CMSC201
Computer Science I for Majors

Lecture 11 – Program Design

Last Class We Covered

- Functions
  - Returning values
  - Matching parameters
  - Matching return assignments

Any Questions from Last Time?

Today’s Objectives

- To learn about modularity and its benefits
- To see an example of breaking a large program into smaller pieces
  - Top Down Design
- To introduce two methods of implementation
  - Top Down and Bottom Up

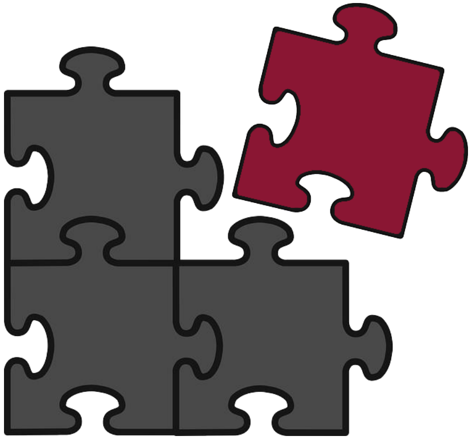
Modularity

Modularity

- A program being ***modular*** means that it is:
- Made up of individual pieces (modules)
  - That can be changed or replaced
  - Without affecting the rest of the system

- So if we replace or change one function, the rest should still work, even after the change

Modularity



- With modularity, you can reuse and repurpose your code

- What are some pieces of code you’ve had to write multiple times?
  - Getting input between some min and max
  - Using a sentinel loop to create a list
  - What else?

Image from pixabay.com

Functions and Program Structure

- So far, functions have been used as a mechanism for reducing code duplication
- Another reason to use functions is to make your programs more modular

- As the algorithms you design get increasingly complex, it gets more and more difficult to make sense out of the programs

Functions and Program Structure

- One option to handle this complexity is to break it down into smaller pieces

- Each piece makes sense on their own
- You can easily combine them together to form the complete program

Complex Problems

- If we only take a problem in one piece, it may seem too complicated to even begin to solve
  - A program that recommends classes to take based on availability, how often the class is offered, and the professor's rating
  - Creating a video game from scratch

Top Down Design

Top Down Design

- Computer programmers often use a ***divide and conquer*** approach to problem solving:
  - Break the problem into parts
  - Solve each part individually
  - Assemble into the larger solution
- One example of this technique is known as ***top down design***

Top Down Design

- Breaking the problem down into pieces makes it more manageable to solve
- ***Top-down design*** is a process in which:
  - A big problem is broken down into small sub-problems
    - Which can themselves be broken down into even smaller sub-problems
      - And so on and so forth…

Top Down Design: Illustration

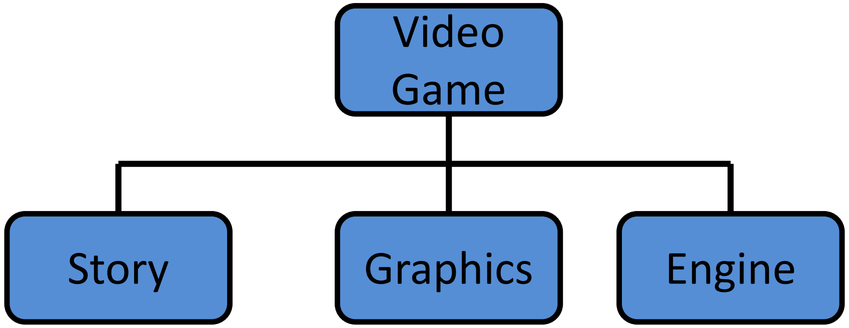
- First, start with a clear statement of the problem or concept

