МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

КОМПЬЮТЕРНАЯ ГРАФИКА

Методические рекомендации к лабораторным работам для студентов направлений подготовки 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия» дневной формы обучения

Могилев 2021

УДК 621.01
ББК 36.4
К 87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «12» июня 2021 г., протокол № 6

Составитель канд. техн. наук, доц. А. В. Шилов

Рецензент канд. техн. наук, доц. С. К. Крутолевич

Методические рекомендации к лабораторным работам предназначены для студентов направлений подготовки 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия» дневной формы обучения.

Учебно-методическое издание

КОМПЬЮТЕРНАЯ ГРАФИКА

Ответственный за выпуск	А. И. Якимов
Корректор	Т. А. Рыжикова
Компьютерная верстка	Н. П. Полевничая

Подписано в печать. Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью изучения курса «Компьютерная графика» является приобретение обучающимися навыков работы в Unreal Engine 4 (UE4). Данное программное обеспечение представляет собой движок и редактор, разработанный компанией Epic Games для создания игр и приложений от мультиплатформенных игр с миллионными бюджетами до мобильных приложений. Unreal Engine работает на операционных системах Windows и macOS, и разработанные в нем проекты могут быть развернуты на платформах Windows, Mac, PlayStation 4, Xbox One, iOS, Android, HTML5 и Linux.

В самой простой форме UE4 является коллекцией редакторов, используемых в различных стадиях производства игр или приложений. Редактор UE4 интегрирует множество сложных процессов разработки в удобную для использования среду разработчика. На изучение UE4, как и на изучение других инструментов, потребуется время.

Для выполнения практических заданий необходимо скачать движок UE4 с официального сайта: https://www.unrealengine.com/en-US/. Данное программное обеспечение является бесплатным с несколькими видами лицензий.

Лабораторные работы содержат рекомендованную последовательность действий для работы в среде UE4 при разработке компьютерной игры. В результате успешного выполнения всех заданий обучающийся получит навыки, необходимые для создания компьютерной игры.

1 Основные инструменты и ключевые элементы UE4

Цель работы: научиться создавать проект в Unreal Engine 4 (UE4), ориентироваться в среде разработки UE4, использовать панель Viewport, World Outliner, Details, Content Browser, создавать классы Blueprint, добавлять компоненты к классам, применять изменения в экземплярах класса на блюпринт-класс, сохранять уровни в UE4.

При нажатии на ярлык UE4 откроется меню с предложением создать проект определенного типа. Выберите **Games** и нажмите кнопку **Next** (рисунок 1.1).

Тип проекта определяет предустановленные настройки проекта.

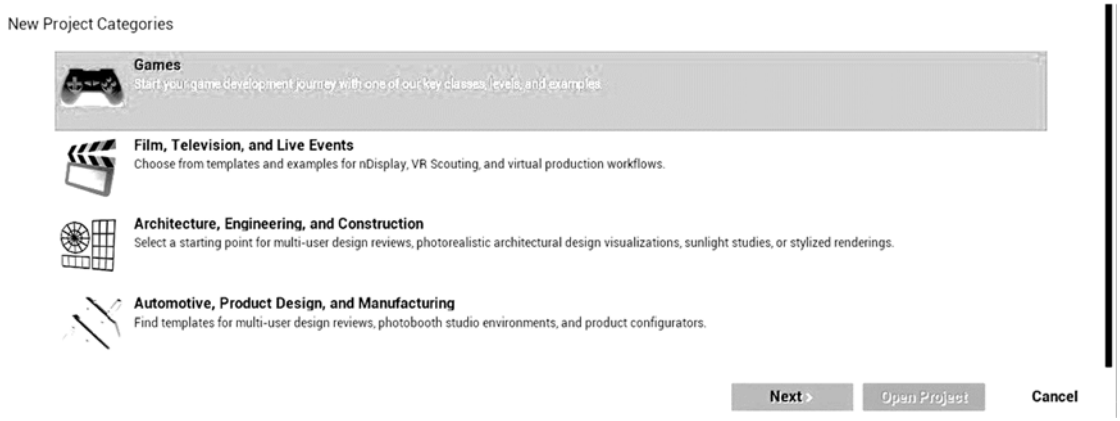

Рисунок 1.1 – Меню выбора типа проекта

Следующим шагом является выбор темплейта проекта. Темплейтом является предустановленный набор ассетов, который будет встроен в проект. Для игр темплейты разделены на игровые жанры. Темплейт **Blank** означает, что будет создан «пустой» проект без предустановленных ассетов. Выберите его и нажмите кнопку **Next** (рисунок 1.2).

