Pedagogical Goal

• Your technical skills should not be tied to any particular game engine
• Just like your programming skills should not be tied to any particular programming language
• Use best tools for each job
• ... or tools you were given 😊

Game Engine Definition

**Game Engine**
“A series of modules and interfaces that allows a development team to focus on product *game-play content*, rather than *technical content*.”

[Julian Gold, O-O Game Dev.]

• But this class is *about* “the technical content”! 😊

Buy versus Build

• Depends on your needs, resources and constraints
  – Technical needs (e.g., “pushing the envelope”?)
  – Financial resources (e.g., venture capital?)
  – Time constraints (e.g., 1 month or 2 years?)
  – Platform constraints (e.g., Flash?)
  – Other factors (e.g., sequel?)
• Most games commonly built today with some sort of “engine layer”
**Choices: “It’s a Jungle Out There”**
- 372 3D engines reviewed at DevMaster.net
- IndieDB shows 407 engines, rates top 10

**Evaluation by Size – Lines of Code**

“Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs.” - Bill Gates

- Dragonfly (2014) 4k
- id Tech 1 (1999) 79k
- id Tech 2 (2001) 138k
- id Tech 3 (2005) 329k
- id Tech 4 (2011) 586k
- UE4 v4.6 (2015) 1964k

**Game Engine Architecture**

What are architecture choices for Game Engine layer?
Types of Engine Architectures (Broadly)

- **Monolithic** (e.g., GameMaker)
- **Modular**
  - Extensible IDE (e.g., Unity, UE4)
  - Open Class Library (e.g., C4, UE4, or what Dragonfly would be when it grows up 😊)

Monolithic Engines

- “Old style” - typically grew out of specific game
  - e.g., ID Tech for first-person shooters
- Tend to be genre-specific
  - e.g., GameMaker for arcade-style games
- Difficult to go beyond extensions/modifications not anticipated in API (e.g., scripting)
- Proven, comprehensive capabilities
  - Good for original purpose

Modular Engines

- “Modern” - often developed by *game engine company* (relatively new category)
  - e.g., Unity
- Use object-oriented techniques for greater modularity
- Much easier to extend/replace components than for monolithic engines

Modular: Extensible IDE’s

- GUI-oriented development process
  - More accessible for novice/casual programmers
  - More “art asset friendly”
- Comprehensive asset management
  - Integrated with IDE
- Limited (or controlled) exposure of internals
  - Prevents abuse
  - Prevents some extensions

3/24/2015
Modular: Open Class Library

- Code-oriented development
- Carefully layered
- Allows maximum modifiability
- Often open source
  - UE4 source available, but not freely distributable
- Not as accessible for novices and “casual” programmers

Game Engine Architecture Blocks

Where to place each component?
Best Engine Choice is Relative to Situation

• Similar issues of needs, resources and constraints (as in buy vs. build)
  – Platform, programming language constraints
  – Cost constraints (commercial run $ to $$$)
  – Specific technical features required (e.g., MMO)
  – Previous experience of staff
  – Support from developers, user community (e.g., forums)
  – Pedagogical goals (e.g., this course, or even to teach yourself)

Choice of UE4 for IMGD 4000

• Relatively easy (trivial) for artists
  – C4 tough, Dragonfly limited
  – Comparable to Unity?
• Programming in C++
  – Still “gold standard” for tech game development
  – Need for IMGD majors to do more, get better
• Full support of mature IDE
  – Microsoft Visual Studio (Windows), Xcode (Mac)
• Source code available
  – Aid in debugging interactions
  – Future offerings may delve into code

UE4 in Timeline of FPS Game Engines

(Click below to open browser to image for zooming)

http://commons.wikimedia.org/wiki/File:Fpsengine.svg
Feature Comparisons

- C4 & Unity from [DevMaster.net](http://www.devmaster.net)
- UE4 from [UE4 Features](http://realtimecoding.com) and other UE4 docs

