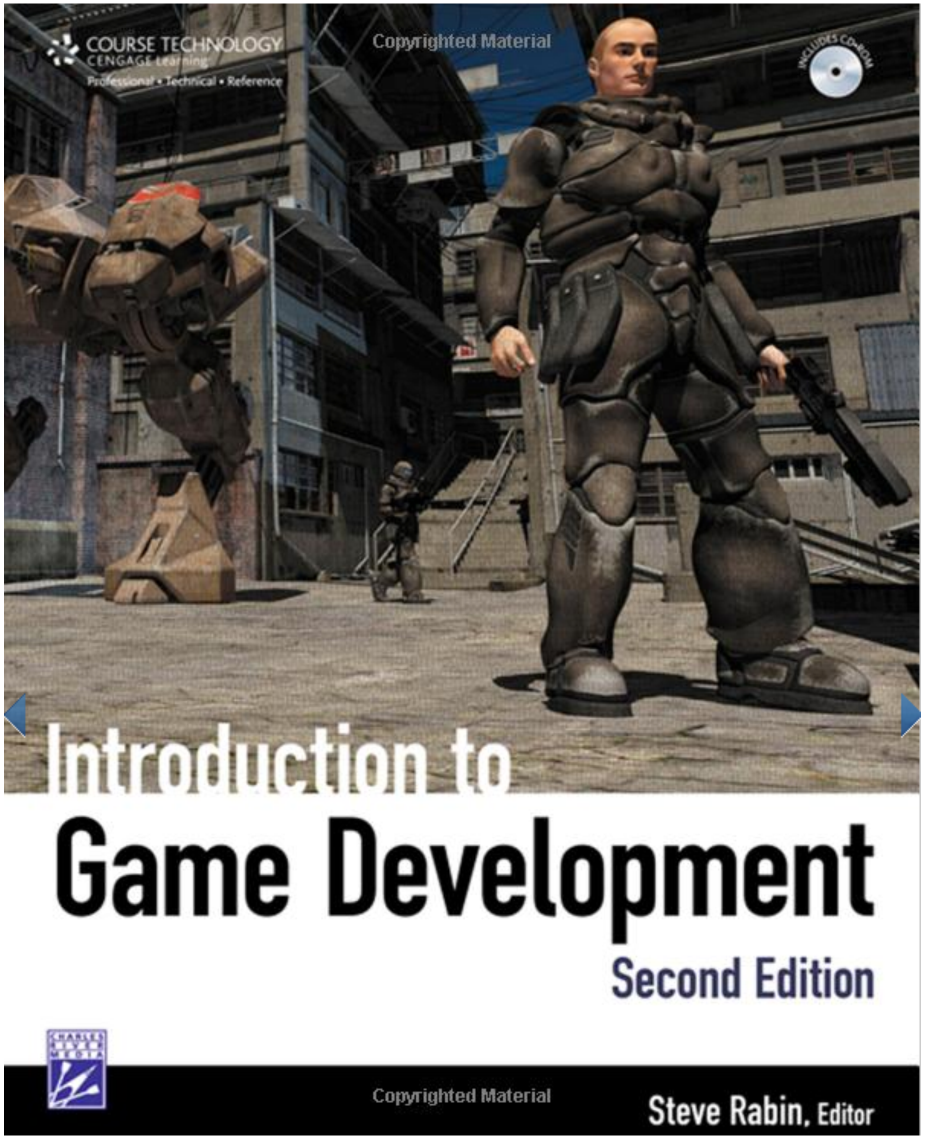
Game Architecture

- Rabin is a good overview of everything to do with Games
- A lot of these slides come from the 1st edition



Game Architecture

- The code for modern games is highly complex
  - Code bases exceeding a million lines of code
- Many commonly accepted approaches
  - Developed and proven over time
  - Ignore them at your peril!
- Globally optimized and balanced
  - Lots of very smart folks work on each of ’em

Overall Architecture

- Main structure
  - Game-specific code
  - Game-engine code
  - Level of integration varies
- Architecture types
  - Ad-hoc (everything accesses everything)
  - Modular
  - DAG (directed acyclic graph)
  - Layered

