Game Components

Introduction

Games are software, therefore, game development is software development. By this I mean that game architecture is software architecture, only specialized to the kinds of things games do.

A key difference in game software is the broad scope of CS topics involved in today's games, versus other large scale software projects. Games are not necessarily more complex, but they do have, typically, a wider scope.

Game Architecture Overview

The following represent major components of typical game engines:

- Rendering
- Input
- Audio
- AI
- Networking
- Physics
- Scripting
- Misc Components
    - Event System
    - Menuing System
    - Configuration System
    - Database System
    - Localization
    - Debugging and Performance Tools
- Content Generation Tools
    - Level editors
    - Modeling (3D, Environment, etc.)
    - Texture Editors
    - Simulations (pre-computed water, lava, physically based animations, explosions, etc.)

Rendering

*This is what draws things to the screen.

*Provides an interface for other game systems to get things drawn. Other game systems do not make native OpenGL/DirectX (Graphics API) calls to do things, instead, this rendering system provides a higher level abstraction of the kinds of things the game engine is capable of doing.

*Manages different screen resolutions, aspect ratios, color ranges, etc.

*Font support

*Might be related to offline rendering capabilities, including the ability for the rendering component to be embedded into a custom editing tool.

*This is way beyond what I expect for this class, but take a look at this web site to get a feel for what a "game" rendering API looks like:

http://www.udk.com

Input

*Need a system to collect the various user inputs and distribute them to the game components that are interested.

*Might use a subscriber model to do this (draw this)

*Might have an input queue and pull from the queue.

*Might be multiple queues, one for each input type

*Input Components:

*Keyboard

*Mouse (motion smoothing issues)

*Game Controllers

*Sound (voice recognition)

*Video (think Eye Toy, Kinect)

Audio

*This is what makes your speakers do something. We will have less emphasis on this for the course because of our extremely limited ability to create sound assets.

*Similar to the rendering system: provides an the ability for other game system components to allow audio events to be performed.

*Menu clicks, menu/screen transitions, etc.

*In game sounds: foot steps, monsters, anything that makes sound

*Music system, including "dynamic" (context aware) music.

*Audio is one of those things that is generally the same across most games, in other words, "it is figured out." Things like surround sound, 3D positioning, spatialization, multi-channel playback (4.0, 5.1, 6.1, 7.1), mixing and etc. are now handled pretty much by the platform API.

AI

The collection of techniques used within a game to produce the illusion of behavior in characters/objects/whatever not controlled by the player.

*The techniques used come from a broad range of sources.

*Some techniques used in game AI are:

Behavior Trees
Decision Trees
Finite State Machines
Evolutionary Algorithms
Hierarchical Logic
Machine Learning
Natural Language Processing
Navigation
Neural Networks
Path Finding
Planning Algorithms
Priority Algorithms

http://www.aigamedev.com
http://www.gameai.com

Networking

Providing the ability for two of more instances of a game client, running on different physical machines, to communicate for some game play purpose.

*Uses for networking...

Sharing/Transferring of game data: Characters, stats, saved games, etc.
Social Communities (in game): Chat systems, mail systems, clans, guilds, etc.
Human opponents over a LAN
Human opponents over the Internet
Peer-to-Peer multi-player models
Client/Server multi-player models
MMO architectures

*Most common networking layer is TCP/IP, using either TCP or UDP

*Issues in networking

Bandwidth, which limits the number of concurrent players
Latency, which limits the speed of character interactions
Packet Loss, which affects the kinds of game and experience that can be expected
Synchronization, which limits the speed of interactions, increases complexity

Physics

Physics is used to describe the movement of objects in the game world.

*Relatively speaking, robust physics in games is fairly new and is taking on a greater role in a complete game engine.

*Many objects in games require accurate physical models to control their actions

*Rigid body physics: Think about the ubiquitous crate. Any game play objects that might bump into each other need to have those interactions simulated.

*Ability to modify animations to incorporate physical interactions with the game

environment.

*Rag doll physics to control animations

*Vehicle simulations, think about racing games and the complex friction models for tires, aerodynamics, crashing.

*Simulate explosions, rather than artists having to manually animation

*Cloth simulation

*Collision detection is now becoming a part of the physical simulation, not a part of the graphics engine.

*Physics does not need to be realistic, because realistic might not be fun!

*Show the Farseer XNA Physics demo

Scripting

In an effort to loosen the coupling between the game (play) and the software that provides the game environment, many developers are creating scripting capabilities into their game. The scripting language, along with game specific content, is use to create the game; in the background the game engine takes the instructions from the scripting engine in combination with the game assets to present the game

*The scripting language may or may not be the same language as the game engine.

*If the game engine is provided as linkable objects, the scripting language might be the same as the game engine. For example, game engine written in C++, provides as libraries (or similar), then the game is written in C++, using the engine libraries to do all the engine stuff

*The game engine might be a runtime engine with the scripting code interpreted by a runtime interpreter that drives the various game engine components. The Unreal Engine follows this pattern...it compiles Unreal Script into a byte code that is then interpreted at runtime to create the game.

Misc Components

Not much detail on this right now, but game engines contain many other sub-systems that support the overall experience and technical needs. Sometimes these systems are small and relatively minor, while some may play a major role in the game engine design (such as the database system). Others may not contribute to the game itself (debugging and performance), but provide essential development support.

*Event System (game triggers)
*Menuing System
*Configuration System
*Database System
*Localization
*Debugging and Performance Tools

Content Generation Tools