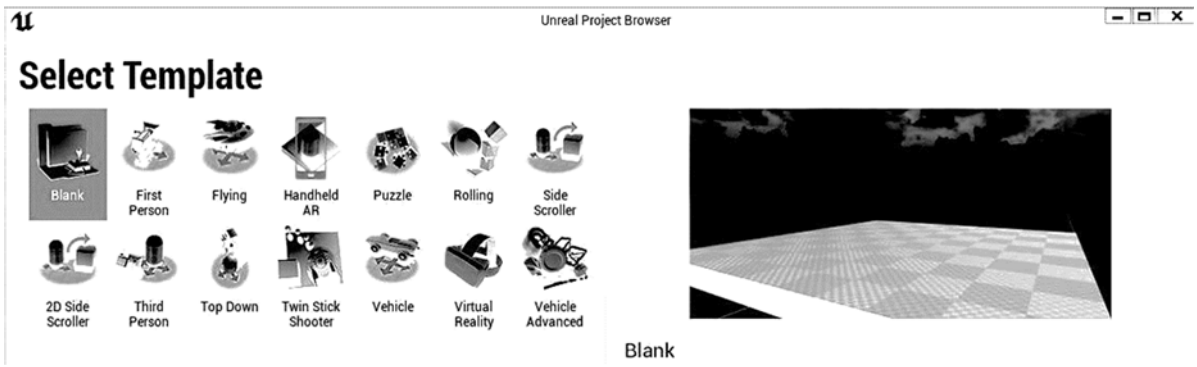

Рисунок 1.2 – Меню выбора шаблона проекта

Последним шагом является выбор настроек проекта. Самой важной настройкой является выбор блюпринт- или C++ проекта. Все настройки можно изменить в ходе разработке игры, но с данной опцией лучше определиться с самого

начала, т. к. трансформация блюпринт-проекта в C++ и обратно может вызвать ошибки в скриптах и блюпринт-классах (рисунок 1.3).

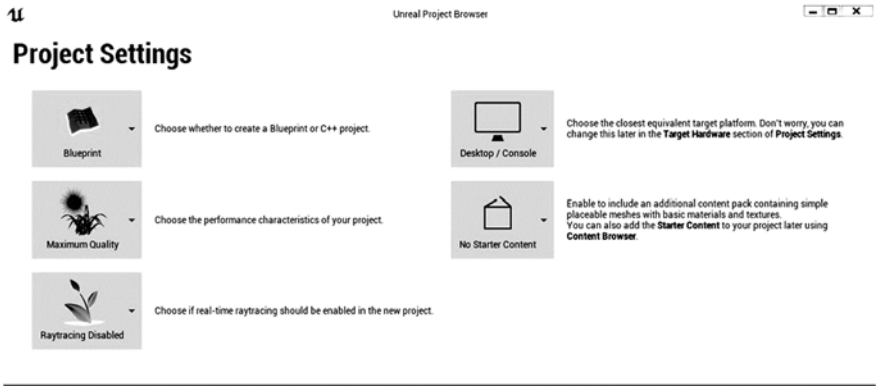

Рисунок 1.3 – Окно выбора настроек проекта

При открытии формы пустого (**Blank)** проекта в нем находятся основные панели и окна UE4. Под **окном** в среде UE4 понимается большая область, в которой располагаются интерактивные элементы среды UE4.

Окно **Viewport** – это окно, через которое можно наблюдать уровень. В нём находится виртуальное пространство, включая все объекты Вашего уровня, в том числе те, что скрыты в самой игре – например, вспомогательные иконки (billboard) объектов. Для переключения режима отображения между разработчиком и **Game view** (рисунок 1.4) нажмите на клавишу **G**.