Caveats:
- Not complete - broad view of main features touched upon in IMGD 4000
- Info is not audited (e.g., DevMaster.net from enthusiasts, UE4 from my knowledge and Epic docs)
- Let’s not get bogged down in the details – the idea is to get overall sense of emphasis

General Features

Object-Oriented Design, Plug-in Architecture, Save/Load System
- Clean class hierarchy for scene graph nodes
- General state serialization support for saving worlds
- Separation between per-instance and shared data
- External scene graph referencing from within another scene graph
- Support for pack files and virtual directory hierarchy
- Skinable GUIs

Object-Oriented Design, Plug-in Architecture, Save/Load System
- Professional FPS controller ready to drop in (and tune)
- Streamed loading for the Unity Web Player
- Unity asset server / asset source code version control
- Cross-platform Web player
- Standalone executables for both Mac OS X and Windows
- Mac OS X Dashboard Widgets
- iPhone Publishing is available as add-on product
- Streaming Asset Bundles: the ability to stream in any asset (terrain, mesh, etc) into the game

Object-Oriented Design, Plug-in Architecture, Save/Load System
- Professional FPS controller ready to drop in (and tune)
- Multiplatform compilation – Windows, Mac, Linux Mobile
- Built-in content and community integration

Physics

- Basic Physics, Collision Detection, Rigid Body
  - Built-in character controller
  - Built-in projectile controller
  - Real-time fluid surface simulation
  - Real-time cloth simulation

- Basic Physics, Collision Detection, Rigid Body, Vehicle Physics
  - Powered by the PhysX Engine, which also supports particle physics

- Basic Physics, Collision Detection, Rigid Body, Vehicle Physics
  - Powered by the PhysX Engine, which also supports particle physics

Scripting

- Graphical script editor
- Scripts are edited graphically for easy artist/designer access
- Games can easily define custom script components, and these automatically appear in the editor
- Controllers can advertise custom function calls that can be accessed from scripts
- Scripts support variables, looping, and conditional execution, all shown in a concise graphical manner
- Uses the Mono and supports JavaScript, C# and Boo, interoperable (to a certain extent) and JIT’ed to native code
- Complete scripting documentation
- Source-level debugging

- Blueprints visual scripting, easier “programming” for artists and designers
- Live debugging of script code before trying out in game
- Extensible scripting • Objects can link with blueprints to be used in script code
Builtin-Editors

- Full-featured integrated cross-platform world editor
- Interface panel editor
- Complete built-in windowing system
- Powerful and intuitive interface design
- Advanced surface attribute manipulation and material management
- Editor provides asset pipeline: save a file and it updates automatically
- Asset Server that provides version control capabilities for Unity projects
- Optimized for use with large projects
- Updates, commits, and graphical version comparisons inside the Unity editor.

World editor
- Version control integration – indicates objects that are checked in, out. Can do diffs, etc. within editor

Graphics

Lighting: Per-vertex, Per-pixel, Lightmapping, Radiosity, Gloss maps, Anisotropic:
- Texturing: Basic, Multi-texturing, Bumpmapping, Mipmapping, Projected
- Shaders: Vertex, Pixel, High Level
- Shadows: Shadow Mapping, Projected planar, Shadow Volume

Client-Server:
- Fast, reliable network implementation using UDP/IP
- Solid fault tolerance and hacker resistance
- Advanced security measures, including packet encryption
- Automatic message distribution to entity controllers
- Cross-platform internet voice chat

Client-Server:
- Build on Raknet
- Supports .NET library and asynchronous WWW API
- Multiplayer networking (advanced NAT punch-through, delta compression, easy to set up)
(c.f. guest lectures later in term)

Networking

Client-Server:
- Communication via RPC
- Reliable and unreliable
- Built-in voice support
- Network simulation features (e.g., packet lag, packet loss)

AI system:
- Behavior trees
- Real-time navmesh (pathfinding)
- Environment query tree

AI

AI system:
- Behavior trees
- Real-time navmesh (pathfinding)
- Environment query tree