

11/9/11

**TAGD**

# This Week's Question

- ◎ This week, we had a pair of questions from some twin TAGD members who seem to have a question connection:

*I would like to know about different programming languages and how useful they are for games.*

*I would like to know about different game engines and how they are used in games.*

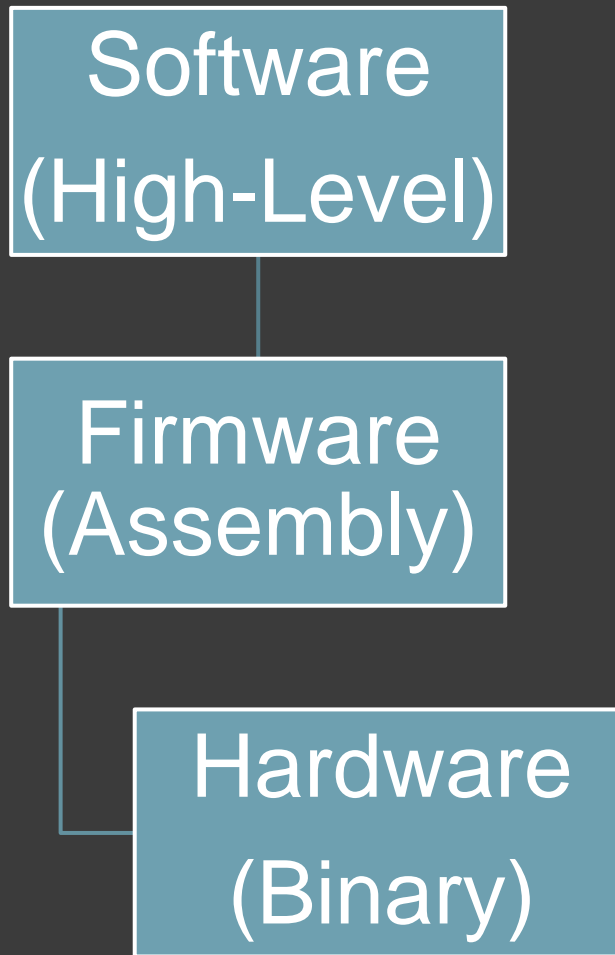
- So today we're going to talk about both of these, as well as how they are related.

# Abstraction

- ⦿ Abstraction is one of the basic tenets of computer science.
- ⦿ Software is built in layers – each making our job easier.
  - Shields us from unnecessary implementation details.
  - Provides structures that help us solve problems.

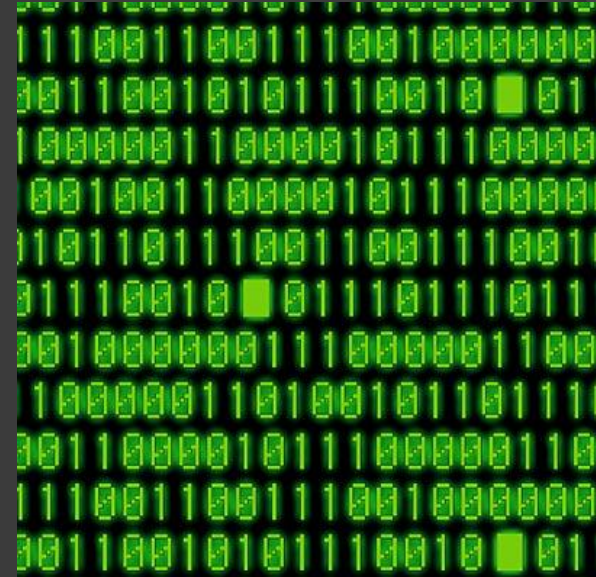
# Language Abstraction

- Basic idea at each level is to make it easier to perform actions we do often.
- There are two main tiers:
  - Low-Level
  - High-Level



