УДК 004.9

Студ. К.И. Боболя
Науч. рук.: доц. О.А. Новосельская; доц. Д.М. Романенко
(Кафедра информатики и веб-дизайна, БГТУ)

ПАТТЕРН МАШИНЫ СОСТОЯНИЙ

Паттерн State Machine в Unity обеспечивает легкое добавление новых состояний и переходов, а также гибкое управление поведением персонажа в зависимости от игровых событий и условий.

Следуя основам реализации паттерна State Machine, для облегчения дальнейшей работы создан интерфейс IState. Каждое состояние имеет вход, обработчик и выход. в данном случае Enter() – вхождение, Exit – выход , Update, FixedUpdate – методы обработчиков события. Так на смене состояний единожды проигрывается методы Exit предыдущего состония и Enter нового. в то время как активно состояние, следуя принципу работы скрипта Unity циклически проигрываются методы Update и PhysicsUpdate. Остальные указанные методы отвечают за обработку событий класса MonoBehavior реализованные через API Unity. Данными события можно назвать столкновения с коллайдерами, вхождениями в триггерные коллайдеры и т.д.

Таким образом машина состояний позволяет заранее «закешировать» все возможные вариаций состояний, просто подставляя в контексте нужный скрипт.

Для организации работы машины состояний, с возможностью её расширения, был реализован класс абстракции StateMachine. Класс PlayerMovementStateMachine отвечает за объявление и доступ к «кэшированным» состояниям. Каждое состояние вынесено в отельный классист. Большинство классов имеют материнский класс, объединяющий состояния с общей логикой. Так первичное разделение действий происходит на наземные – «Grounded» и воздушные – «Airborn». Наземные действия разделяются на группы перемещения, остановки и т.д. Данный класс на входе включает анимацию обычного бега и выключает на выходе. Он является наследником класса PlayerGroundedState, отвечающий за общую логику движения персонажа по поверхности и выход в состояние в воздухе PlayerAirborn State.

Использование паттерна State Machine в Unity обеспечивает четкую организацию кода, позволяет легко добавлять новые состояния и переходы, а также гибко реагировать на события и условия в игре. Благодаря этому, разработчики могут создавать сложные и интерактивные системы управления персонажем, при этом сохраняя простоту и структурированность кодовой базы.