Game Architecture and Planning

•••

ElectronicArmory.com
3D Game Development Course

Game Design Patterns

- Singleton
- Component
- Command
- Flyweight
- Observer
- Prototype
- State

- Game Loop
- Double Buffer
- Update Method
- Event Queue
- Object Pool
- Dirty Flag

Singleton

```
class InventorySystem
{
public:
  static InventorySystem& instance()
  {
          // Don’t initialize until needed
          if (instance_ == NULL){ instance_ = new InventorySystem(); }
          return *instance_;
  }

private:
  InventorySystem() {}

  static InventorySystem* instance_;
};
```

Component

Instead of subclassing our Character, weapons or other class, each time we want different functionality, we add components.

For example: A sword that fires ice or electricity. That can be added using a MagicProjectileComponent that you define.

Actors will have a list of component objects that it has attached to it.

They adhere to an interface that all components need to respect at a minimum.

Ex: StaticMeshComponent, SphereComponent

Command

An object-oriented method for callbacks.

Old way:

```
void FancyMoveToMethod(Vector3D vectorToMoveTo, &FunctionWhenDone);
```

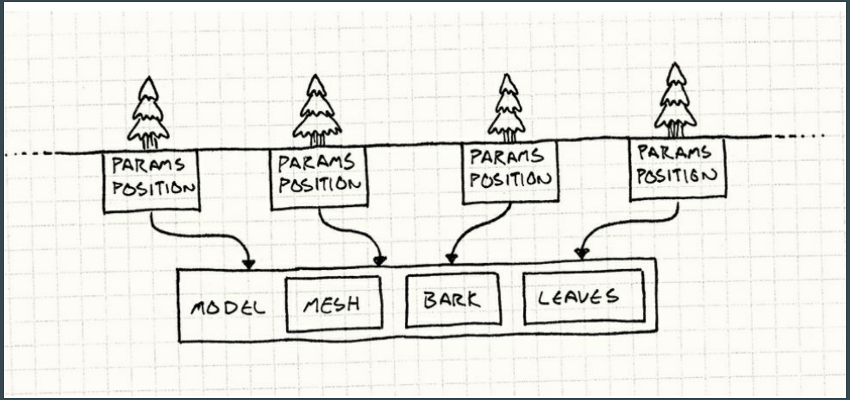
Command Way:

```
MoveCommand()->Execute()
```

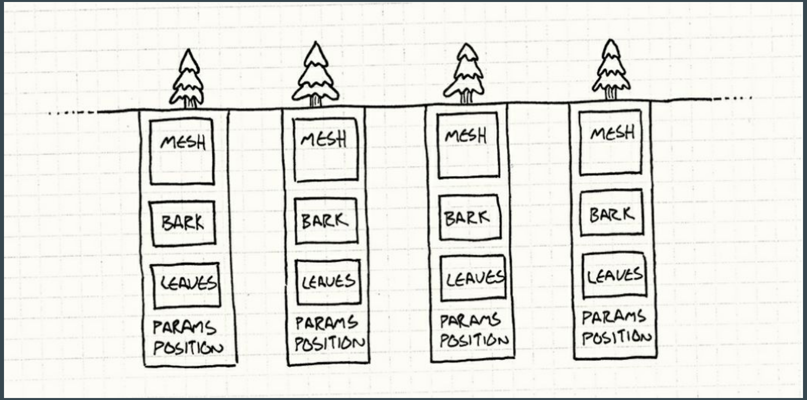
```
void Execute(){ this->MoveTo(Vector3D vectorToMoveTo); }
```

Flyweight

Lightweight Class with shared attributes



vs.



Credit: Game Programming Patterns: https://amzn.to/2X0oUkD

Observer

“Subscribe” to event types to be notified and take action. Can unsubscribe when it no longer cares about that event or it no longer exists.

Setup an object to observe a particular event and get notified of that event without a coupling to the originating object.

Ex: Player defeats another player:

- Each player will be notified
- Achievement system will be notified
- Server will be notified
- GameMode will be notified
- etc...

Prototype

Use an object you've already setup to duplicate the objects attributes, functionality or behavior.

Ex:

Goblin *ProtoGoblin = Goblin({health = 89, defense = 44, int = 33, weapons = [sword, dagger, goblin grenade],etc);

SpawnActor(ProtoGoblin); // Instead of remembering all those values

State

The being, behavior or situation of an object.

Ex: Attacking, Defending, Injured, Scared, Patrolling, etc

States can be implemented using Enums:

- Behavior {Attacking, Defending, Patrolling}
- Being{Injured, Scared, Angry, Jealous} // But what level

Or State Machines:

Player->Execute(); => void Execute(){ Attack(); } || void Execute(){ Defend(); }

Game Loop

```
while (true){
    HandleInput();
    ProcessPhyics();
    UpdateActors();
    Render();
}
```

Double Buffer

Allows the writing to an object without affecting the current object being used, displayed or read. While an object is being read (by the graphics or another system), changing that object as it’s read is dangerous.

Instead, write to a second object and swap out the current object when it’s done being read.

Ex: Frame Buffers. Draw to a background context while the current context is being displayed on the screen. Writing to the screen while it’s being shown could have strange effects like artifacting, flickering or ghost images.

Update Method

Event Tick!

Each object is responsible for its own behavior each frame.

Delta seconds represents the small amount of time between frames and helps make movement and other frames CPU or frame rate independent.

Ex:

30 fps = 0.033 seconds per frame

60 fps = 0.0167 seconds per frame

X += Delta seconds * MovementSpeed; instead of X += MovementSpeed;

Event Queue

Sometimes events can’t be processed all at the same time, or we’re not currently able to act upon that event. Event Queues allow the storing of events to be handled in the future or by priority (Priority Queue).

Object Pool

Instead of instantiating objects whenever they’re needed, then discarded, we can store them in an Object Pool. Instantiation and memory allocating/deallocation is slow!

Ex: Rows in a ListView, RecyclerView, TableView.

Level sections in an infinite runner game (right turn, left turn, straight). When the section goes behind the player and off the screen, return that level section to the object pool. When we need another right turn, grab it from the Object Pool. If you need two rights and only one exists, then create it and put it back in the Pool.

Dirty Flag

Whenever something needs to be updated or processed, set a Boolean value on a “Dirty Flag” and only act upon that process when the flag is dirty.

Ex: Objects in your game have moved on the screen: set the screen to dirty.

Inventory has changed: Set InventoryHasChanged to true so when you open your inventory, you can process the contents and update the display.