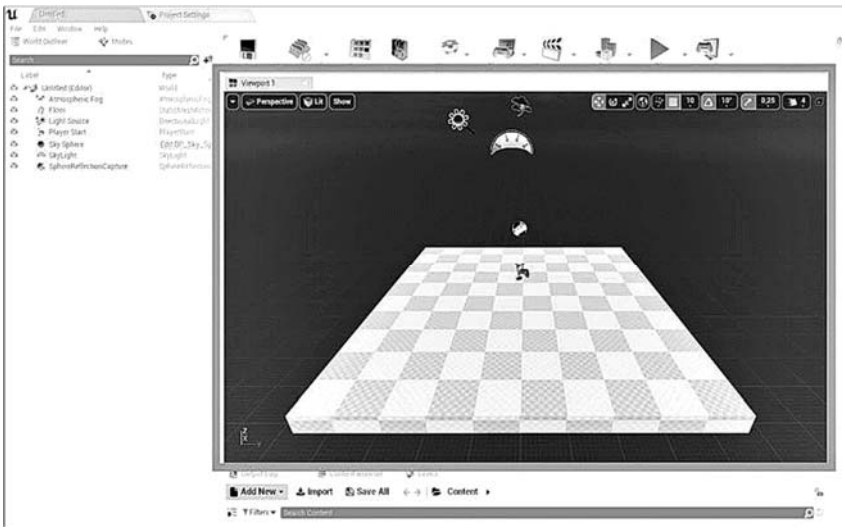

Рисунок 1.4 – Окно Viewport

Вы можете осмотреть пространство уровня, удерживая **ПКМ** и перемещая её. Для перемещения по уровню нужно зажать **ПКМ** и использовать клавиши **WASD**. Клавиши **Q** и **E** будут поднимать и опускать камеру соответственно.

Под панелью в среде UE4 будем понимать небольшую область экрана (зачастую вертикальную), в которой располагаются интерактивные элементы

среды UE4. В отличие от окон, в панели располагаются контекстные параметры того или иного виртуального объекта.

В панели **World Outliner** отображаются все объекты на текущем уровне. Вы можете упорядочить список, распределив связанные объекты по папкам, а также искать и фильтровать их по типам, можно объединять объекты в группы (рисунок 1.5). Обратите внимание, что можно использовать все классические сочетания клавиш, характерные для множества программ. Можно выделить все объекты разом, нажав сочетание клавиш Ctrl + A, дублировать объекты сочетанием клавиш Ctrl + D, копировать в буфер обмена – Ctrl + C, вставить из буфера обмена – Ctrl + V. Используйте клавишу Delete для удаления выбранных объектов.

