Who is this guy?!

- Joey Hammer
  - jhammer@cs.ucsd.edu
  - 858.336.6076

- EGL: Experimental Game Lab
- High Moon Studios
- Particle / Aftermath
- Deathmatch Zwei
Experimental Game Lab

- Engine Programmer
- Maya Plugin Programmer
- Audio

Come on by!
Calit2: 1611
EGL: Scalable City
EGL: Scalable City
EGL: Scalable City
EGL: Scalable City
High Moon Studios

- R&D Programmer
  - Win32
  - Xbox 360
  - PS3
- Coffee Boy
Particle / Aftermath

- **Main Interest:**
  - Game Engine
    - Architecture / Design
    - Implementation
    - Tools
  - Spherical Harmonics
    - Rendering Method
    - UE3
  - Game Audio
    - Reactive Music
Particle Project
Universe Creation Toolkit
Superschule-Mädchen-Stadt
Deathmatch Zwei

Joseph Bollish, Ian Marsh, Joey Hammer,
Ryosuke J. Onaka, Ming Kawaguchi,
Brian Ryujin, William Lee

CSE 125: Spring 2005
Superschule-Mädchen-Stadt Deathmatch Zwei
Zwei Overview

- First-Person Shooter
- Single Player and Multiplayer Game
- Wacky Concepts & Storyline
  - Cartoon-like Aesthetic
  - Interesting Weapons
  - Simple Power-ups

- Fun to Play!!!
Zwei Technical Overview

- Modular Engine Design
- Complex Asset Pipelines
- Data-Driven Systems
  - Audio
  - Input
  - Physics
  - NPCs
  - Game Events
Zwei Engine

- Graphics: Direct3D – ZFX
- Audio: DirectMusic – Dynamite Squirrel
- Input: DirectInput
- Networking: DirectPlay
- Physics: written in-house
- NPC AI: written in-house
- UI / Menu: DirectX Samples
Zwei Asset Pipeline

- Environment & Ships
  - Maya → MilkShape → CBF Converter → .CBF

- Japanese School Girls
  - ??? → MilkShape → CBF Converter → .CBF

- Sound Effects & Music Samples
  - Joey → Reason → SoundForge → .wav

- Textures
  - Photoshop → { .bmp, .tga }
Zwei Team Dynamics

- Group of 7 (large... perhaps too big)
- Strong Team Organization
  - Started from the Beginning
  - Shared Leadership
- Balanced Skills
  - Coding Expertise
  - API Familiarity
- Mediocre communication (again, large size)
Zwei Collective Ownership

- **Game Ownership**
  - We all cared about Zwei
  - Mutual Agreement
    - Game Concept and Direction
    - Features (weapons, mechanics, overall aesthetic)

- **Code Ownership**
  - Problems were solved by Group-Solutions
  - High-Level Decisions were made by a majority
Zwei Retrospect :: Pros

- Simple, Fun Gameplay
- Creative Concept
  - Deviated from our original idea
    (Mechwarrior-like gameplay with destructible environments)
  - Let the gameplay / “fun-factor” decide how it would turn out
- Data-Driven Framework
- Well Tested

**Demoed well!**
Zwei Retrospect :: Cons

- Idle team members
- Asset pipeline put us behind schedule (ZFX)
- Modules compiled as DLLs (ZFX)
- Inverted priorities for particular features
- Some modules had to be rewritten or reduced
  - Networking: used *managed* code in first iteration
    - Broke everything!
    - Had to be rewritten with unmanaged code
  - Audio – too powerful!
    - Slowed game to 2 FPS!
    - Had to be reduced into not using functionality we had originally hoped to have
Game Engine Design
Game Engine Overview

Application - AI, input response, game data.

Animation, deformation, physics, collision detection.

