Principles of Computer Game Design and Implementation

Lecture 3

Acknowledgement

- All of the materials of this module are inherited from Prof. Boris Konev.

We already knew

- Introduction to this module
- History of video games
- High-level information for a game (such as Game platform, player motivation, game structure, player-game model, character archetype, game genres

Outline for Today

- Overall architecture
- Game structure
- scripting language
- Game engine
- Programming language

Game Architecture

More than Code

- Until the 1980s programmers developed the whole game (and did the art and sounds too!)
- Now programmers write code to support designers and artists (content creators)
- The code for modern games is highly complex
- With code bases exceeding a million lines of code, a well-defined architecture is essential

History

- Initially, games were written as a monolith entity
  - Ad-hoc manner
  - Low-level programming languages (Assembly, C)
    - Low resource requirements
    - Atari 2600 VCS only had 4K memory for the entire game!
  - Rapid development of hardware lead to poor code reuse

History

- id Software games (Doom and Quake) were so popular that other developers preferred to licence their 3D manipulation code rather than develop it from scratch
- Leads to a better design in computer games

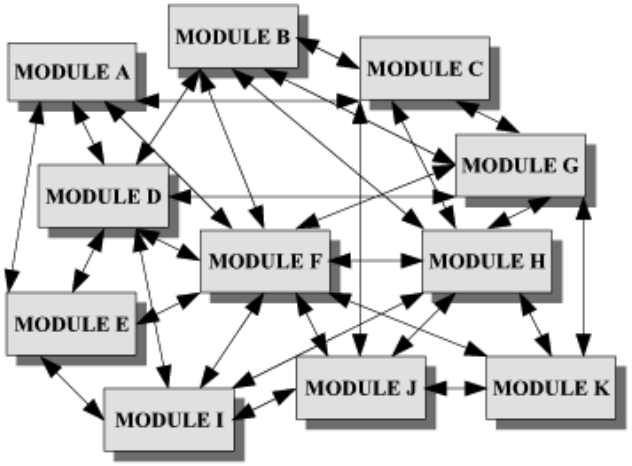
Overall Architecture: Ad-hoc

- No organisation
- Code grows “organically”
- Subsystems not identified
  nor isolated
- Works for small projects

(used in the past also for efficiency)

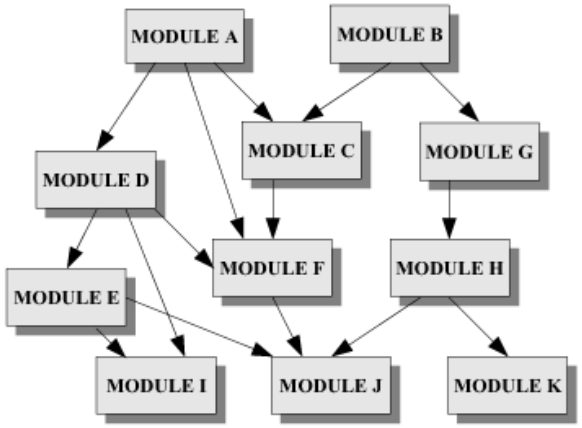
Overall Architecture: Modular

- Subsystems clearly isolated
- Well-defined module interfaces
- Reuse and maintainability
- Dependencies between modules are not controlled



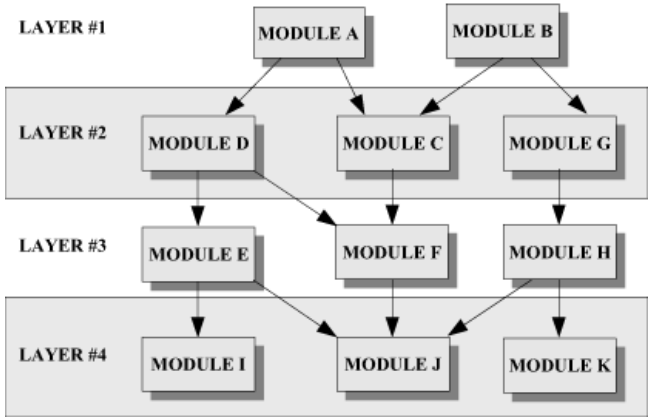
Overall Architecture: DAG

- Modular + no cycles
- Classify modules
  - Higher-level
    - E.g. Game-specific code
  - Lower-level
    - E.g. Platform-specific code



