GAME 1024 - Advanced Game Programming
Course Syllabus
Instructor: Kain Shin

Course Description:
This course is designed to introduce the student to the software engineering aspect of game programming. C++ is required. Your class project will be written in C++ with no exceptions. The purpose of this class is to help prepare you for a job as a professional game programmer at a typical game studio; therefore, Java, C#, or any other programming language will not be accepted for your class project. By the end of this class, you should have the basic building blocks of your own personal game engine written from scratch that you can build upon for the purposes of showing off to potential employers once you add your own flair and convert your class project to either your own game or some useful development tool.

Course Length:
42 hours

Objectives:
After leaving this class, the student will:

- appreciate that game programming is really programming first, game second
- Understand how to architect a game codebase so that it is flexible and easy to work amongst a team of programmers working concurrently
- know how to debug code, and how to write code to be debugged
- Understand the importance of data-driving initilization (properties) and execution (scripting) so that iteration can be done by non-programmers
- understand common modern asset management practices
- have some instinct for the performance tuning of video game code
- understand cross-platform porting issues, and the implications of platform
- understand the role of the programmer in a typical professional game company setting
- Be a better C++ programmer than they were when they entered the class

I will NOT focus on…

- Anything specifically covered in GAME 1023
    - 2D/3D Graphics
    - Input management
    - Event Based Programming
    - Time Based Animation
    - Basic Sound
    - Object Based Organization of Game Programs (Anatomy of a Game Program)
    - Game Design
    - State Machines
    - Managing Game Time
    - Game GUI Implementation
    - 3D Math
    - Intro to AI
- Animation, Graphics, Shaders (GAME 1023, GAME 1040)
- Network Programming (GAME 2014, GAME 2016)
- Audio Programming (GAME 2022)
- AI/Pathfinding (GAME 2023)

- Commercial/Open Source Game Engines (GAME 2019)
- Game Mathematics (GAME 2018)
- General C++ (You are supposed to know it by now)
- Tools Programming (GAME 1020)
- Asset Management (GAME 1050)
- Mobile/Handheld Programming (GAME 2017, GAME 2021)

I will definitely NOT grade you on…

- Game Art - This is a programming class, not an art class
- Game Design - This is a programming class, not a design class. Game design is fun, and we all want to be designers in some capacity, but when you get hired, you will be hired as a programmer first and foremost, designer second… maybe.

Development Environment and Teaching Philosophy

We will be using SDL (Simple DirectMedia Layer) because it is a cross-platform programming environment that gives you low level access to the basic hardware systems, such as sound, input, and graphics. This class is not meant to teach you platform-specific technologies such as Win32, DirectX, or OpenGL which you may or not use on your next job. Those are good things to know, but this class is meant to give you general software engineering skills which all game programmers across the world use. The low level part of SDL's description is important because the intention is to get you capable of building the high level anatomy of a game from scratch with only the low level pieces.

Required Materials

- A development environment that supports SDL
- Note taking supplies, whether it be pencil and paper or typing into a laptop. This also implies that you have fingers available for this purpose.
- Eyes and ears with which to see and hear my presentations with. Special dispensation will be granted to deaf students who can read lips
- A working brain. Must be fully functional in areas of explicit thinking, abstract thinking, and memory. No exceptions.

Evaluation (Grading Policies):

College work must exhibit higher order thinking skills including analysis, synthesis, and evaluation. Mere knowledge about a situation or demonstration of comprehension of the material is not sufficient to prepare you for employment consideration. As a Video Game Development student, you must consistently apply higher order thinking in order demonstrate mastery of the material covered in this course. Grades are given for results not for effort. Read the definitions for each grade noted below, as this is really how grades are determined.

Grading is based on an absolute scale - you are not competing with anyone else, but you will be challenging yourself. There are no distributions of grades; hence, all of you can earn an A in this course. Note: Students earns grades, faculty members do not give them. Your final grade will be based on both individual and learning team performance.

Your final grade will be based on the points that you earn during the course. You may receive “fractions” of points on some assignments. When calculating your final grade, I will use the standard rounding convention – meaning that scores with a fraction of ½ or greater will be rounded up, those with a score of less than ½ will be rounded down. I will use the following grading scale to calculate your letter grade.

The grading scale is based on a 100-point (or percentage) scale:
How points and percentages equate to grades

A	90 and above	A = Excellent performance. Work is exemplary and worthy of emulation by others. Student is in full attendance and constructively contributes to the learning environment.
B	80-89	B = Above average performance. All assignments are complete and exhibit a complete understanding and an ability to apply concepts.
C	70-79	C = Average performance. Accomplishes only the minimum requirements. Oral and written communication is at an acceptable level for a college student.
D	60-69	D = Demonstrates understanding at the most rudimentary level. Work is minimally passing.
F	< 59	F= Work is not passing, characterized by incompleteness, lateness, unsatisfactory demonstration of understanding and application.