- A single big idea

Video Game

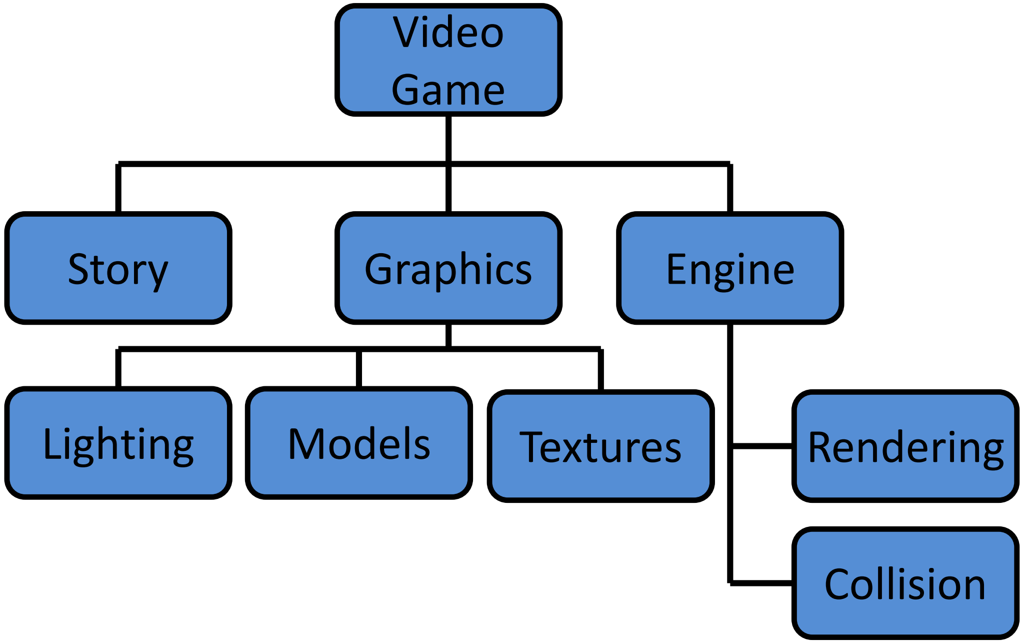
Top Down Design: Illustration

- Next, break it down into several parts



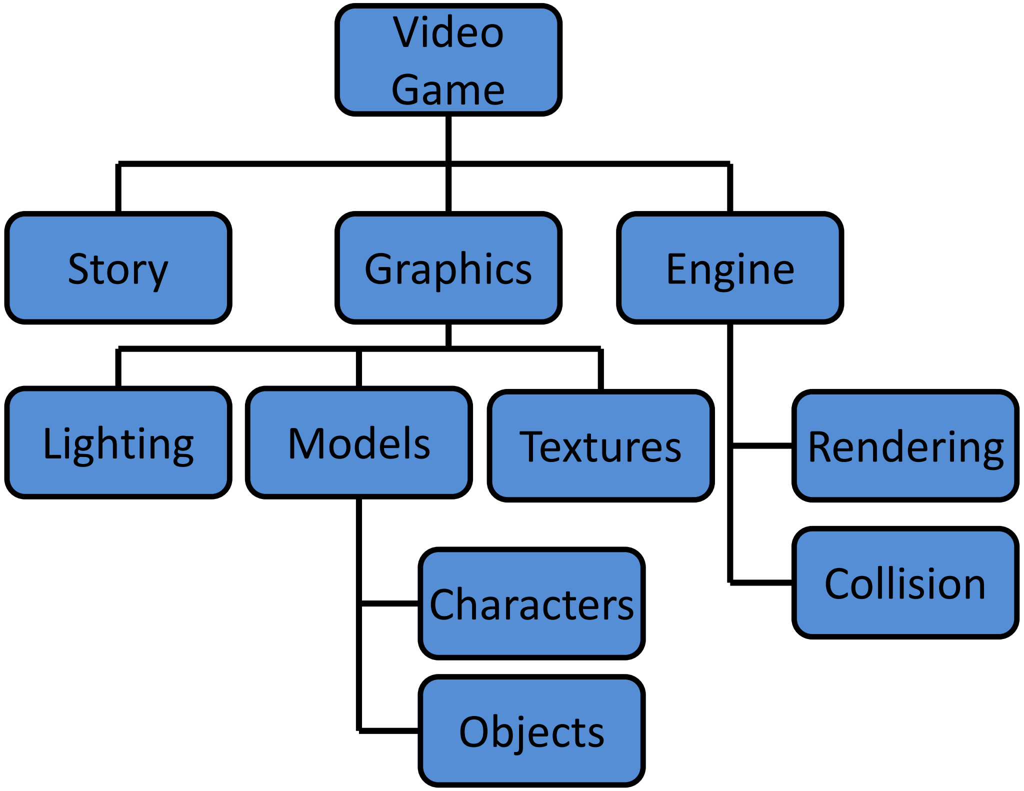
Top Down Design: Illustration

- Next, break it down into several parts
- If any of those parts can be further broken down, then the process continues…



