seconds)`

• Tells the game that a given amount of time has elapsed since the previous tick
  – Nearly all logic takes place during ticks
  – No drawing should take place during ticks
Draws

• General contract:
  – `void draw(Graphics *g);`
  – `void draw();`

• Tells the game to draw itself
  – Convert game state into viewable form
  – No side effects from draw calls

• More information coming up in Graphics section
Input Events

• Most APIs provide input events rather than making you manually poll mouse and keyboard

• Exact contract differs depending on type, but usually of the form:
  – `void onDDDEEE(QDDDEvent *event);`
  – `DDD` = device type (e.g. mouse, key)
  – `EEE` = event type (e.g. moved, pressed)

• Tells the game that an event has occurred
  – Event object contains information about the event
    • e.g. how far the mouse moved; what key was pressed...
Putting it Together

• The Application class

class Application {
    public:
        void tick(float seconds);
        void draw(Graphics *g);
        void onKeyPressed(QKeyEvent *event);
        // more device and event types here...
        void onMouseDragged(QKeyEvent *event);
}
Putting it Together

• Application represents an instance of a game
• You will implement an Application class in Warmup1
QUESTIONS?
Basic Engine Architecture

SCREEN MANAGEMENT
We have an Application

- But how do we build a game around that?
- Drawing/ticking/event handling is very different depending on what’s going on!
  - Menu system
  - The actual game
  - Minigames within game
Screens within Application

• Rather than keeping track of “modes”, separate each “mode” into a dedicated Screen subclass
  – MenuScreen, GameScreen, etc.

• A Screen has similar methods to the Application
  – tick
  – draw
  – input event methods
Keeping track of **Screens**

- **Simplest way:**
  - Single **Screen in Application at a time**
  - **Application** forwards all events to this screen

- **Alternatively:**
  - Map of **Screens** maintained by the **Application**
  - **Screens** can consume events or pass them to a different screen
What are Screens good for?

• For Warmup 1, Screens should
  – Draw the entire game
  – Handle all of the game logic

• In general, Screens shouldn’t do this
  – Results in serious spaghetti code

• Solution: GameWorld
  – Covered next week...
QUESTIONS?
LECTURE 1

Camera and Graphics
Cameras

- Physical camera will render a “film” – a 2D representation of the 3D space
- For virtual cameras, goal is similar
  - Render by squashing view volume (or frustum) onto 2D plane
Cameras in 3D Space

- Camera is not very useful unless we know
  - Where it is (position)
  - What it’s orientation is (pitch, roll, yaw)
Camera Orientation

• **Yaw**
  - Stick a pin in the top of the camera and rotate it around it by this angle

• **Pitch**
  - The camera looking up and looking down by this angle

• **Roll**
  - Only really used in flight simulators
Camera Orientation

- Alternatively...
- Specify direction the camera is facing as a vector
  - Called the “look vector”
Camera position

• Position of camera in the world

• For Warmup 1, in order to achieve first person...
  – Make camera position same as player position
  – Update camera position to make the same as player position
Other Camera Parameters

• Field of view angle
  – How wide is the view volume?

• Aspect ratio
  – Ratio of the width of screen to the height of the screen
Our Camera Class

• Default Camera class provided
  – src/engine/graphics/Camera.h(cpp)

• Allows you to specify all of the above attributes
  – Most likely will only modify position, pitch, yaw
First Person Camera

QUESTIONS?
Camera and Graphics

BASIC GRAPHICS
Motivation

• Certain graphics calls are common to many games
  – Setting up a camera
  – Drawing shapes
  – Setting material properties for shapes
  – Drawing text

• We can store all of our shapes, materials, fonts, etc. in one centralized object
  – Helps us not load them into memory more than once
  – Helps us keep track of them and delete them

• Encapsulated in a “Graphics” object
Graphics Object

• Default Graphics object provided
  – src/engine/graphics/Graphics.h (cpp)

• Methods for …
  – Setting the active camera
    • This camera will be used for rendering
  – Drawing shapes
    • Rectangles (quads), cylinders, and spheres for now
  – Setting materials
    • Change color, texture, lighting of shapes
  – More!
Other Classes

• src/graphics/Shape.h(cpp)
  – Describes the geometry of a shape
• src/graphics/Material.h(cpp)
  – Describes material properties of a shape
• More!
Doing it Yourself

- Feel free to modify graphics support code!
- Feel free to write your own graphics code!
Basic Graphics

QUESTIONS?
LECTURE 1

Controls
PLAYER MOVEMENT
Coordinate systems

- Different game engines define 3D coordinate systems differently
- Most commonly:
  - "Horizontal plane"  
    - Plane parallel to the ground (the xz-plane)
  - "Up-axis"  
    - Axis perpendicular to horizontal plane (the y-axis)
Horizontal Movement