The breakdown for the four areas of the student's final grade is as follows:

Attendance and Participation	30%
Quizzes	10%
Project Deliverables (including Milestones)	60%

	Total: 100%

Quizzes:
There will be a quiz at the beginning of every class to go over material from preceding lectures. This quiz also doubles as the attendence token. Quizzes will be followed by an interactive discussion of the relevant material.

Participation:
One person talking in front of a class for almost three hours is really boring, even for a subject as inherently interesting as game programming. I’m teaching this class voluntarily because I’m hoping to have some fun while doing it. I could be working my day job as a game programmer during these evenings or doing a hundred different other productive things with my time, but I’m choosing to spend this time with you fine people. I do not intend to make the lectures last the entire duration of the class. This means we have to make good use of the time after the presentation lecture. We can talk about the lecture material, work on the class project, or play video games while talking shop. It’s up to all of us to make our time in the class as enjoyable as possible while still getting the most out of it. I really want you to get your money’s worth out of this class. Help me help you figure out how to do that.

The Class Project

- Don't be overwhelmed: I will tell you exactly how to do this project during the class lectures
- Since I have never taught this class before, there is a good chance that these milestone requirements will change to scale with your capabilities as a class
- Grading Metrics:
    1. Meeting your milestones on time. There will be three milestones, including your final milestone. You will get 5 points off for every day that you're late on a milestone.
    2. Actually meeting the milestone requirements as written on this syllabus.
    3. Good commenting habits and readable code. If I get in a bad mood while trying to read your code, I will take points off.
    4. Gracefully handle error cases in user data (asserts, or warning messages are good, crashing with an access violation is bad).
    5. Using the architecture in tune with teachings of the course (Listener and Puppet analogies, etc.). Failure to use this architecture and simply hacking your code together defeats the purpose of this class and will cost you a LOT of points. If you meet the written requirements of the milestone, then this will typically be the path of least resistance.
- Tasks for Milestone 1:
    1. Get my simple SDL program up and running on your computer. Since I'm not teaching graphics, these will be your freebie tools to start out with.
    2. Install Perforce on your computer and get your code stored on it (we may cut this if it is not feasible)
    3. #define your own versions of "new" and "delete". Call them NEW and DELETE, respectively. For now, leave them as the default new and delete. You'll be writing a custom memory allocator eventually.
    4. Make it so that you are drawing a list of managed "actors" on the screen instead of raw sprites and text. Actors will be a concept discussed in class. For the purpose of this project, an actor will manifest in this "game" as a sprite with a label underneath
    5. Use an Input System: Map a mouse-click button to a game action such as "possess"
    6. Make the actor change its sprite and label when the user clicks on it
- Tasks for Milestone 2:
    1. Use an INI File: Define some application settings, such as button mappings in the ini file so that when you port your game over to the Playstation 3 version, all you'd have to do is edit the ini file without touching code
    2. Allow a user to click on an actor with the mouse in order to control the actor movement WASD style. Use the controller analogy discussed in class. Make it so that I can data-drive the button that causes possesion… it will not always be the mouse button that causes possession.
    3. Make the actor "run away" from your mouse cursor with data-driven speed, taking care not to let it go offscreen… unless they are "evil", upon which they will chase your mouse cursor.
    4. Scripting: Hook your own script functions to the game so that a designer can specify what should happen when the user clicks on an actor. A designer might want the actor to either "become possessed by WASD" and/or "become evil". So you may have one script file that gets called by code called "OnSelect.lua" and two other script functions that can be used by a designer called "BecomeEvil" and "BecomePossessed".

5. Use XML: Populate the world with these actors using an XML file. Allow each actor to have instance properties such as “running speed”, “image”, and “label”
6. Localization: Make sure all strings are localized such that switching out a file will change all text in the game without changing any code
7. Logging/Debug Output: Make sure all notable occurences are logged in a log file and outputted in the debug window
8. Do one of the following:
    - Store user statistics in XML format
    - Test/Tuning Support: Bring up a debug menu that will allow the user to change properties on the possessed actor
    - Add a useful profiling system to your game
9. Polish up any rough spots that were pointed out in the evaluations of Milestone one and two
10. Optional Bonus: Write a resource manager that loads all bitmaps used in the program into a single memory mapped file. Make it so that when you call your program with a cook parameter, it basically creates one giant super-bitmap that holds all bitmaps used in the game. When the game runs and detects a “cooked” bitmap, it will load one large file instead of a bunch of tiny files. All sprites will share a spot on the bitmap so that no graphics data is duplicated.
11. Optional Bonus: Write a custom memory allocator which you will use with your versions of NEW and DELETE to ensure that no memory is leaked and that the number of heap allocation occurences are minimized
12. Optional Bonus: Integrate your custom memory allocator with STL or whatever template library you are using
13. Optional Bonus: Work in streaming, somehow

Outline (12 classes total at 3.5 hours each)

Lesson 1: Class Intro

- Start with a C++ Quiz as well as a little intro. It will not be for a grade, but it is meant to let you know if you need to drop this class.
- Class Format, Syllabus, and Expectations
- Assign the Class Project
- Be a Good Programmer First, The Game Part Will Follow

Lesson 2: Getting Started

- Version Control: Using Perforce
- Some useful programming tools for your project (STL, boost, smart pointers, etc.)
- The Listener Analogy for Managers
- The Puppet Analogy for Actors
- Organizing your files into a good directory structure
- Header Management
- Organizing the main loop components

Lesson 3: Data-Driven Properties

- Using an ini file
- Introduction to XML

- Discussion: Should I use an XML or ini file?

