УДК 004.021

СРАВНЕНИЕ АРХИТЕКТУРНЫХ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ ИГР

А.Д. КАРПОВИЧ
(Представлено: Д.В. ПЯТКИН)

Производится сравнение архитектурных паттернов проектирования, чаще всего использующихся при программировании игр. Так же в статье описываются преимущества компонентно-ориентированного программирования в сравнении со стандартным объектно-ориентированным подходом.

Большие игры требуют создание сложной архитектуры. В результате чего, создаются сложные сущности и сложное взаимодействие между классами. *Объектно-ориентированный подход (ООП)* [1] программирования помогает в решение многих задач. Однако при разработке более сложных систем, в том числе систем, моделирующих процессы окружающего мира, появляется ряд других проблем, для решения которых необходим другой подход. Разработка игр, с использование ООП, влечет за собой постоянный рефакторинг кода и увеличение длительности разработки. Грамотно выбранная архитектура позволяет сильно упростить и ускорить процесс разработки игры.

Архитектура компьютерной игры [2] – система организации программы, которая определяет внутреннюю логику построения кода, выбор структурных элементов и определение связей между ними.

Сложная система игрового приложения состоит из нескольких подсистем – функциональных модулей, сервисов, слоёв, подпрограмм, соединённых в определённой последовательности. При таком функциональном разбиении разработчики получают не сильно связный код, а набор понятных элементов, взаимодействующих по простым правилам.

Важно в процессе разработке следовать признакам хорошей архитектуры:

– В него легко вносить правки. Новые фрагменты не требуют переписывания уже существующих.

– Система эффективна. Код решает поставленные задачи и работает в любых условиях.

– Срок разработки можно уменьшить, увеличив команду. Задачи легко делятся между разработчиками.

– Фрагменты кода можно повторно использовать в других системах.

Одним из самых простых архитектурных шаблонов использующимся в играх, а также одним из самых старых, считается *Game Loop* [3]. Этот шаблон применяется практически в каждой игре. Однако стоит заметить, что вне игровой индустрии этот шаблон не нашёл широкого применения

Game Loop (Игровой цикл) - это общий контроль потока для всей игровой программы. Каждая итерация игрового цикла известна как кадр. Большинство игр в режиме реального времени обновляются несколько раз в секунду: 30 и 60 являются двумя наиболее распространенными интервалами. Если игра работает со скоростью 60 кадров в секунду, то игровой цикл завершает 60 итераций каждую секунду.

```
while game is running
    process inputs
    update game world
    generate outputs
loop
```

Листинг 1. – Псевдокод игрового цикла

Game Loop работает на протяжении всей игры. На каждой итерации игровой цикл обрабатывает пользовательский ввод без блокировки, обновляет состояние игры и осуществляет рендеринг игры.

Ещё одним часто встречающимся архитектурным решением в программировании игр является Model-View-Controller(MVC) [4]. Он часто используется в приложениях с графическим интерфейсом пользователя (GUI) [5]. MVC разделяет программу на Контроллеры (Controllers), Представления (Views) и Модели (Models):

– Модель предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

– Представление отвечает за отображение данных модели пользователю, реагируя на изменения модели.

– Контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений.

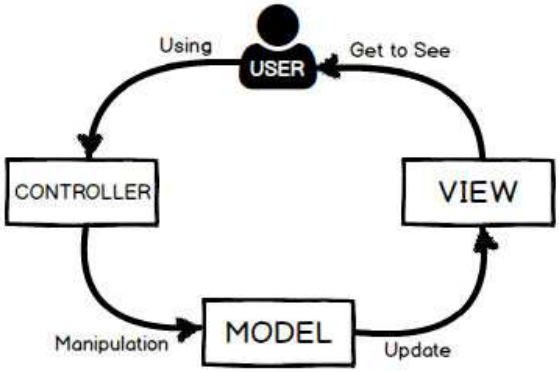

Рисунок 1. – Диаграмма MVC

Основная цель применения этой концепции состоит в отделении бизнес-логики (модели) от её визуализации (представления, вида). За счёт такого разделения повышается возможность повторного использования кода. Так же это разделение позволяет программистам, разрабатывающим бизнес-логику, работать независимо от разработчиков представления.