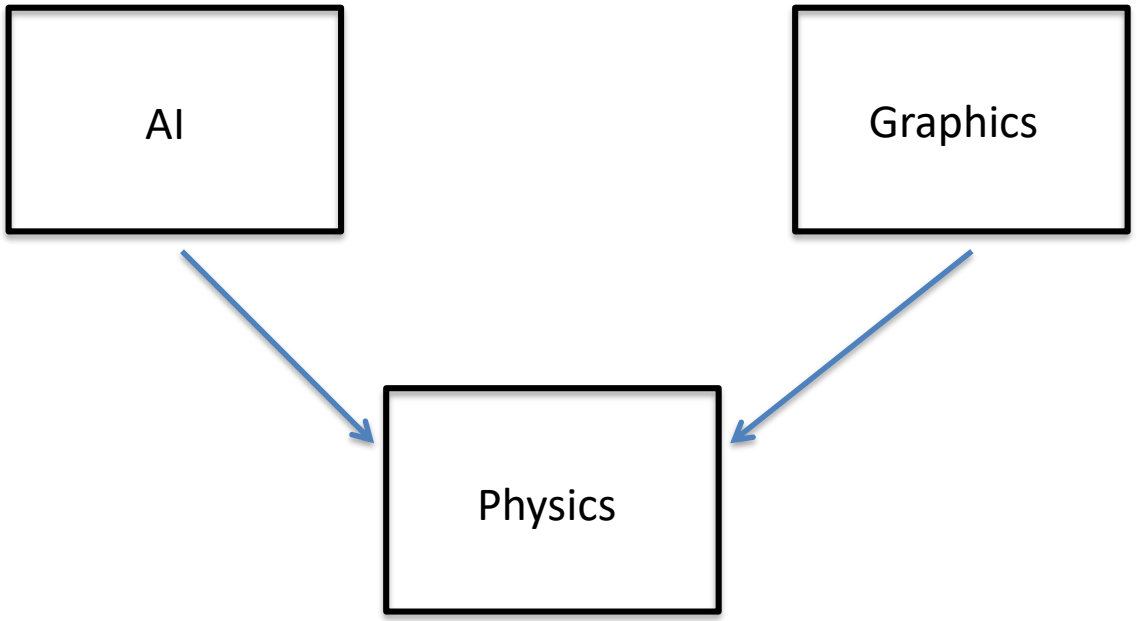
Overall Architecture: Layered

- Rigid layers
  - Can only interact with modules directly below
  - Can lead to code duplication
    - Give MODULE A access to MODULE I
  - Improves portability and best for code reuse



Perils of Modular Architecture

- We want something like this



- We want something like this for *this* game
  - No silver bullet

Game Subsystems

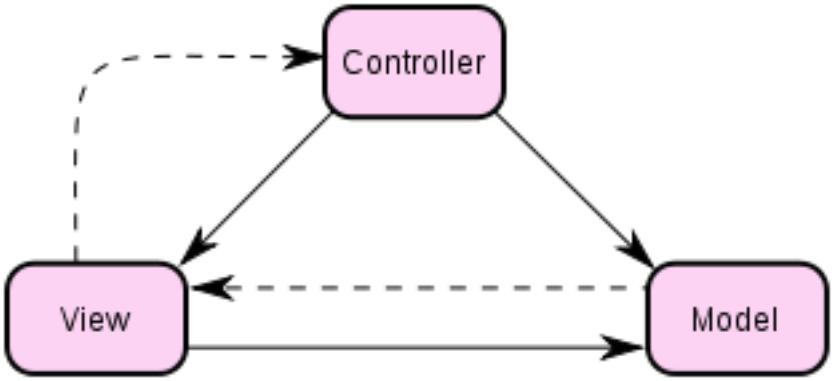
- Input
- Networking
- Rendering
- Sound
- Script
- Loading
- Front-end
- HUD
- Physics
- AI/Gameplay