Lesson 4: Debugging Techniques
- Overwriting “new” to detect leaks using Crt
- Asserts
- Breakpoints
- Output Messages
- Log Files
- Remote Debugging
- Attaching to an external exe (like on a console)
- Using a stand-alone debugging app

Milestone 1 is due before Lesson 5
Lesson 5: Scripting
- Scripting is not properties, properties is not scripting…
- Introduction to Lua
- Discussion: Lua can be used for properties, so why bother with XML for properties?

Lesson 6: Common Helper Functionality
- Localization
- Test/Tuning Support
- Tracking User Statistics
- Recording Play Sessions

Lesson 7: Advanced Features
- Event System Optimization
- Custom Memory Allocation
- Resource Management (Block Loading, Instancing)
- Streaming Technology
- Save/Load

Lesson 8: Hardware Issues in Game Programming
- Know your CPU
- Memory as a bottleneck
- Multiprocessor systems
- Trend: Higher percentage of silicon on the CPU dedicated to cache rather than branch prediction
- GPUs
- Other auxilary processors – physics, sound, network
- Reading from the CD/DVD is very very slow
- Considerations of streaming

Lesson 9: Overview of Specializations (Most of this is covered in other classes)
- User Interface

    - Systems Design
    - Requirements of a user interface
    - Basic Impementation
- Graphics
    - 2D Animation
    - 3D Animation (Bones System)
    - Scene graph
    - Particles
    - Overview of Shaders
- AI
    - Path Planning
    - Agent/Decision Architectures
- Camera Programming Overview
- Audio Programming Overview
- Physics Programming Overview
- Multiplayer/Network Programming Overview
- Tools Programming Overview

Lesson 10: I Will Totally Help You With Your Class Project Today

Milestone 2 is due before Lesson 11
Lesson 11: Class Projects are Due

- Let's Talk About What Else You Can Do With That Engine and How

Lesson 12: Making a Living as a Game Programmer

- Real World Game Programming in a Multi-Disciplined Team
- Interviewing/Job Seeking Advice
- Let's Do Mock Interviews as a Class

Video Game Development Program Philosophy
The Video Game Development Program has been designed, developed and implemented in partnership with leading video games studio managers and directors in Austin. The video games industry has undergone significant changes in how games are developed. They are rarely developed by few persons working in isolation. Today's games are often developed by teams of 50 to 200 on schedules from 2 to 3 years with budgets of $10M to $20M. The large publishers drive the game development funding and schedules. Consequently, it is critical that personnel in the industry communicate and collaborate effectively. This drove the certificate requirements definition. Students are required to successfully complete courses in four categories:

1. The base industry courses: Video Games Industry, Business of Video Games and Video Games Development.
    a. Students will understand what drives the industry, why games are developed, what is needed for success and how to get from idea to delivery.

2. The course specialization courses: Video Game Programming, Video Game Art, Video Game Design and Video Game Production.
    a. Students will understand the requirements, objectives, limitations and goals of the different disciplines in a studio. This is essential for communication and collaboration.
    b. Students in these core courses will be cross-discipline in order to build an understanding and appreciation of how different discipline teams collaborate and contribute to the final product.
3. The five specialization electives.
    a. Students will develop skills in the discipline in which the student will seek employment.
4. Non-specialization electives
    a. These are optional courses that will give you a deeper understanding of what other disciplines do and how they function. They will help you understand how to work with others on the team and to get the 'big picture.' These courses do not count towards the Video Game development certificate.
5. Capstone Project
    a. This multi-person team project will simulate the real video game development environment. Students will develop a concept, turn it into a design, implement the programming and art required and produce it on the committed schedule. Go/no go milestones and final "publisher" acceptance reviews will mimic the industry. The students will have a deliverable for their portfolio that can be used for employment purposes.

Throughout the program each course will focus on knowledge transfer, skill building and teamwork. There will be a heavy emphasis on projects that will broaden and deepen each student's portfolio development. Portfolios are critical to demonstrating an individual's capabilities. Some projects will individual, many will be team based. How much a student gets out of each course will largely be determined by how much the students puts into the course. Video game development is highly complex, difficult work. The courses are designed to prepare students for that environment. So, come expecting
to work hard.

The program is designed to reinforce key concepts such as teamwork, collaboration, and cooperation across all disciplines in the games development and management process. Many concepts are repeated throughout the program because they are extremely important to successful game development.

Finally: We're here to have fun. If you're not enjoying video game development, or this course, please let me know, and we can talk about what we need to do to change that! Good luck!