Game Engines

IMGD 4000

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 <u>about</u> "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"

Why Build?

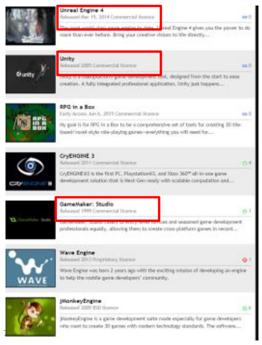
- Need – Technical needs of game not supported by existing engines
- Pedagogy – learn specific skill/concept
- Control – Provide a better understanding of engine-game interaction when making game
  - Can extend/adjust engine if needed
- Genre – have engine especially fit genre (lightweight, just features required)
- Licensing – don't want to pay out royalty fees
  - Note, simple cost should not be a reason – there are many excellent cheap/free engines → it will "cost" *more* to build an engine!

Why Buy?

- Financial – don't have the time/money to build and engine
- Support – existing engine has large user community and/or documentation and/or technical support
- Robust – existing engine has fewer bugs, tried and true code base
- Experience – development team has prior experience with engine

Choices: “It’s a Jungle Out There”

- 375 3D engines reviewed at DevMaster.net
- IndieDB shows 470 engines
  - Most popular (left)
- We are *not* going to try to review them all here


Unreal Engine 4
Released Mar 19, 2014 Commercial license
The most world-class game engine to date. Unreal Engine 4 gives you the power to do more than ever before. Bring your creative visions to life directly...

Unity
Released 2005 Commercial license
Unity is a multiplatform game development tool, designed from the start to ease creation. A fully integrated professional application, Unity just happens...

RPG in a Box
Early Access Jun 6, 2015 Commercial license
My goal is for RPG in a Box to be a comprehensive set of tools for creating 3D tile based/voxel-style role-playing games—everything you will need for...

CryENGINE 3
Released 2011 Commercial license
CryENGINE®3 is the first PC, PlayStation®3, and Xbox 360™ all-in-one game development solution that is Next-Gen ready with scalable computation and...

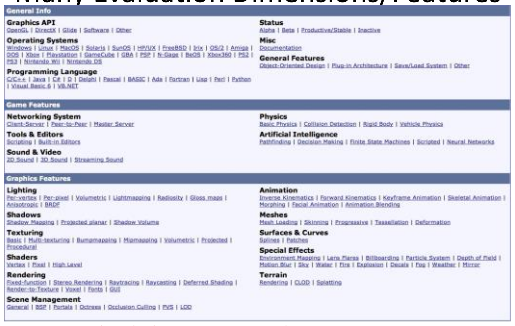
GameMaker: Studio
Released 1999 Commercial license
GameMaker: Studio caters to entry-level novices and seasoned game development professionals equally, allowing them to create cross-platform games in record...

Wave Engine
Released 2013 Proprietary license
Wave Engine was born 2 years ago with the exciting mission of developing an engine to help the mobile game developers' community.

JMonkeyEngine
Released 2009 BSD license
JMonkeyEngine is a game development suite made especially for game developers who want to create 3D games with modern technology standards. The software...

[DevMaster.net]

Many Evaluation Dimensions/Features


General Info

Graphics API
OpenGL | DirectX | Glide | Software | Other

Operating Systems
Windows | Linux | MacOS | Solaris | SunOS | HP/UX | FreeBSD | Irix | OS/2 | Amiga | DOS | Xbox | Playstation | GameCube | GBA | PSP | N-Gage | BeOS | Xbox360 | PS2 | PS3 | Nintendo Wii | Nintendo DS

Programming Language
C/C++ | Java | C# | D | Delphi | Pascal | BASIC | Ada | Fortran | Lisp | Perl | Python | Visual Basic 6 | VB.NET

Status
Alpha | Beta | Productive/Stable | Inactive

Misc
Documentation

General Features
Object-Oriented Design | Plug-in Architecture | Save/Load System | Other

Game Features

Networking System
Client-Server | Peer-to-Peer | Master Server

Tools & Editors
Scripting | Built-in Editors

Sound & Video
2D Sound | 3D Sound | Streaming Sound

Physics
Basic Physics | Collision Detection | Rigid Body | Vehicle Physics

Artificial Intelligence
Pathfinding | Decision Making | Finite State Machines | Scripted | Neural Networks

Graphics Features

Lighting
Per-vertex | Per-pixel | Volumetric | Lightmapping | Radiosity | Gloss maps | Anisotropic | BRDF

Shadows
Shadow Mapping | Projected planar | Shadow Volume

Texturing
Basic | Multi-texturing | Bumpmapping | Mipmapping | Volumetric | Projected | Procedural

Shaders
Vertex | Pixel | High Level

Rendering
Fixed-function | Stereo Rendering | Raytracing | Raycasting | Deferred Shading | Render-to-Texture | Voxel | Fonts | GUI

Scene Management
General | BSP | Portals | Octrees | Occlusion Culling | PVS | LOD

Animation
Inverse Kinematics | Forward Kinematics | Keyframe Animation | Skeletal Animation | Morphing | Facial Animation | Animation Blending

Meshes
Mesh Loading | Skinning | Progressive | Tessellation | Deformation

Surfaces & Curves
Splines | Patches

Special Effects
Environment Mapping | Lens Flares | Billboarding | Particle System | Depth of Field | Motion Blur | Sky | Water | Fire | Explosion | Decals | Fog | Weather | Mirror

Terrain
Rendering | CLOD | Splatting

If there’s a feature term here you don’t know, you should look it up!

