Game Programming Pattern

chap.1 Introduction

- Introduction
  - Abstraction and decoupling make evolving your program faster and easier, but don't waste time doing them unless you're confident the code in question needs that flexibility.
  - Think about and design for performance throughout your development cycle, but put off the low-level, nitty-gritty optimizations that lock assumptions into your code until as late as possible.
  - Move quickly to explore your game's design space, but don't go so fast that you leave a mess behind you. You'll have to live with it, after all.
  - If you are going to ditch code, don't waste time making it pretty. Rock stars trash hotel rooms because they know they're going to check out the next day.

Chap.2 Design Patterns Revisited

- Command
  - Definition
    - Callback in OO
  - Objective
    - Make things flexible & reusable
  - Consist of
    - Actor
    - Action (function)
  - Concept
    - Center Control

      e.g.

      1. Input Handler
      2. Command → Actor → Callback
    - Decouple actions and specific game object
    - Undo supported
      - Record previous state

      * Only store the necessary data

- Fly weight
  - Definition
    - Class and inheritance (OO method)
  - Objective
    - Save computing power
  - Concept
    - Seperate the stored parameters between shared value and unique value
  - Hardware's API support

- Direct 3D
- OpenGL

- **Observer**
  - Concept
    - Event Linstener
    - Send Notification
    - Send Message
    - Observe State of functions
- **Prototype**
  - Concept
    - Use one to handle different objects
    - Clean & Universal Function
- **Singleton**
  - Definition
    - One unique global instance

    e.g

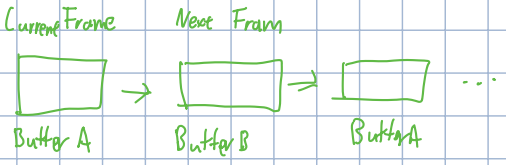
    Game Manager gm.
  - Objective
    - For calling its methods.
  - Concept
    - Avoid using if

- **State**
  - Concept
    - FSM
      - Enum for state instead of Bool flag

Chap. 3 Sequencing Patterns

- **Double Buffer**
  - Concept
    - Two buffer renders two frames

    e.g



- **Game Loop**
  - Concept
    - Keep the while loop wait for input and do something

- **Update Method**
  - Concept
    - Update() in Main Loop
  - Classes
    - Entity
    - Component Class

- Delegate Class

Chap. 4 Behavioral Pattern

- Bytecode
  - Concept
    - Build a bytecode Vitural Machine to do the low level program.
  - Objective
    - Make data to be more flexible, controlable. Imprae efficient (LuaJit is better but iOS ban it so use bytecode)

- Subclass Sandbox
  - Concept
    - Build a codebase as a libary that provides every method the gameplay needs.

    e.g.

    Particle system

    Unity Enigine Libary

- Type Object
  - Concept
    - Class in OO

Chap. 5 Decoupling Patterns

- Component
  - Concept
    - Seperate code and function into component

    e.g.

    Unity Rigidbody; Collider, Audio, UI, GetComponent<>()

- Event Queue
  - Concept
    - Build a queue for handling message/event sending and receiving

    e.g.

    Sender → Queue → Receive

    Update() Queue PlaySound()

- Service Locator
  - Concept
    - Decouple function with Service Locator

    e.g

    play sound → Service find Audio Component → Play sound
    - Often used in Networking & Easy deploy cross different platform

Chap. 6 Optimization Pattern

- Data Locality
  - Concept
    - Keep thinking where the data store to optimize game, use cache properly and avoid cache miss
- Dirty Flag
  - Concept
    - Avoid unnecessary work in the program
  - Pratical Usage

- Cache the GameObject at Game Start
- Move it or do something with it when it's needed.
- Avoid move object every frame.
- Use a dirty flag (Boolean) track it the Game Object updated. Modify it when it's unsyn.

Object Pool

Concept

- Use a Object Pool to store object in order to save memory and recusable
- Every object has a inuse flag

Spatial Partition

Concept

- Divide the game space into gird. Ensure every givd only contain small amount of objects. If it's not, subdivide it.