Однако в игровых приложениях возможности MVC обычно не используются в полной мере. Роль Контроллера в управлении потоком данных может взять на себя игровой цикл и набор Систем, а количество разнородных Компонентов гораздо больше двух и не описывается простым делением на Модель и Представление. Кроме того, многие компоненты являются необязательными.

В игровых приложениях лучше всего отдать предпочтение использованию архитектурных решений, реализующих *компонентно-ориентированный подход (КОП)* [6].

Компонентно-ориентированный подход – парадигма программирования, существенным образом опирающаяся на понятие *компонента* [7] – независимого модуля исходного кода программы, предназначенного для повторного использования и развёртывания и реализующегося в виде множества языковых конструкций (например, «классов» в объектно-ориентированных языках программирования), объединённых по общему признаку и организованных в соответствии с определёнными правилами и ограничениями.

Этот подход решает одну из главных проблем объектно-ориентированного подхода – проблему хрупких базовых классов – ситуации, когда изменить реализацию типа-предка невозможно, не нарушив корректность функционирования типов-потомков.

В ООП подходе объект определяется описываемым его классом. В КОП подходе, объект определяется компонентами, из которых он состоит. Такой подход упрощает повторное использование написанного кода – использование одного компонента в разных объектах. Также из различных комбинаций уже существующих компонентов, можно собрать новый тип объекта.

Для примера, возьмем объект «персонаж». С точки зрения ООП – это был бы один большой класс. С точки зрения КОП – это набор компонентов из которых состоит объект «персонаж»:

– характеристики персонажа – компонент «Stats»;
– управление персонажем – компонент «CharacterController»;
– анимация персонажа – «CharacterAnimationController»;
– обработчик столкновений – «CharacterCollisionHandler».

Шаблон проектирования *Component-Entity-System* [8] стал эффективным дополнением КОП подхода и Game Loop паттерна. Этот шаблон проектирования разделяет различные проблемы и задачи между сущностями (Entities), компонентами (Components) и системами (Systems):

– Entity(Сущность) - это контейнерные объекты, к которым могут быть присоединены компоненты. Entity являются основой для всех объектов на сцене. Такие игровые объекты, как юниты, декорации или пули, являются сущностями. Сущности ничем не отличаются друг от друга, и являются всего лишь вспомогательным контейнером для компонентов. Без компонентов Сущности не выполняют никаких функций и ничего не делают.

– Component(Компонент) – повторно используемый модуль или контейнер данных, который может быть присоединён к объектам для обеспечения внешнего вида, поведения или функциональности. Вся логика реализуется через компоненты. Различные типы объектов определяются путем смешивания, сопоставления и конфигурирования компонентов. Например: компонент ИИ (Искусственного интеллекта), компонент физического движения, графический компонент;

– System(Система) – обеспечивает управление классами компонентов. Системы часто являются необязательными, но их можно использовать для разделения логики и данных. Системы не владеют ни сущностями, ни компонентами. Они имеют доступ к ним через независимые объекты-диспетчеры, которые в свою очередь управляют жизненным циклом сущностей и компонентов.

Сущности компоненты и системы обмениваются информацией через Entity-Component-System Manager который является диспетчером событий.