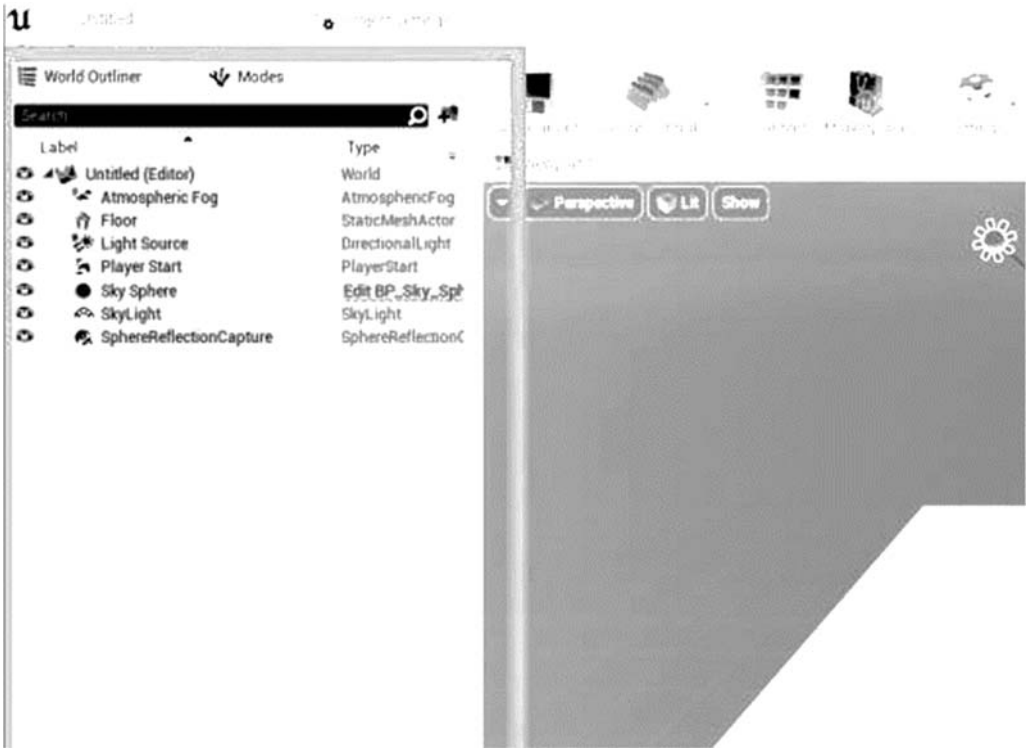

Рисунок 1.5 – Панель World Outliner отображает все объекты текущего уровня

В панели **Details** отображаются все свойства выбранного объекта, экземпляра блюпринт-класса. Эта панель используется для изменения параметров объекта. Внесённые изменения повлияют только на выбранный экземпляр объекта (рисунок 1.6). В панели Details также перечислены компоненты выбранного объекта. Обратите внимание, что при выборе отдельного компонента список свойств может изменяться.

Content Browser – в этой панели отображаются все файлы проекта. Её можно использовать для создания папок и упорядочивания файлов. Здесь также можно выполнять поиск по файлам с помощью поисковой строки или фильтров (рисунок 1.7).

Blueprints – это система визуального программирования в UE 4. Соединяя проводами (лапшой) **ноды, эвенты, функции, переменные,** вы можете создать

комплексный геймплей.

Рисунок 1.6 – Панель Details демонстрирует все свойства выбранного объекта

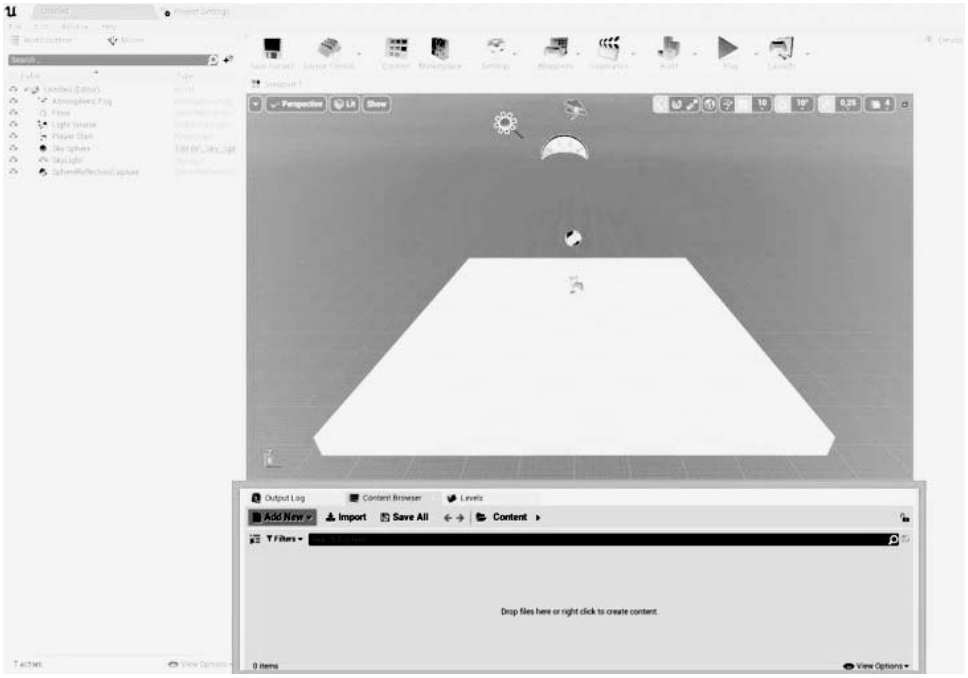

Рисунок 1.7 – Панель Content Browser, показывающая список файлов проекта в UE4

Также можно часто встретить использование термина Blueprint иди Blueprint class для обозначения класса, созданного в системе Blueprints. Классы Blueprints подчиняются правилам ООП и компонентным подходам к организации архитектуры приложения.