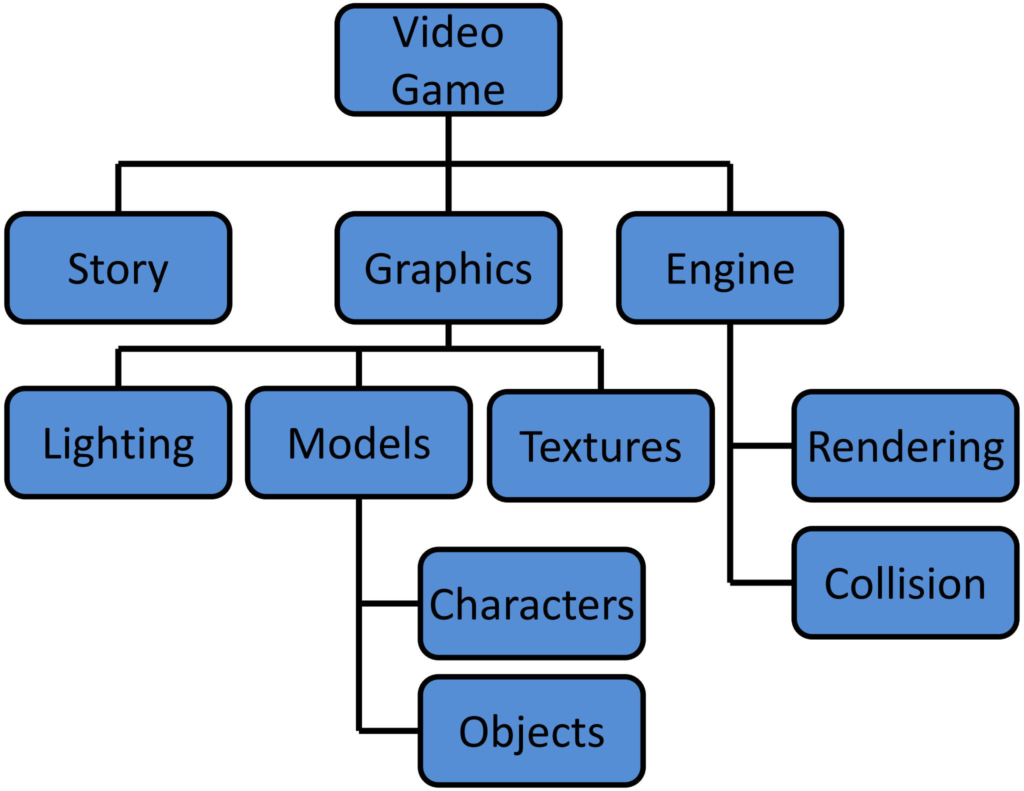
Top Down Design: Illustration

- And so on…



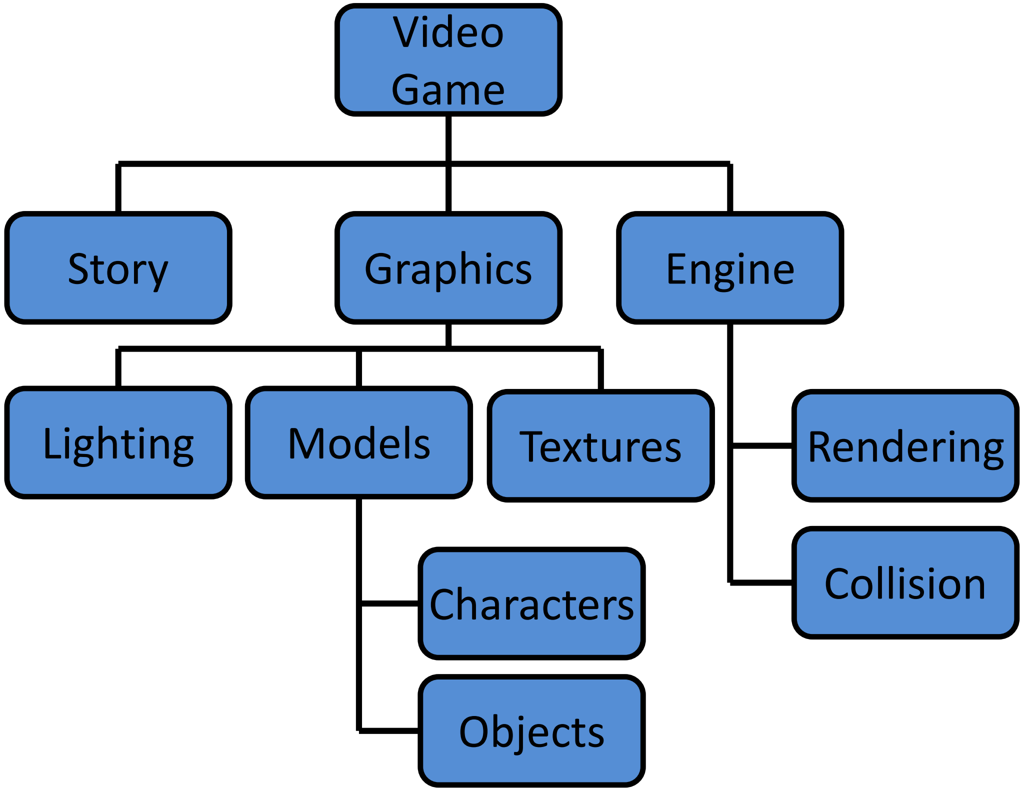
Top Down Design: Illustration

- Your final design might look like this chart, which shows the overall structure of the smaller pieces that together make up the “big idea” of the program



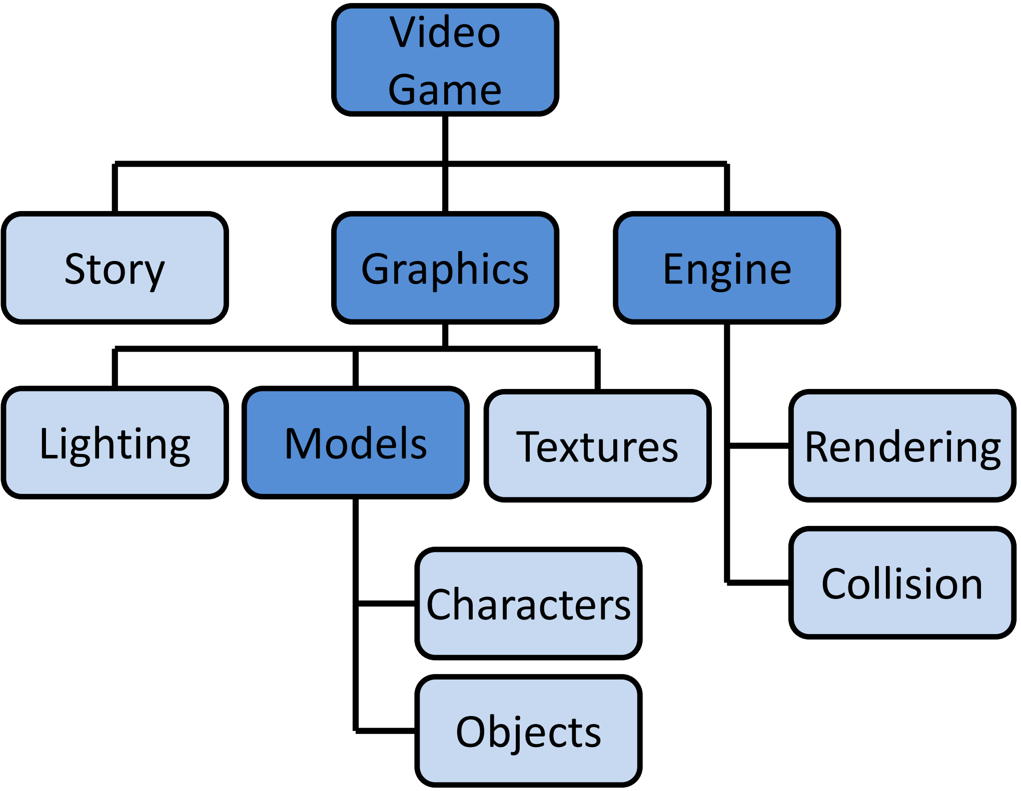
Top Down Design: Illustration

- This is like an upside-down "tree," where each of the nodes represents a single process (or a function)



Top Down Design: Illustration

- The bottom nodes are “leaves” that represent pieces that need to be developed
- They are then recombined to create the solution to the original problem