- Keep track of your player position
- Forward movement:
  - Use the horizontal component of the look vector
  - `forward_speed = some positive constant`
  - `dir = normalize(look.x, 0, look.y)`
  - `pos = pos + forward_speed * dir`
- Strafing
  - Use the perpendicular of the horizontal direction
  - `sideways_speed = some positive constant`
  - `perp = normalize(dir.z, 0, -dir.x)`
  - `pos = pos + sideways_speed * perp`
Vertical Movement

- Keep track of the player’s vertical position and velocity
- Jump
  - Assign some positive velocity when the player jumps
  - Make sure the player is on the ground (pos.y == 0) before jumping
- Apply gravitational acceleration each tick
  - $dt = \text{time since last tick}$
  - $g = \text{some negative constant}$
  - velocity = velocity + $g \times dt$
  - pos.y = pos.y + velocity
- Collision with ground
  - After moving the player, set pos.y = max(pos.y, 0)
LECTURE 1
Tips for Warmup 1
Support Code Overview

- **Qt Framework**
  - `main.cpp` – starts up program, toggles fullscreen
  - `mainwindow.h/.ui/.cpp` – sets up window
  - `view.h/.cpp` – basic event framework, where your Application class should reside

- **Vector math – glm (important!)**
  - 2, 3, 4 dimensional vectors and matrices
  - Tons of math – see online documentation

- **QRC files**
  - Allows for easy access of external resources
  - Can use to load your own resources
Support Code Overview

• Utility
  – src/engine/util/CommonIncludes.h
    • Includes glm, iostream
    • Include this anywhere you need glm

• Graphics
  – src/engine/graphics/*
  – Described in previous section
Support Code Overview

• **Methods in view.h/.cpp**
  
  – `DDDEEEEEvent(QEEEEvent *event)` – call `app.DDDEEEE(event)`
  – `tick(float seconds)` – call `app.tick(seconds)`
  – `paintGL()` – call `app.draw(graphics)` or `app.draw()`
  – `resizeGL(int x, int y)` – call `app.resize(dimensions)`

• **Make Application a separate class from View!**
  
  – Put instance of Application class in View, so that you can pass events on to Application
Setup Guide

• If you have time, go through the CS1950U setup guide! (highly recommended)
  – On the Docs page of the website

• It covers …
  – How to set up a camera
  – How to draw something using the graphics object
  – How to add basic player controls
On Your Own

• Play around with graphics object calls
• Specifically try to move, resize and rotate shapes
• 3D graphics can be tricky, especially if you haven’t done it before
  – Feel free to email us or come to hours if there’s something you don’t understand
Qt vs. STDLib

- QString – substrings, splitting, hashcodes
- QList – type-generic dynamic dynamic array
- QHash – type-generic hashtable
- QSet – type-generic set
- QTimer – sets up the game loop
- QThread – easy-to-use threading API
- QPair – great for vector hashcodes

http://qt-project.org/doc/qt-4.8/qtcore.html
Qt vs. C++ STDLib

- QString – std::string
- QList – std::vector
- QHash – std::unordered_map
- QSet – std::unordered_set
- QPair – std::pair

http://qt-project.org/doc/qt-4.8/qtc core.html
C++ Tip of the Week

SMART POINTERS
Raw pointers

• Problems:
  – Declaration doesn’t indicate who owns the object (i.e. who destroys it)
  – Must destroy exactly once
  – Memory leaks
Smart Pointers

• The solution to all of the problems (and more)
  – Most importantly, delete / free object they refer to automatically if pointer goes out of scope

• 3 types in modern C++
  – std::unique_ptr
  – std::shared_ptr
  – std::weak_ptr
Shared Pointers

• In general the one to use
• Same size as raw pointers and perform the exact same instructions
Creating a Shared Pointer

• Use “std::make_shared<T>(args);”
• More verbose than creating a normal pointer, but worth it
Creating a Shared Pointer

- With shared pointers

```cpp
#include <memory>    // Include header file
...
...
std::shared_ptr<Camera> cam = std::make_shared<Camera>();
...
...
```

- Without shared pointers

```cpp
Camera *cam = new Camera();
...
...
delete cam;
```
Copying a Shared Pointer

• Can make as many copies of a shared pointer as you want
  – `std::make_shared<T> s1 = ...;`
  – `std::make_shared<T> s2 = s1;`
  – `std::make_shared<T> s3 = s2;`
  – ...
  – Each refer to the same object

• Object managed by all shared pointers only deleted when all shared pointers go out of scope
In Summary…

• Unique/shared pointers make memory management easier

• Please don’t have memory leaks in your handin code
WARMUP 1 IS OUT

Sign up for design checks!