Панель **Modes:** в этой панели можно переключаться между различными инструментами, например, **Landscape Tool** и **Foliage Tool**. Инструментом по умолчанию является **Place Tool**. Он позволяет располагать на уровне различные типы объектов, такие как 3D-модели, источники освещения и камеры (рисунок 1.8). Добавьте объект **Empty Actor** на уровень. Просто перетащите его иконку с вкладки **Basic** панели **Modes** (рисунок 1.9) в окно **Viewport**. Вы только что поставили пустой объект, он будет выделен сферой-билбордом (не забывайте про клавишу **G**) – данная сфера видна только для разработчика. Иногда объекты, которые могут находиться на уровне, в Unreal Eninge 4 называют Actor или актёр. В методических рекомендациях будут использоваться слова «объект UE4» и «актёр».

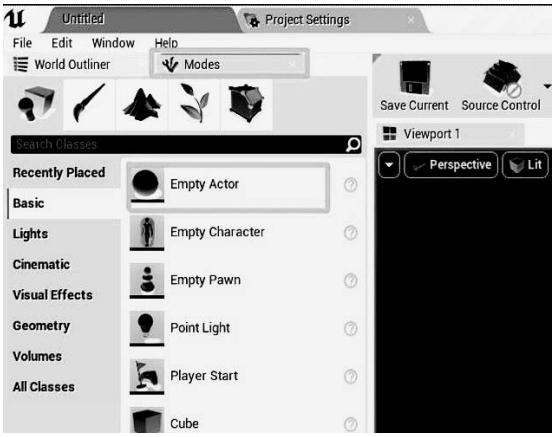

Рисунок 1.8 – Панель Modes, поставьте объект Empty Actor на уровень

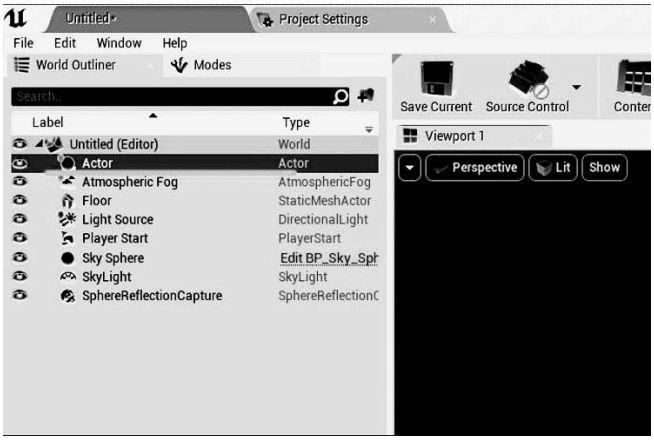

Рисунок 1.9 – Empty Actor на уровне

Актёры (Actor) – это главные объекты игры, с которыми можно взаимодействовать или которыми можно управлять. Это могут быть сам персонаж игрока, аптечки, патроны, оружие, другие персонажи, двери и любые интерактивные объекты.

Вы можете попробовать передвигать объект, вращать его и масштабировать. Используйте панель инструментов в правом верхнем углу **Viewport** для переключения между режимами (рисунок 1.10).

Рисунок 1.10 – Панель инструментов Transform

Перейдите в окно компонентов панели **Details,** выбранного **Empty Actor**, нажмите на кнопку **Add Component** и выберите **Point Light** (рисунок 1.11).

Чтобы превратить объект с компонентом point light в блюпринт-класс, нажмите на синюю кнопку **Blueprint/Add Script** (рисунок 1.12). Выберите папку, в которой будет сохранен блюпринт-класс (рисунок 1.13).

Нажмите ПКМ на папке Content и нажмите на появившийся пункт Create Folder. Обычно папку, в которой хранятся блюпринт-классы, принято называть Blueprints или BPs. Вы также можете дать название этого класса в поле ниже. В данном случае это Pulsar_BP. Нажимайте кнопку Create Blueprint.

Существует несколько способов создать **Blueprint Class,** например, через кнопку **Blueprints** в верхней панели инструментов. В выпадающем списке вы также можете создать пустой Blueprint класс и после дополнять его нужными компонентами (рисунок 1.14).

Вы можете перетаскивать блюпринт-класс из окна Content Browser на уровень, создавая экземпляры выбранного класса (рисунок 1.15).