Ideally, we wan them to be as independent as possible

- Each system as a black box with controlled communication
- But…
  - Renderer, Physics, Networking, sound, AI all need positions of objects

Inspiration: MVC Pattern

- In business applications, Model-View-Controller design pattern is quite popular



- Model: data
- View: UI
- Controller: links the two

- World model
- Graphics
- Game Engine

Game State

- A collection of information that presents the state of game entities in a particular moment
  - Position, orientation, velocity
  - Behaviour, intentions, …
  - Geometry

- Putting it all together (global state) may not be a good idea

Game Structure

Large Projects

- Game code
  - Anything related directly to the game
- Game engine
  - Any code that can be reused between different games
- Tools
  - In house tools
  - Plug-ins for off-the-shelf tools

Game Code

- Everything directly related to the game
  - Camera behaviour
  - Characters
  - AI entities
  - Choices
  - …
- C, C++, but increasingly *scripting languages* used

Scripting Languages

- Why use scripting languages?
  - Ease and speed of development
  - Short iteration time
  - Code becomes **a game asset**
  - Offer additional features and are customizable
  - Can be mastered by artists / designers

Scripting Languages

- Drawbacks
  - Slow performance
  - Limited tool support
  - Dynamic typing makes it difficult to catch errors
  - Awkward interface with the rest of the game
  - Difficult to implement well

Scripting Languages

- Popular scripting languages
  - Python
  - Lua
  - Custom scripting languages
    - UnrealScript, QuakeC, NWNScript

Game Engine

- To isolate game from hardware
- To encourage code reuse
- To simplify game development

- Tasks:
  - Rendering (2D or 3D), physics, sound, animation, networking
  - AI
  - Interface to game code

C++

- Initially, there was no alternative to the Assembly language (performance, resources)
- Then, C became the most popular language for games

- Today, C++ is the language of choice for game development especially in game engines

C++: Strengths

- Performance
  - Control over low-level functionality (memory management, etc)
  - Can switch to assembly or C whenever necessary
  - Good interface with OS, hardware, and other languages

C++: Strengths

- High-level, object-oriented
  - High-level language features are essential for making today's complex games
  - Has inheritance, polymorphism, templates, and exceptions
  - Strongly typed, so it has improved reliability

C++: Strengths

- C Heritage
  - C++ is the only high-level language that is backwards-compatible with C
  - Has APIs and compiler support in all platforms
  - Easier transition for experienced programmers

C++: Strengths

- Libraries
  - STL (Standard Template Library)
    - Comprehensive set of standard libraries
  - Boost: widely used library with wide variety of functionality
  - Many commercial C++ libraries also available

C++: Weaknesses

- Too low-level
  - Still forces programmers to deal with low-level issues
  - Too error-prone
  - Attention to low-level details is overkill for high-level features or tools

C++: Weaknesses

- Too complicated
  - Because of its C heritage, C++ is very complicated
  - Long learning curve to become competent with the language

Java for Game Development

- Why use Java?
  - It's a high-level OO language that simplifies many C++ features
  - Adds several useful high-level features
  - Easy to develop for multiple platforms because of intermediate bytecode
  - Good library support

Java for Game Development

- Performance
  - Has typically been Java's weak point
  - Has improved in the last few years: still not up to C++ level, but very close
  - Uses Just-In-Time compiling and HotSpot optimizations
  - Now has high-performance libraries
  - Also has access to native functionality

Java for Game Development

- Platforms
  - Well suited to downloadable and browser-based games
  - Dominates development on mobile and handheld platforms
  - Possible to use in full PC games
    - More likely to be embedded into a game
  - Not currently used in consoles

Java for Game Development

- Teaching Java game development
  - Java is taught to all our students
  - We can concentrate on game development issues rather than on the study of a new language
  - Knowledge can be used in broader context

We will use a Java game engine,

jMonkeyEngine