Engine Scene Renderer

GFX Renderer  Sound Renderer  Input handler

GFX API  Sound API  Input API

GFX hardware  Sound hardware  Input Hardware
Design Principles (Ideal)

- Modular Design
- Flexible Coding Style / Architecture
- Object-Oriented Architecture
- Demo-Driven
- Data-Driven
Modular Design

- Many hands touching code
- Want bugs to be isolated
- Ideal Case:
  - Static Libraries for each module
  - Framework Layer
  - Game Application Layer
- Realistic Case:
  - Single Application
- Suggestion: Try for somewhere in between
Flexible Coding Style / Architecture

- Mid-Project Changes WILL Happen!
  - Feature Priorities
  - Game Requirements
  - Mechanics
  - Ideas
- Don’t over-design!
- Iterative Development
  - Write code for NOW!
    Add what’s needed as you go…
Object-Oriented Architecture

- **Structs**
  - Data without behavior

- **Classes**
  - Data coupled with behavior

- **Interfaces**
  - Data / Behavior abstraction

- **Inheritance / Polymorphism**

- **SRP**
Inheritance / Polymorphism

- Don’t overuse!
  - expensive
  - harder to maintain
  - dangerous
- Keep shallow hierarchies
- Utilize *containment* and *encapsulation* principles in addition to inheritance
Single Responsibility Principle (SRP)

- “Classes should have only one reason to change”
  - Robert C. Martin

- A class should encapsulate ONE behavior / purpose
- If it has more than one behavior / purpose…
  - Break it into two or more classes
  - Think about abstraction
  - Rethink design

- Remember your clients!
  - Team members
  - YOU!
Demo-Driven

- GOAL: have a working demo each week
  - Module demos before integration phase
  - Progressive demo after integration phase
- Shows team what others are doing
- Prototyping
  - Do NOT be afraid!
  - Provides testing grounds for:
    - New features
    - New mechanics
    - New assets
Data-Driven

- Use *.ini files for *EVERYTHING*!
  - int
  - string

  OR Write a mini-parser for more options

- Do not hardcode asset attributes

- You want assets to be added and change as easily as possible!

- Minimize the number of places in the code where you need to change things by adding a new asset

- Industry: Artists vs. Programmers
C++: Gotchas & Coding Practice
Floating-Point Comparisons

- `==` will only give a bitwise comparison
  - Not good enough with floats
- Use epsilon comparison:
  ```c
  if((fA - fB) > fEpsilon)
  ```
Asserts

- Use them everywhere you can!
- Only active in Debug Build
- Nulled out in Release Build – compiler will remove the nulled instructions
- Asserts are for programmers
- Helps find bugs quick!
- Good for making sure parameters are valid
Const Correctness

- If you haven’t written const-correct code before, this could be hard to get used to
- It is important to know what it does
  - may see it in the API’s you use
- Important for shared-ownership code
- Empowers compiler to enforce your assumptions
- See Llopis chapter 3
Pass by Reference

• Instead of passing pointers of objects, pass references
• Even better, pass a `const` reference!
• Pointers can be NULL, references cannot
• Don’t need to pass POD’s by reference – use pointers for these
Naming Conventions

- Use long, descriptive filenames
- Use long, descriptive class names
- Use long, descriptive function names
- Use long, descriptive variable names
- Visual Studio .NET intellisense can fill in
- Will pay off in long run – especially during “crunch time”
Commenting

- Comments get out of date fast!
- Comment hacky things to let your team know where you made assumptions
- Assertions are some of the best comments
- Shouldn’t have to comment if using naming conventions above
Use the STL

- Containers
  - vector
  - string
  - list
  - deque
  - set / multiset
  - map / multimap
- Algorithms
  - sorting
  - searching
- Iterators
- Effective STL (Meyers)
Global Variables

- Sometimes you just have to…

Suggestions:
- Only make something global if everything (or almost everything) uses it
- Use a global struct:
  - Declare in a header file
  - Define in a source file
  - Use extern to gain access to it – only in cpp!!!
- Keep centralized
  - Globals.h and Globals.cpp
Memory Management

- Do your best…
  Remember: end goal is a 15 min demo

- Every new requires a corresponding delete

- If dynamically creating an array:
  - new[] will be called
  - MUST destroy with: delete[] operator
Development Environment
Visual Studio .NET

