Introduction to object oriented design

Procedural Programming vs OOP

Procedural Programming

- Top-down design
- Discrete functions performing small tasks
- Data transfer via function parameters, return values, global variables
- Procedures (functions) and Data
  - Procedures and data are separate (or loosely coupled)
  - Similar functions are collected together

Limitations of Procedural Programming

- Functions are usually global
- Functions are task specific, not data specific
  - There tend to be many similar functions which only differ by the type of data they are manipulating. This makes the code difficult to maintain
- Quickly increases in complexity
  - Adding new functionality becomes progressively more difficult as the project complexity increases
  - The code can become large and difficult to understand

Object Oriented Programming

- What if functions were tied to data and not the other way around?
- In object oriented programming code is structured around objects, their relationships, and their interfaces.

Example

- Consider objects needed for Space Invaders
  - Invaders
  - Players
  - Bullets
- What data and functionality is required by each object?
  - Data: Position, lives, speed, etc.
  - Functions: move, shoot, die, etc.

Objects (Classes)

- Bind together data and functions
- Objects have a declaration and definition (just like functions)
- An object is a programmer-defined complex data type.
  - C++ calls this a class.
  - Very similar to a struct but has more advanced functionality.

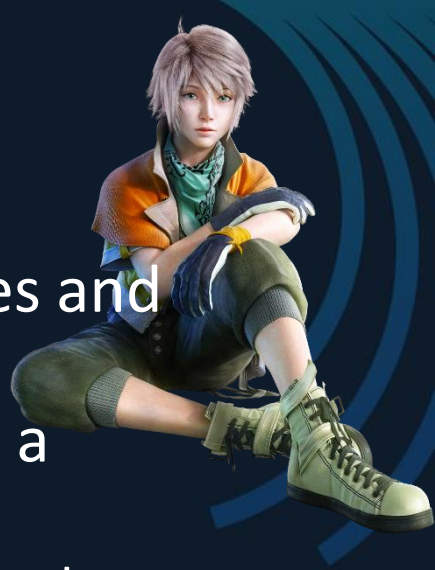
Objects, not functions

- OOP divides a program into Objects (not functions)
- Objects are:
  - Any entity involving data and operations on that data
  - Any game component
    - Splash screens, menu, level manager, etc.
  - Doesn't need to be a tangible thing
    - Different behaviours for an NPC could each be class

Four main tenets of OO

- There are four buzzwords which are used to sum up what OO is about
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

Abstraction



- Abstraction is the process by which general rules and concepts are derived from specific examples
- For example imagine we want to write code for a character in our RPG game.
  - Our design might include concept art with amazing costume, hair etc.
  - But if the only thing which effects the behaviour of the NPC in the game is a set of stats and it's position in the world, that's all we have in our object - it is an abstract representation.

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Encapsulation and reusability

- Each object has its own set of variables and functions
  - These are all related to the purpose of the object.
- Objects are defined to be reusable
  - For example, a programmer might build a user menu from four object types: Button, image, sound, and text box
  - These four types can be used repeatedly to create a larger, more complex menu system

Encapsulation and interfaces

- An object might contain a great deal of functionality
  - But only a small amount of it is of interest to other objects
  - Encapsulation allows us to hide the stuff which is not of interest to the object from the outside world.
- We can limit how much access the outside world has to our object
  - This provides greater security and control

Interface Example

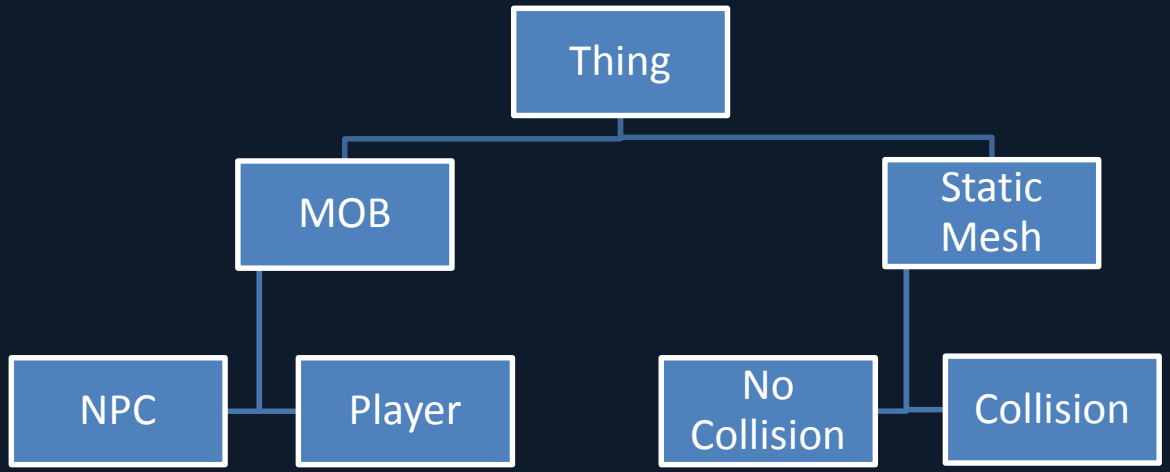
- We want to write code for an NPC in our game
  - The NPC has a variable which represents its health
  - When the NPC gets attacked, that health variable is reduced. If it reaches 0 the NPC dies
  - In OO we want to make the variable “private”, so that it can only be accessed by functions contained in the object
  - We produce a “public” function to decrement the health. The same function can check that when health reaches zero the NPC dies
- Encapsulation and interfaces make it easier to write bug resistant code

