IMGD 1001 - The Game Development Process: Intro to Programming

by

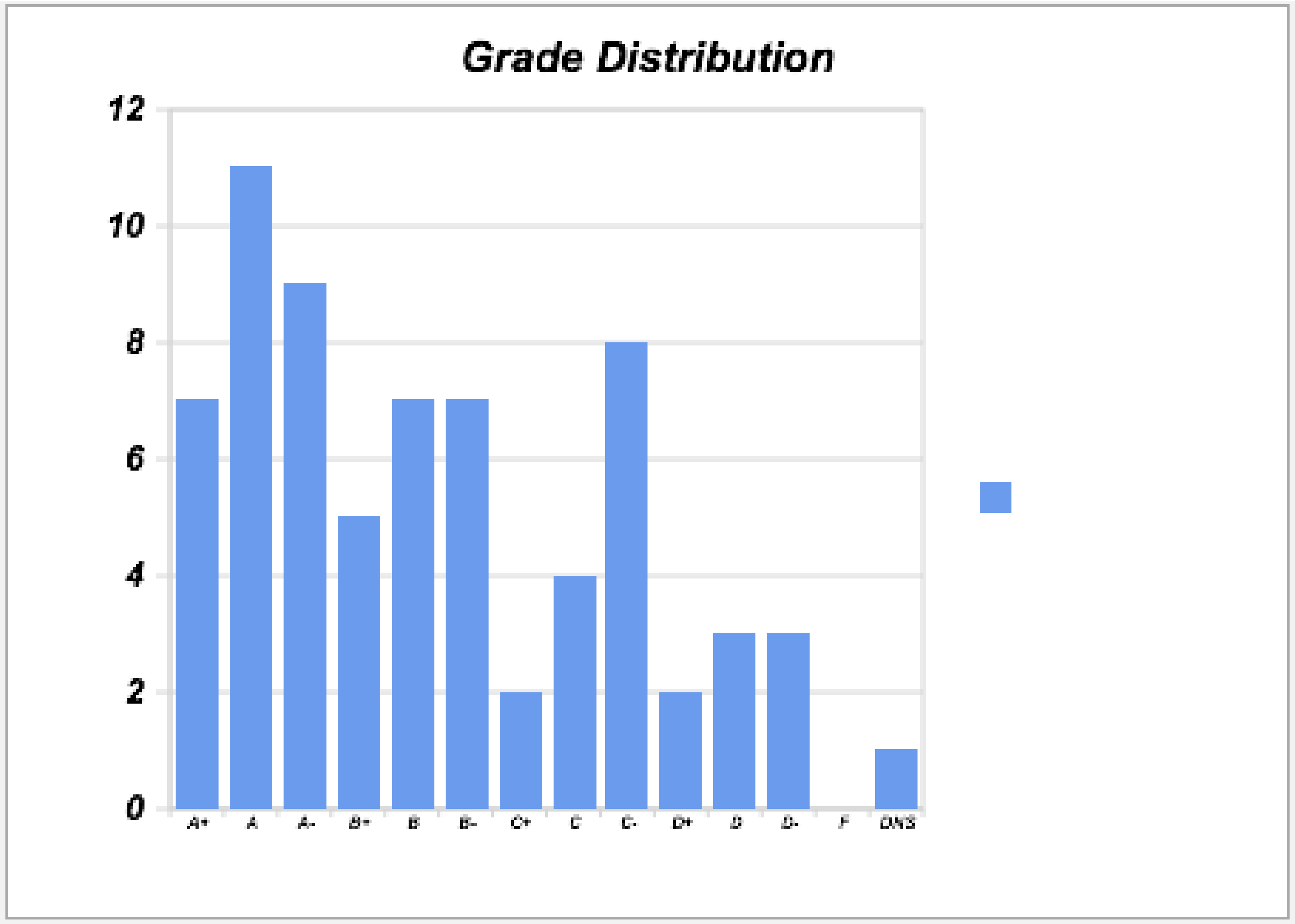
Robert W. Lindeman (gogo@wpi.edu)

Kent Quirk (kent_quirk@cognitoy.com)

(with lots of input from Mark Claypool!)