Analogy: Paper Outline

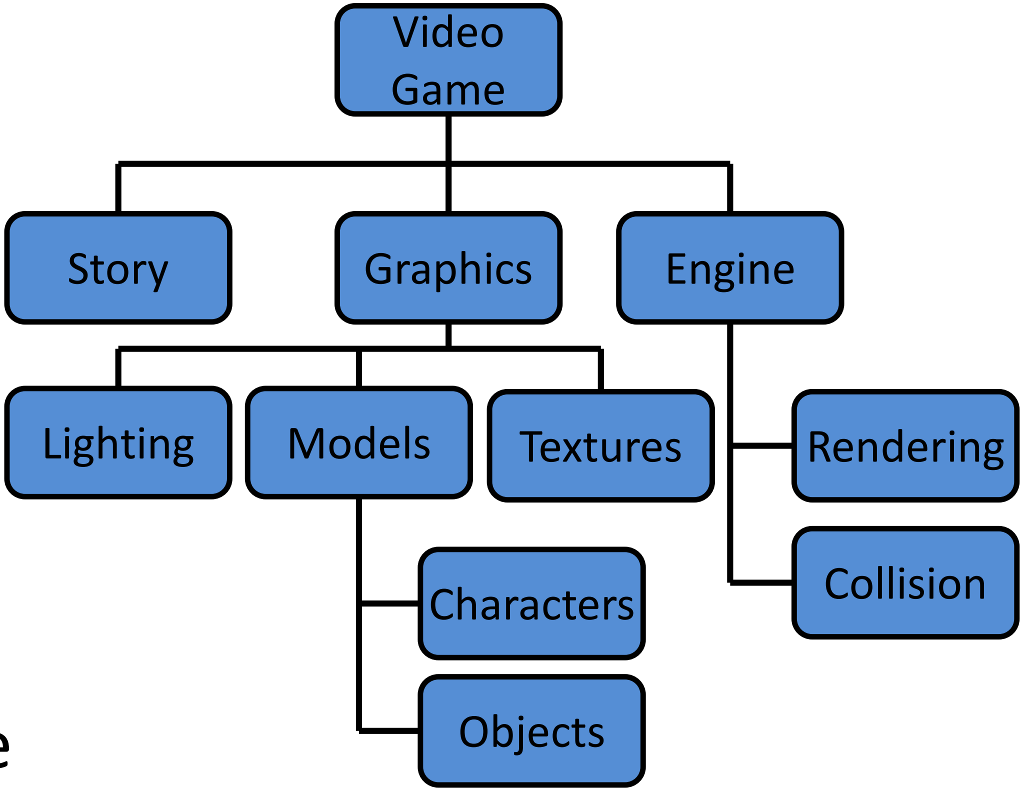
- Think of it as an outline for a paper you’re writing for a class assignment

- You don’t just start writing things down!
  - You come up with a plan of the important points you’ll cover, and in what order
  - This helps you to formulate your thoughts as well

Implementing a Design in Code

Bottom Up Implementation

- Develop each of the modules separately
  - Test that each one works as expected
- Then combine into their larger parts
  - Continue until the program is complete



Bottom Up Implementation

- To test your functions, you will probably use **`main()`** as a (temporary) test bed
  - You can even call it **`testMain()`** if you want
- Call each function with different test inputs
  - How does function ABC handle zeros?
  - Does this **`if`** statement work right if XYZ?
  - Ensure that functions “play nicely” together

Top Down Implementation

- Sort of the “opposite” of bottom up
- Create “dummy” functions that fulfill the requirements, but don’t perform their job
  - For example, a function that is supposed to take in a list of grades and return the average; it takes in the list, but then simply returns a 1
- Write up a “functional” **`main()`** that calls these dummy functions
  - Helps to pinpoint other functions you may need

Which To Choose?

- Top down? Or bottom up?

- It’s up to you!
  - As you do more programming, you will develop your own preference and style

- For now, just use something – don’t code up everything at once without testing anything!

Announcements

- Project 1 is out on Blackboard now
  - Must use the design provided in class
  - Design due by Saturday (March 11th)
  - Project due by Friday (March 17th) at 8:59:59 PM

- Midterm will be next week
  - We'll have an in-class review on Monday/Tuesday
  - Review worksheet only available <u>in class!</u>