Overview: Initialization/Shutdown

- The initialization step prepares everything that is necessary to start a part of the game
- The shutdown step undoes everything the initialization step did, but in reverse order
- This is IMPORTANT
  - Applies to main loop, down to individual steps
  - In Unity:
    - Start/Awake
    - OnEnable/OnDisable
    - OnLevelWasLoaded/OnApplicationQuit

Overview: The Main Loop

- All interactive programs are driven by a loop that performs a series of tasks every frame
  - GUI, 3D, VR, Simulation
  - Games are no exception
- Separate loops for the front end and the game itself, or unified main loop
  - Both work; a question of preference and style

Overview: Main Game Loop

http://wiki.unity3d.com/index.php?title=Event_Execution_Order

- Tasks
  - Handling time
  - Gathering player input
  - Networking
  - Simulation
  - Collision detection and response
  - Object updates
  - Rendering
  - Other miscellaneous tasks

Overview: Main Game Loop

- Coupling
  - Can decouple the rendering step from simulation and update steps
  - Results in higher frame rate, smoother animation, and greater responsiveness
    - May be necessary for complex simulations
  - Implementation is tricky and can be error-prone
    - Co-routines can help, but aren't panacea

Overview: Main Game Loop

- Execution order
  - Can help keep player interaction seamless
    - Avoid “one frame behind” problems
  - Can maximize parallelism
  - Exact ordering depends on hardware

Game Entities

- What are game entities?
  - Basically anything in a game world that can be interacted with
  - More precisely, a self-contained piece of logical interactive content
  - Only things we will interact with should become game entities

Game Entities

- Organization
  - Simple list
  - Multiple databases
  - Logical tree
  - Spatial database

Game Entities

- Updating
  - Updating each entity once per frame can be too expensive
  - Can use a tree structure to impose a hierarchy for updating
  - Can use a priority queue to decide which entities to update every frame

Game Entities

- Object creation
  - Basic object factories
  - Extensible object factories
  - Using automatic registration
  - Using explicit registration
- Identification (pointers vs. uids)
- Communication (messages)

Game Entities

- Level instantiation
  - Loading a level involves loading both assets and the game state
  - It is necessary to create the game entities and set the correct state for them
  - Using instance data vs. template data

Memory Management

- Only applies to languages with explicit memory management (C or C++)
- Memory problems are one of the leading causes of bugs in programs
  - Or, "Reason 437 why I dislike C++"

Memory Management

- Chapter in "Introduction to Game Development" (Steve Rabin) is good
  - E.g., avoiding memory fragmentation
- Custom memory managers are great!
- Two most important reasons:
  - Simple error-checking schemes
  - Debugging tools
- Engines (e.g., Unity, C4, etc) handle much of this for you

File I/O

- As with memory, Rabin book gives lots of good advice on how to deal with loading things from disk
  - E.g., to avoid long load times
- Aside from efficiency, keeps things together!
- Unity handles much of this already
  - For assets in your project
  - No great support for access to other files

Game Resources

- A game resource (or asset) is anything that gets loaded that could be shared by several parts of the game
  - A texture, an animation, a sound, etc
- We want to load and share resources easily
- There will be many different types of resources in a game

Game Resources

- Resource manager
  - Uses registering object factory pattern
  - Can register different types of resources
  - All resource creation goes through the resource manager
  - Any requests for existing resources don't load it again

Game Resources

- Resource lifetime
  - If resources are shared, how do we know when we can destroy them?
    - All at once
    - At the end of the level
  - Explicit lifetime management
  - Reference counting

Game Resources

- Resources and instances
  - Resource is the part of the asset that can be shared among all parts of the game
  - Instance is the unique data that each part of the game needs to keep

Serialization

- Every game needs to save and restore some game state
- Level editing and creation could be implemented as a saved game
  - Many tools use this approach to create game levels
  - E.g., Nebula2 uses a simple database
- For you, may also be worth doing