Exam

- Flow
- Functional Symmetry
- Keyframes
- Handedness



Intro to Game Programming

- What is it?
- Types of programming
- Language survey
- Categories of languages

Back in the day…

- Games were created by one or two programmers in a garage
  - They didn't necessarily know how to make good games
    - Exceptions: Wright, Pajitnov, Meier
- Now, programmers make systems
  - Designers and artists make the content
- Except casual / mobile
  - But even there most of the successful companies are teams

Areas of Specialization

- Engine
  - Architecture
  - Physics
  - UI / Interaction
- Network
- Graphics
- AI
- Scripting / Level design
- Tools

Engine programming

- The platform that runs the game
- It's a system, requires high-level and low-level thinking (architecture)
  - What does an architect do?
- Integrates Physics and provides the UI operating environment
- Usually C++ (why?)
- Key background: Software Engineering

Networking

- A specialization of its own
- Includes multitasking and scalability
- Server side and client side
- Can be hugely complex
  - Particularly for MMOs
- Key background: Computer Science

AI / Scripting / Level Design

- AI is its own subspecialty
  - Again, CS is valuable
  - But often reinvented by non-CS people
    - Not very good
    - But - that might not be bad! Sometimes gameplay is better for simpler AI
      - People are easily fooled
- Sometimes coded by the designers
- Often done in a scripting language or something easily tweaked and tuned

Tools

- Many games need tools for production
  - Sometimes in-house only
  - Sometimes also shipped to customer for mods
- Just-in-time programming, often.
- Scripting language, batch files, whatever’s at hand
  - Skimping on tools can cost you a lot!
  - People are a lot more expensive than software
    - Even expensive software
    - Not always true for students and startups

Generalists

- Valuable to have someone who knows a little bit of everything
- They'll integrate and cross-pollenate
- But too many of them can lead to chaos
- General rule:
  - Specialize for a while, but "sharpen the saw" from time to time.

Survey of key programming languages

- C++
- Java
- Scripting Languages
- Flash

C++ (1 of 3)

- Until mid ‘90s, C was the systems programming language of choice
  - But it wasn’t “Object-oriented” and didn’t scale well to larger projects
- C++ created to take C to the next level
- Calling it “A better C” is too limiting
  - C is a well-tuned bicycle
  - C++ is a large tractor-trailer
    - With a sleeper cab
    - Filled with tools

C++ (2 of 3)

- Supports large scale programming with:
  - Strong typing
  - Objects
  - Exceptions
  - Cross-platform toolset
  - Templates
  - Metaprogramming
- Industry standard
  - Everyone uses it
    - Few use it well -- it’s just too big

C++ (3 of 3)

- Many libraries available (middleware)
  - OpenGL
  - DirectX
  - Standard Template Library
  - Game Engines
  - Video / Audio tools

C++ (Summary)

- When to use?
  - Any code where performance is crucial
    - Used to be all -- now game engine such as graphics and (sometimes) AI
    - Game-specific code is often not C++
  - If you have a legacy code base, expertise
  - If your middleware libraries expect it
- When not to use?
  - Tool building (GUIs are tough)
  - High-level game tasks (technical designers)

Java (1 of)

- Basically, created to be the Object-oriented language for the web
  - Designed by theorists
  - Sometimes gives short shrift to practicality
- Very portable
  - "Write once, run everywhere"
    - In reality: Write once, debug everywhere
  - From desktops to cellphones

Java (2 of 3)

- Concepts from C++
  - But cleaner
  - Abstract away the hardware and many of the standard bugs
    - Memory management
    - Simpler templates
    - Introspection
  - Portability a huge design feature
  - Performance sometimes a problem
    - Virtual machine, JIT compiler
    - 2-10x slower (who cares?)

Java (3 of 3)

- Only recently useful for games
  - Cell phone games
  - Web games
  - *Project Darkstar* from Sun
  - Java 3D
- Used in:
  - *Star Wars Galaxies*
  - *You Don't Know Jack*
  - Cell phone games
  - Lots of server-side stuff

Scripting Languages

- Really means "Languages you don't have to compile first"
  - Kind of a slam
  - In 1990 there was a huge difference between compiled and "interpreted" languages
    - Modern technology has blurred it all
- Many (most) games use one
  - Use one once you find your data starts getting smart.
  - You need one if your data file wants to do:
    `center = (left + right) / 2`

Scripting Languages (2)

- Can get very powerful
  - Entire UI systems
  - AI and level design
- If done right, provides a nice separation of engine and gameplay
- Easier to program for game and level designers
  - But you probably still need professional developers to design the big picture.
- Fast iterations!