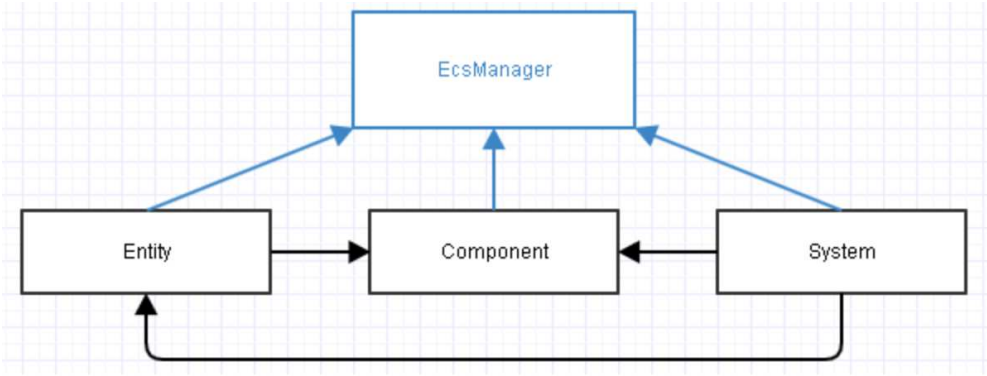

Рисунок 2. – Диаграмма Entity-Component-System

К примеру: мы имеем 3 компонента: Transform, задающий положение и масштабирование объекта на игровой сцене, Velocity, позволяющий объекту перемещаться и Material, который отображает объект на экране:

```
class TransformComponent : BaseComponent {
  Vector3 position;
  Vector3 rotation;
  Vector3 scale;
}

class VelocityComponent : BaseComponent {
 float x;
 float y;
 float speed;
}

class MaterialComponent : BaseComponent {
 Color color;
 Shader shader;
}
```

Листинг 2. – Пример компонентов

На основе этих компонентов с помощью композиции можно создать произвольное количество различных сущностей. Сущности по своей сути являются простыми контейнерами для компонентов:

```
class Player : BaseEntity {
 TransformComponent position;
 VelocityComponent velocity;
 MaterialComponent material;
}

class Floor : BaseEntity {
 TransformComponent position;
 MaterialComponent material;
}
```

Листинг 3. – Пример сущностей

Системы управляют логикой игры. Они принимают сущности и запускают операции над объектами, которые имеют конкретные компоненты, требуемые системой.

```
class SystemRender : BaseSystem {
  void update( Entity entities ) {
    for (auto entity : entities) {
      TransformComponent position = entity.getComponent<TransformComponent>();
      MaterialComponent material = entity.getComponent<MaterialComponent>();
      Render( position, material );
    }
  }
}
```

Листинг 4. – Пример Системы

В данной статье рассмотрены архитектурные паттерны проектирования игр, а также техники их использования. Подводя итоги, можно сделать вывод, что компонентно-ориентированный подход при разработке игр – наиболее инновационный. КОП делает процесс разработки более интуитивным и гибким.

ЛИТЕРАТУРА

1. Объектно-ориентированное программирование [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование. – Дата доступа: 16.09.18.
2. Архитектура, производительность и игры [Электронный ресурс] / Game Programming Patterns. – Режим доступа: https://martalex.gitbooks.io/gameprogrammingpatterns/content/chapter-1/1.1-architecture-performance-and-games.html. – Дата доступа: 16.09.18.
3. Игровой цикл [Электронный ресурс] / Game Programming Patterns. – Режим доступа: https://martalex.gitbooks.io/gameprogrammingpatterns/content/chapter-3/3.2-game-loop.html. – Дата доступа: 18.09.18.
4. Model-View-Controller [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Model-View-Controller. – Дата доступа: 18.09.18.
5. Графический интерфейс пользователя [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя. – Дата доступа: 19.09.18.
6. Компонентно-ориентированное программирование [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://ru.wikipedia.org/wiki/Компонентно-ориентированное_программирование. – Дата доступа: 19.09.18.
7. Компонент [Электронный ресурс] / Game Programming Patterns. – Режим доступа: https://martalex.gitbooks.io/gameprogrammingpatterns/content/chapter-5/5.1-component.html. – Дата доступа: 20.09.18.
8. Entity–component–system [Электронный ресурс] / Wikipedia – The Free Encyclopedia. – Режим доступа: https://en.wikipedia.org/wiki/Entity%E2%80%93component%E2%80%93system. – Дата доступа: 20.09.18.