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Inheritance

- Objects can inherit from other objects.
- For example in our game world we might have the following hierarchy of objects:



Inheritance

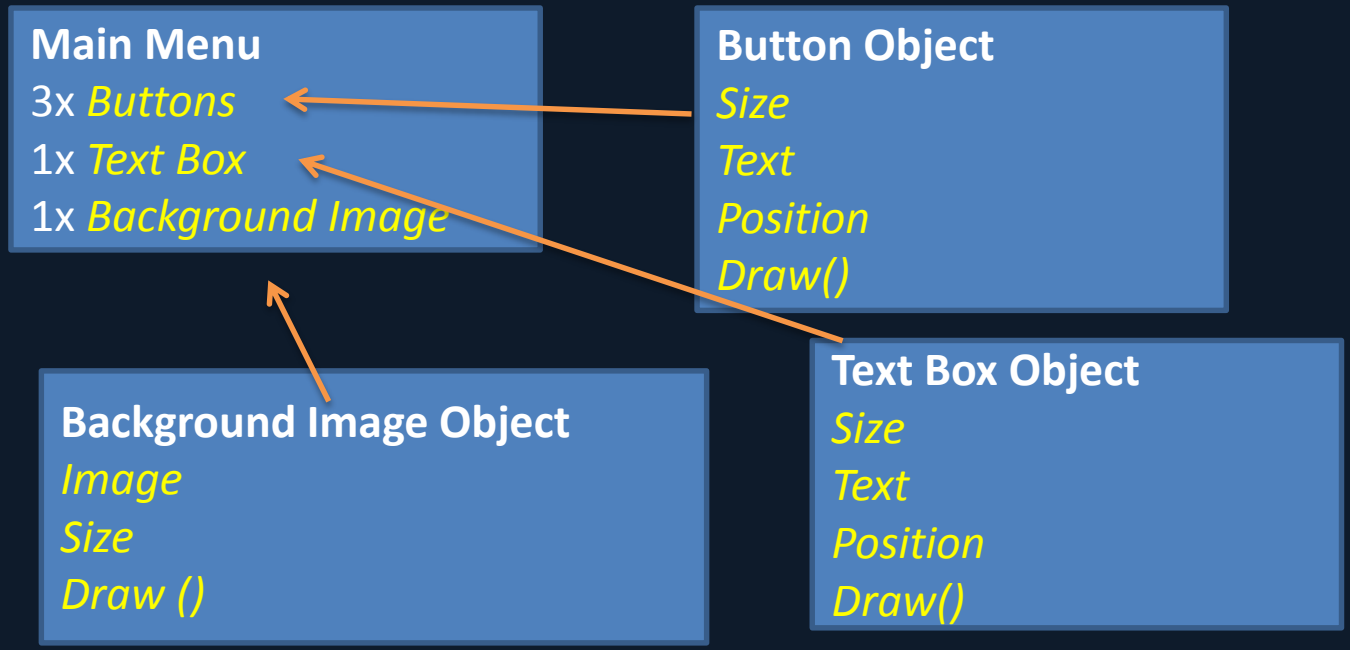
- Things – have a position in the world
  - MOB – have a controller to move them around
    - Player – takes input
    - NPC – has AI
  - Static meshes can have collision or not

Each of the objects inherits all the attributes of it's parent. We re-use functionality and hence reduce the amount of code we have to write (quicker and easier to maintain)

Objects can be hierarchical

- Objects can contain other objects
- Simpler objects combine into more complex ones.
- Similar to how structures can be put inside others

Menu Example



Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Polymorphism

- This is probably the trickiest OO tenet to understand
  - But you've already come across it before with *function overloading*
- For OO it's closely related to inheritance. We'll talk about how in detail later.

Summary

- Object oriented programming offers a number of advantages over the traditional procedural programming approach
- C++ calls its objects classes, which are basically structures with a lot more functionality.
- More on OOP to come!