Scripting Languages (3)

- Code can become an asset
  - Edited / modified as part of content
- Performance can be an issue
  - Scripting systems vary wildly
  - Be smart about it
- Tools may be weak
  - But you don’t need them as much
- Interface to game needs maintenance

Scripting Languages: Python

- Object-oriented ("OO")
- Large(ish) memory hit
- Many tools, growing population of programmers knows it
- You can write whole games in it
  - PyGame
- Integrates well, with effort
- *Blender* (tool), *Eve Online, Civ 4, Cosmic Blobs*

Scripting Languages: Lua

- ("loo-uh")
  - Small, C-like
  - Not OO
  - Really easy to embed
  - Popular choice -- but limits your capabilities
    - Doesn't scale well to large systems
  - *Grim Fandango, Far Cry, Baldur's Gate*

Scripting Languages: Other

- Ruby, Perl
  - Save ‘em for the web - they don’t embed well
- Can use Java as embedded language
- JavaScript / ECMAScript is better
- .NET / Mono
- Home Grown
  - Just say no -- It’s harder than it looks and really hard to make a good one
  - Exception if it’s <u>really</u> specialized - a *Domain-Specific Language*

Scripting Languages: Flash

- Flash is the authoring tool (IDE), the player, the application files
- Advantages
  - Wide audience (V8 - 98%, V9 - 93%)
  - Great for downloadable games
  - Rapid development, esp. for artists
- Disadvantages
  - Lousy for big systems
  - Performance poor before V9
  - Grown, not designed -- programmers cry

More Flash

- Timeline-based system
  - Objects located in space and time
  - Attach scripts to objects and events
- Vector-based graphics
  - Infinitely scalable
  - Can be very fast
- Programming language
  - OO after version 8 (ActionScript 2)
  - Version 9 MUCH faster (AS3)
    - But big changes in language

Language categories (1 of 2)

I. **Low-Level:** Assembly, GLSL
II. **System / Structured:** C, some BASIC
III. **Object-oriented:** C++, Java, BASIC, D
IV. **Dynamic:** Python, Ruby, Perl, ActionScript, Javascript
V. **Functional:** Lisp, OCaml, Haskell, Scheme

C++ can fit almost anywhere!

Language categories (2 of 2)

- Easy to switch within a category -- more work to step across categories; paradigm shift required.
- Categories II and III easiest to learn and teach
- Categories I, IV require paradigm shift
- Category V requires mental gymnastics

How to choose?

- Expertise matters…but not TOO much
  - A good developer can easily pick up new languages in the same class as the old ones
- Interface to other tools, middleware
- Performance matters
  - But not as much as most people think
  - Your performance instincts are probably wrong
- Developer performance matters most
  - Time is money

Building software

- It's hard
- The bigger the system, the harder it gets
- It's not asymptotic -- some systems appear to be literally impossible to build
  - Air traffic control
- Fred Brooks, The Mythical Man-Month
  - "Adding resources to a late software product makes it later"

Methodologies

- A $100 way of saying “Methods”
- A collection of policies and procedures for attempting to get control over software development
- They have names:
  - Code and Fix
  - Waterfall
  - Spiral
  - Agile

Methodologies: Code and Fix

- Really means "We have no methodology"
- All too common
- Little planning, straight to implementation
- Reactive, not proactive
- End with bugs
  - If you add bugs faster than you fix them, "death spiral"
  - Generates crunch time ("EA Spouse")

Methodologies: Waterfall

- Plan the whole project first, then do it
  - Requirements
  - Design
  - Implementation
  - Testing
  - Integration
  - Maintenance
- Fragile when requirements can change
  - Hint: They ALWAYS change

Methodologies: Spiral

- Modified waterfall, but in smaller bites
  - Only tackle the part you can see clearly
  - Sometimes gets stakeholders nervous because dates are hard to predict
    - Hint: dates are always hard to predict
  - Sometimes different pieces will be at different stages (planning the AI while implementing the engine, for example)

Methodologies: Agile

- Goal: get the stakeholders involved in the creation process
  - Customers drive the features and the progress
  - Admit you have no control, proceed day by day
  - Great for feature-driven products
  - Can be tough for games -- where's the design?