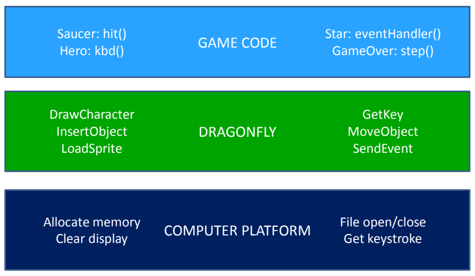
Evaluation by Size – Lines of Code

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

Dragonfly (2015)	5k
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

- Used cloc
- Only counting C, C++ and header files.

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

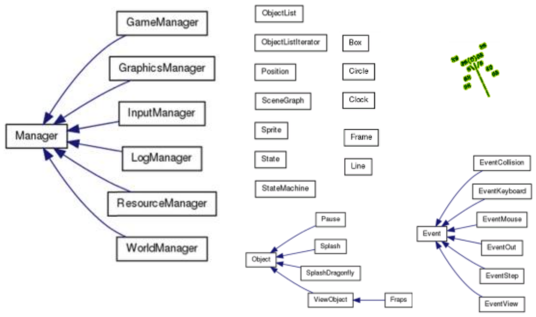
Modular: Extensible IDE's

- 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
  - But also prevents some extensions

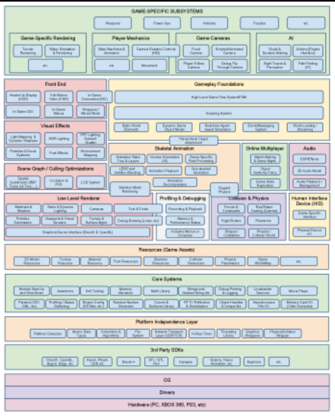
Modular: Open Class Library

- 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

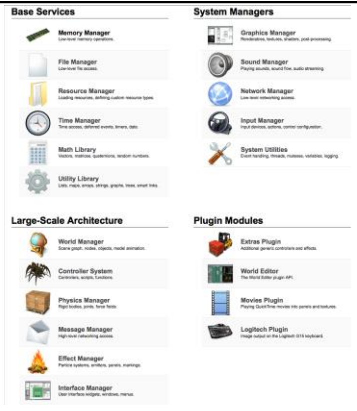


Game Engine Architecture Blocks (complete?)



Game Engine Architecture, by Jason Gregory, 2009, AK Peters, ISBN: 1-5688-1413-5.

Game Engine Architecture Blocks



Base Services

- **Memory Manager** Low-level memory operations.
- **File Manager** Low-level file access.
- **Resource Manager** Loading resources, defining custom resource types.
- **Time Manager** Time access, deferred events, timers, date.
- **Math Library** Vectors, matrices, quaternions, random numbers.
- **Utility Library** Lists, maps, arrays, strings, graphs, trees, smart links.

System Managers

- **Graphics Manager** Renderables, textures, shaders, post-processing.
- **Sound Manager** Playing sounds, sound flow, audio streaming.
- **Network Manager** Low-level networking access.
- **Input Manager** Input devices, actions, control configuration.
- **System Utilities** Event handling, threads, mutexes, variables, logging.

Large-Scale Architecture

- **World Manager** Scene graph, nodes, objects, model animation.
- **Controller System** Controllers, scripts, functions.
- **Physics Manager** Rigid bodies, joints, force fields.
- **Message Manager** High-level networking access.
- **Effect Manager** Particle systems, emitters, panels, markings.
- **Interface Manager** User interface widgets, windows, menus.

Plugin Modules

- **Extras Plugin** Additional generic controllers and effects.
- **World Editor** The World Editor plugin API.
- **Movies Plugin** Playing QuickTime movies into panels and textures.
- **Logitech Plugin** Image output on the Logitech G15 keyboard.

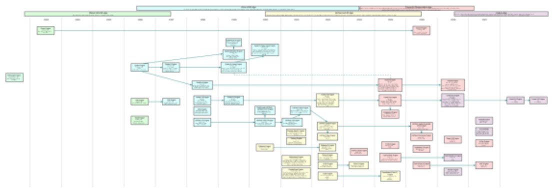
Best *Engine Choice* is Relative to Situation

- 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 art pipeline, 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

vs vs

- C4 & Unity from DevMaster.net
- UE4 from UE4 Features 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 GUI's

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
- Cloth simulation

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'ted 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

25

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
- Editor Extensibility: Create custom editor windows, and new tools and workflows
- 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

26

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
…

Lighting: Per-vertex, Per-pixel, Lightmapping
Texturing: Basic, Bumpmapping, Procedural
Shaders: Vertex, Pixel, High Level
Shadows: Projected planar
…

Lighting: Lightmapping, Per-pixel,
Texturing: Basic, Bumpmapping
Shaders:
Shadows:
…

27

Networking

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)
  (cf. guest lectures later in term)

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

28

AI

AI system?

AI system:
- Real-time navmesh (pathfinding)

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