Выбор архитектурных паттернов

Для построения надёжной, расширяемой и модульной архитектуры 2D-движка были выбраны следующие архитектурные паттерны. Каждый из них решает конкретную задачу, связанную с управлением зависимостями, расширяемостью, обработкой событий или организацией поведения.

1. State (Состояние)

Область применения: управление игровыми сценами (меню, игра, пауза и т.д.)

Реализация:

Абстрактный класс Scene с методами update(), draw(), handle_event().

Конкретные сцены реализуют поведение для разных состояний игры.

Менеджер сцен (SceneManager) переключает активное состояние (сцену).

2. Observer (Наблюдатель)

Область применения: обработка пользовательских событий, взаимодействие между сущностями и UI.

Реализация:

Система событий (EventManager) реализует подписку и рассылку.

Объекты могут подписываться на конкретные типы событий (on('BUTTON_CLICKED')).

Позволяет отделить генератор события от обработчика.

3. Factory Method (Фабричный метод)

Область применения: создание игровых объектов, UI-элементов и сцен.

Реализация:

Создание сущностей и интерфейсных элементов через фабричные функции.

Возможность передавать конфигурации и параметры, не привязываясь к конкретным классам.

Упрощает загрузку объектов из JSON/данных.

4. Component (Компонент)

Область применения: создание расширяемых и переиспользуемых игровых объектов.

Реализация:

Каждый Entity может содержать набор компонентов (Transform, SpriteRenderer, Collider, и т.п.).

Компоненты можно добавлять и удалять в рантайме.

Поведение объектов настраивается не наследованием, а композицией.

5. Singleton (Одиночка)

Область применения: глобальные менеджеры (ресурсы, сцены, ввод, события).

Реализация:

AssetManager, InputManager, EventManager и SceneManager реализуются как одиночки.

Обеспечивается единый доступ к сервисам без передачи в каждый объект.

Важно: Singleton'ы реализуются *контролируемо* (например, через get_instance()), чтобы не нарушать тестируемость.

6. Strategy (Стратегия)

Область применения: поведение игровых объектов, логика AI, правила коллизий.

Реализация:

Объекты могут использовать разные стратегии движения, атаки и обработки событий.

Логика переключается без изменения класса объекта.

7. Template Method (Шаблонный метод)

Область применения: базовые абстракции, где нужно зафиксировать структуру алгоритма.

Реализация:

Абстрактный метод Scene.update() определяет общий алгоритм, а детали реализуются в подклассах (GameScene, MenuScene).

Упрощает создание новых элементов, не нарушая общий контракт поведения.

8. Service Locator (опционально)

Область применения: доступ к глобальным сервисам без жёсткой зависимости от конкретных реализаций.

Реализация (опциональная):

Централизованная регистрация и извлечение сервисов (Locator.get('audio')).

Удобно при тестировании или замене сервисов, но применяется ограниченно — чтобы не создавать скрытых зависимостей.