ОСНОВНЫЕ ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ, ИСПОЛЬЗУЕМЫЕ ПРИ РАЗРАБОТКЕ ВИДЕОИГР

Охезин А.Д., Белова С.В. ассистент, Дударева О.В. к.ф.-м.н. ассистент
г. Бирск, ФГБОУ ВО Бирский филиал БашГУ

Парадигма программирования – это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

В разработке видеоигр, как и при разработке любой программы, могут использоваться различные парадигмы. В настоящее время самым распространенным является объектно-ориентированное программирование (ООП), однако этот метод имеет один недостаток – «хрупкий базовый класс». Этот недостаток заключается в том, что класс может иметь лишь один базовый класс. Поэтому необходимо тщательно разрабатывать архитектуру кода с использованием интерфейсов, делегатов и абстрактных классов.

В 1987 году Никлаус Вирт предложил паттерн написания блоков. Этот паттерн должен был решить недостаток ООП. Его суть заключается в том, что объект состоит из компонентов, которые компилируются отдельно друг от друга и подключаются динамически по мере необходимости. В дальнейшем эта парадигма получила название компонентно-ориентированное программирование (КОП)[1].

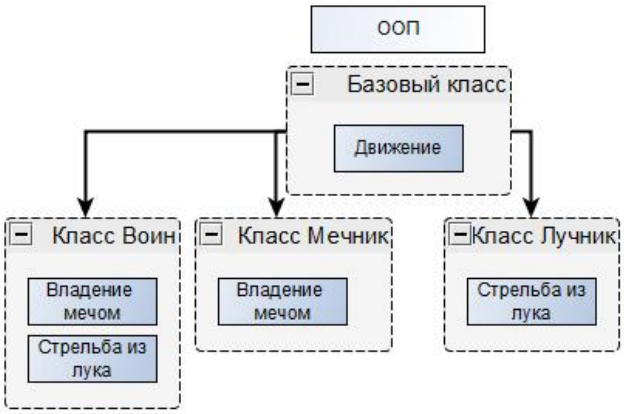

Рис. 1.1. ООП

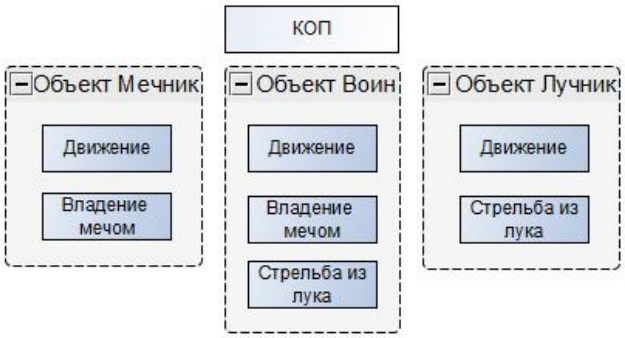

Рис. 1.2. КОП

На рис. 1.1 изображено наследование классов. Классы «Мечник» и «Лучник» реализуют решение задач, но класс «Воин» не может наследовать оба класса, хотя может наследовать один из них – это и есть недостаток ООП. В КОП компонент представляет собой скрипт с одноименным классом, реализующий решение определенной задачи. На рис. 1.2изображено существование трех компонентов: «движение», «владение мечом» и «стрельба из лука». Данные компоненты можно добавлять к различным объектам, что позволяет создавать объекты с различными комбинациями компонентов.

При разработке программ с формами используют событийно-ориентированное программирование (СОП) [2], однако она используется и в программировании видеоигр. СОП - парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь), сообщениями других программ. В программировании видеоигр совместно с КОП в компоненте создается событие и вызывается из этого же компонента. Методы в других компонентах, подписанные на это событие называются подписчиками, а вызвавший событие – отправитель.

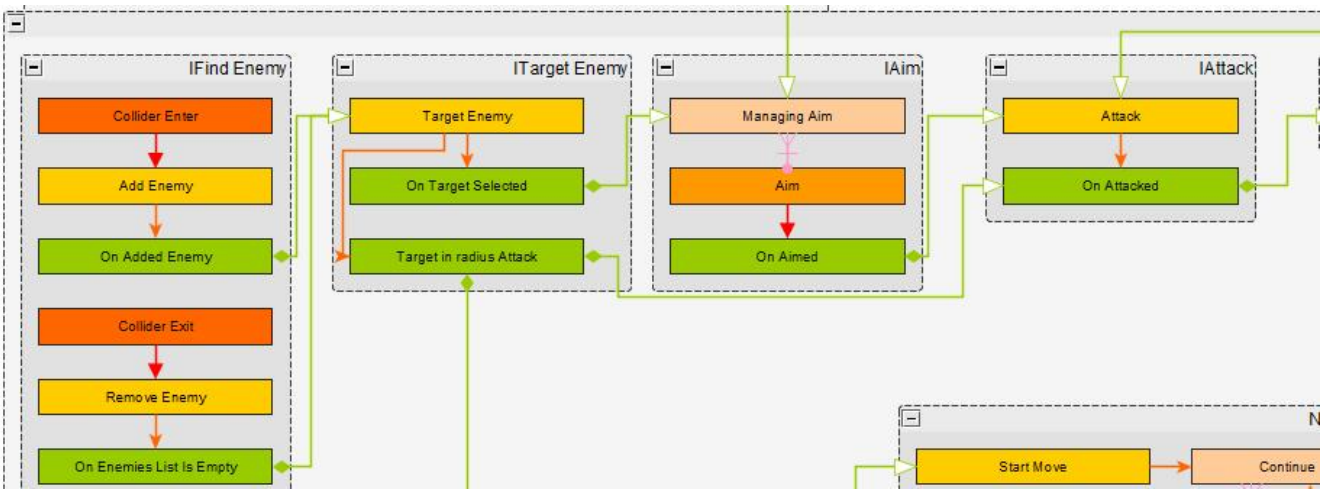

Рис. 2. Событийно-ориентированное программирование совместно с КОП

На рис. 2 зелеными блоками изображены события, а стрелки, идущие от них, указывают методы-подписчиков. Для вызова события используется метод-обработчик события, который проверяет есть ли у данного события подписчики и в случае истинности вызывает событие.