- Lots of compiler options
  (see VS.NET handout)
- Very customizable
- Want to setup things right from the start
- Complicated build systems can get you into trouble
- Ask Joey
Development Console

- Need a console window and an application window (Graphics, MFC control panel, whatever)
- Make sure everyone uses console for debugging
- Custom assert function:
  - `G2_ASSERT()`
  - prints to the console (color: red?)
  - logs file, function, and line number
    - `__FILE__`
    - `__FUNCTION__`
    - `__LINE__`
Development Console

- My Personal Solution: **DebugConsole** class
  - all static functions
  - reactive to build
    ```
    #ifdef _DEBUG
    ...
    #endif
    ```
- Make your custom assert use this interface
- Nice to utilize **printf()** like syntax
  ```
  Print(const char* outputStr, ...);
  ```
- **WATCHOUT!**
  - STL strings need to be passed as **char**
  - use `c_str()`
Development Log

- Keep a log in Debug builds
- Could hook this into `DebugConsole` class
- Important for tracking down bugs during prototyping and integration

Other Suggestions:
- Named with date and time
- Keep a thin log for Release builds
Resources Overview

● Books
  ● Small listing to follow
    ● Not the best out there
    ● Geared for what’s best for this class
  ● Complete listing on pisa

● Internet
  ● Small listing to follow
  ● Complete listing on pisa

● Shorts
C++ Books

- C++ for Game Programmers (Llopis)
  - “C++ for UCSD Students”
  - How I really learned C++
  - Industry-forged advice and examples
  - Treats you like a programmer
  - Great desk reference
Game Math Books

- 3D Game Math Primer (Dunn)
  - Best 3D Game Math book in print!
  - Clean explanations
  - Great examples
  - Useful code!
  - Perfect for rolling your own Math Library!!!
  - **WARNING:** website never updated
Zen of Direct3D Programming (Walsh)
- Great introduction to D3D and Win32
- Useful topics
  - Bitmap Text (HUD)
  - Sprites
  - Quake-like console
Graphics (OpenGL) Books

- OpenGL Game Development (Seddon)
  - WARNING: I haven’t read this completely
  - Good reviews from industry coworkers
Input (DirectInput) Books

- No book on Input
- Beginning DirectX9 (Jones)
  - Great Introduction to DirectX API
  - Includes a simple, but useful chapter on DirectInput
- Seek help in DirectX Samples
- Could roll-your-own with Win32…
Audio (DirectSound) Books

- Really just need sound effects and background music track to loop
- Beginning DirectX9 (Jones)
  - Includes a simple, but useful chapter on DirectSound
  - Will provide what you need for above
- DirectSound does NOT support *.mp3
  Must use DirectShow for this…
I’m an audio nut

DirectX Audio Exposed (Fay)
- THE guide to DirectMusic
- Amazing features!
  - Beat-mapping
  - Key-matching
  - Doppler-Effect
  - Spatialization

Don’t do this unless your game is going to be seriously based on one of these features!
Artificial Intelligence Books

- Programming Game AI By Example (Buckland)
  - Best Game AI book in print!
  - Simple, easy to follow
  - Code that works!
  - Perfect for NPC’s (if you decide to have them)
Game Physics Books

- Physics for Game Developers (Bourg)
  - Simple, easy to follow
  - Useful Code!
  - Cool Topics
    - Hovercrafts
    - Force Fields
- Game Physics (Eberly)
  - Complete Game Physics book
  - Uses WildMagic engine
    - Less useful code
    - Look at mathematics and concepts
Game Networking Books

- No really good books out there
- Look at DirectPlay samples
  - WATCHOUT! No Managed Code!
  - Was deceptively easy at first
Internet Sources

- www.gamedev.net
- www.flipcode.net
- www.devmaster.net
- www.game-development.net
- www.geometrictools.com
- www.codeproject.com/cpp/
- www.hammerian.net