# Low-Level Languages

- Binary is the most basic language, as well as the most difficult.
  - Hasn't been used since the '60s.
- Assembly simplifies the process by giving us distinct commands to use.
  - Usually directly related to available processor commands.



```
00000000 push ebp
00000001 mov ebp, esp
00000003 movzx ecx, [ebp+arg_0]
00000007 pop ebp
00000008 movzx dx, cl
0000000C lea eax, [edx+edx]
0000000F add eax, edx
00000011 shl eax, 2
00000014 add eax, edx
00000016 shr eax, 8
00000019 sub cl, al
0000001B shr cl, 1
0000001D add al, cl
0000001F shr al, 5
00000022 movzx eax, al
00000025 retn
```

# High-Level Languages

- ⦿ There's a problem with low-level languages.
  - They're highly efficient, but totally unreadable.
- ⦿ We would like to be able to read our code.
  - You remember that we talked about this, right?
  - We also have more processing power than we need.
  - So, at the cost of speed, we can create constructs that make programming easier to write and read.

# Types of High-Level Languages

- Logical
- Functional
- Procedural
- Object-Oriented

# Logical

- ⦿ Languages: Planner, Golux, R++, Prolog
- ⦿ In these languages, programmers do not explicitly define how the program will execute commands.
- ⦿ Instead, programmers give a series of rules to the program. The program solves queries given based upon the rules.
- ⦿ Not used for gaming, except in rare cases for AI.

# Functional

- ⦿ Languages: FP, FL, Haskell
- ⦿ Idea is to work on programs as mathematical objects.
- ⦿ Pure versions don't have variables at all, just functions.
- ⦿ Why is this good?
  - No side-effects from functions.
- ⦿ However, these are more or less totally unusable for games – at least for now.
  - Some languages are adding functional parts.

# Imperative

- ⦿ Languages: FORTRAN, COBOL, BASIC
- ⦿ The basis of high-level programming.
- ⦿ Separate statements into blocks of code.
- ⦿ Use switch, goto, and procedure calls to switch between blocks.
- ⦿ Was used for games in “the early days.”
  - Basis for what we use today.
  - Could still use it, if you wanted.

# Object-Oriented

- ⦿ Languages: C++, C#, Java
- ⦿ Extend imperative languages by adding a new block – the object.
  - Allows us to restrict variables and functions so that people can't call them.
  - Allows us to create logical relations between objects.
- ⦿ Dominant language for game making.

# Importance of Language Types

- ◎ Different language types require different types of thinking.
  - Using a new one, while initially frustrating, can help you learn to think differently about how you structure programs.
- ◎ Different problems can be solved better in different languages.
  - Knowing multiple languages will help you write better programs.

# Frameworks and Engines

- ⦿ These build upon languages to assist us in writing programs toward a goal.
  - They don't simplify statements we make often.
  - They prebuild components you are likely to use.
- ⦿ However, unlike full languages, this restricts what we can make.

# Frameworks and Engines

- ⦿ Frameworks provide structure we can use to write a program.
  - XNA has a GameComponent class
    - Can build off of it for AI, Game Mechanics, Graphics, and Sound.
- ⦿ Engines provide the components prebuilt.
  - We only have to use them to finish the program.
  - Unity provides the graphics, sound, and everything, just add game mechanics.

# Confused?

- If you still aren't sure of the difference, here's an analogy:
- Imagine we would like a house.
- We can start with just some tools and materials. (language)
- We can start with premade plans and pieces of the house.(framework)
- We can buy a premade house, and furnish it with our own stuff.(engine)



# Choosing an Engine

- When choosing an engine, care must be taken.
  - Since everything is premade, you need to match the capabilities of the engine with your game.
    - Systems that you can export the game to
    - Graphics supported (2D or 3D).

# Unity

- ◎ 3D engine
  - Some 2D support through libraries.
- ◎ Very easy to use
  - Multiple scripting languages
  - Drag-and-Drop
- ◎ Can make games for all kinds of systems



# Unreal

- Epic Games' engine.
- 3D only.
- Most used engine by game developers.
- Based in C++.
- Can write for PC, PS3, 360, and recently mobile.



# AndEngine

- 2D engine
- Extends OpenGL.
- For mobile platforms exclusively.



# Homework

- ⦿ Go try a different language for a couple of days.
  - See how much you don't like it
  - See what you learn.