В языке программирования C# для реализации СОП событие создается с помощью ключевого слова event, а подписка и отписка осуществляется операторами += и -= соответственно.

В игровом движке Unityтакже имеются свои события. Для создания события нужно создать переменную типа UnityEvents. Для подписки используется метод AddListener(); для отписки – RemoveListener().

События Unityот событий C# отличаются в производительности. В работе [3] автор провел тесты на производительность и на количество мусора, создаваемое при использовании событий. Автор пришел к заключению, что UnityEventsмедленнее событий C#от 2 до 40 раз. Однако, если количество подписчиков события UnityEventsбольше двух, то оно создает меньше мусора, чем событие C#.

Событийно-ориентированное программирование в разработке видеоигр имеет недостатки: увеличивается количество строк кода, необходимо вычислять условия подписки и отписки от событий, сложность понимания взаимосвязей

между компонентами. Последний недостаток решается созданием схем взаимосвязей (рис.2), но это увеличивает время наподдержкуПО.

Вместо СОП используется автоматное программирование – парадигма программирования, суть которой заключается в том, что существуют состояния объекта и объект управления, изменяющий эти состояния при определенных условиях. Объект может находится только в одном состоянии и после выполнения условий объект перейдет в другое заранее определенное состояние. В игровой индустрии эта парадигма с использованием конечного автомата получила название «машина состояний».

Существует множество машин состояний для игрового движка Unity. В Unity5 появилась возможность создавать машины состояний используя инструмент Animator(рис. 3). Узел – это возможное состояние объекта, желтым цветом указывается активное состояние. Стрелка – возможный переход к следующему состоянию, в этом переходе указывается условие, при котором будет осуществлен переход. Взаимосвязь между машиной состояний и компонентами осуществляется скриптом, который наследуетStateMachineBehaviouri прикрепленный к узлу состояния. В переопределенных методах базового класса вызываются методы в компонентах объекта - появляется связь от машины состояния к компонентам, а компоненты изменяют значения переменных Animator'a, таким образом осуществляя взаимосвязь между компонентами и машиной состояния.

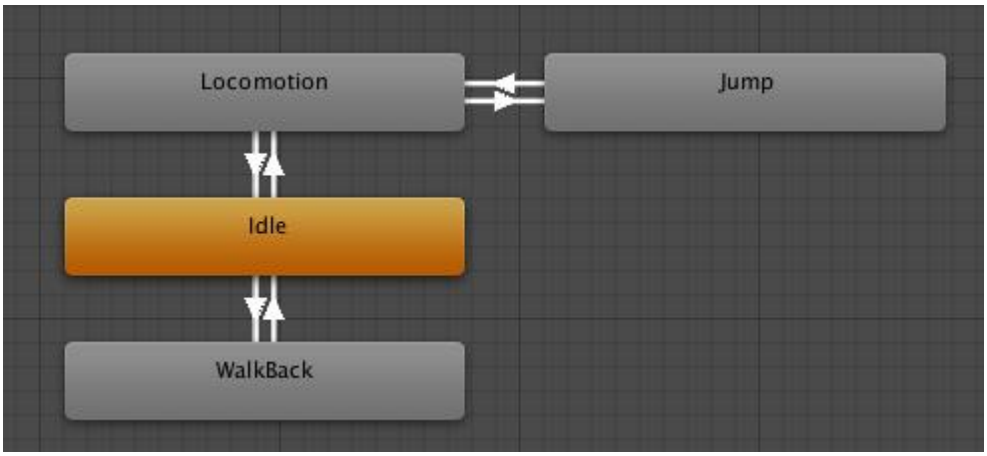

Рис. 3. Машина состояний в игровом движке Unity

В программировании игр используются различные парадигмы программирования в зависимости от поставленных целей и задач. В настоящее время очень часто используют компонентно-ориентированный подход совместно с машиной состояния, потому что они легко и быстро реализуются, а также удобны и упрощают программисту проектирование архитектуры приложения. Однако, в небольших проектах часто можно встретить объектно-ориентированное и событийно-ориентированное программирование.

Литература

1. Компонентно-ориентированное программирование[Электронный ресурс]URL: https://ru.wikipedia.org/wiki/Компонентно-ориентированное_програм-мирование (дата обращения: 20.04.2018)
2. Событийно-ориентированное программирование[Электронный ресурс]URL: https://ru.wikipedia.org/wiki/Событийно-ориентированное_програм-мирование (дата обращения: 20.04.2018)
3. EventPerformance: C# vs. UnityEvent [Электронныйресурс] URL: https://jacksondunstan.com/articles/3335 (датаобращения: 20.04.2018)