При двойном нажатии на файл блюпринта вы перейдете в отдельное окно Blueprint **Editor** (рисунок 1.16).

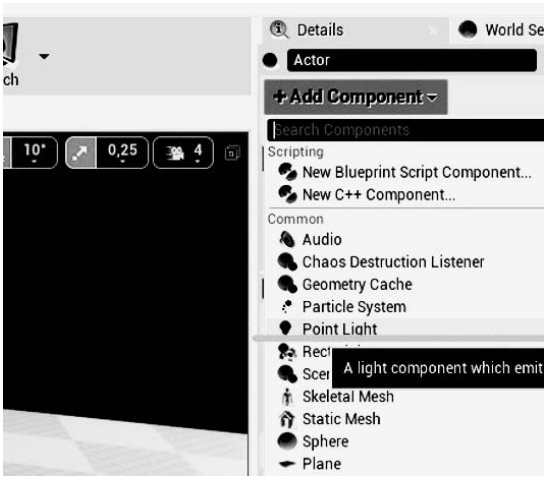

Рисунок 1.11 – Добавление компонента Point Light к объекту Empty Actor

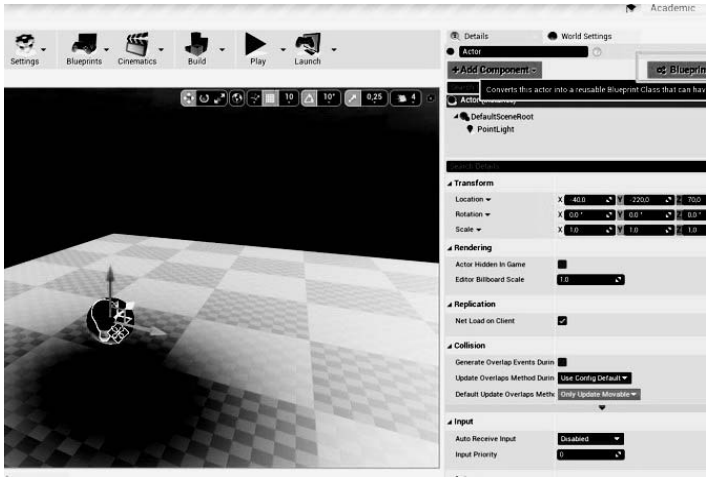

Рисунок 1.12 – Открытие окна сохранения объекта в виде блюпринт-класса

Рисунок 1.13 – Окно выбора пути сохранения блюпринт-класса

Рисунок 1.14 – Создание блюпринт-класса

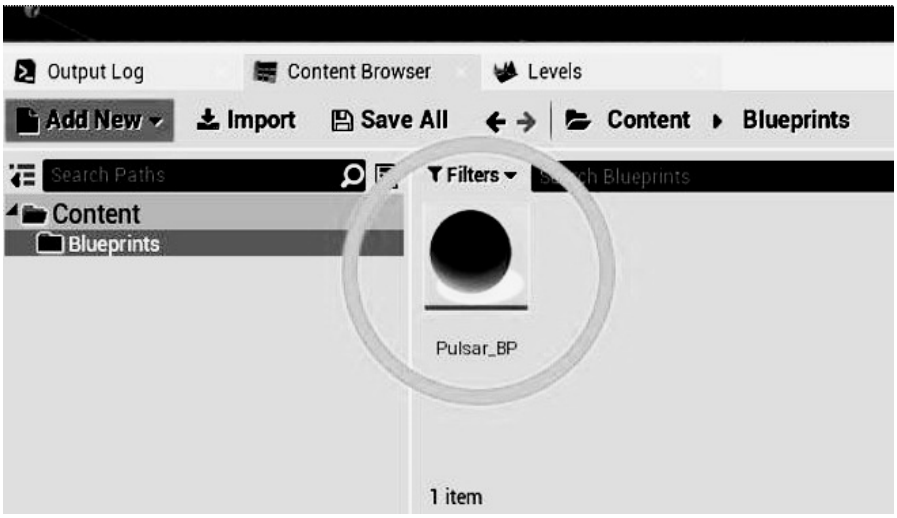

Рисунок 1.15 – Нажатие ПКМ на иконку блюпринт-класса

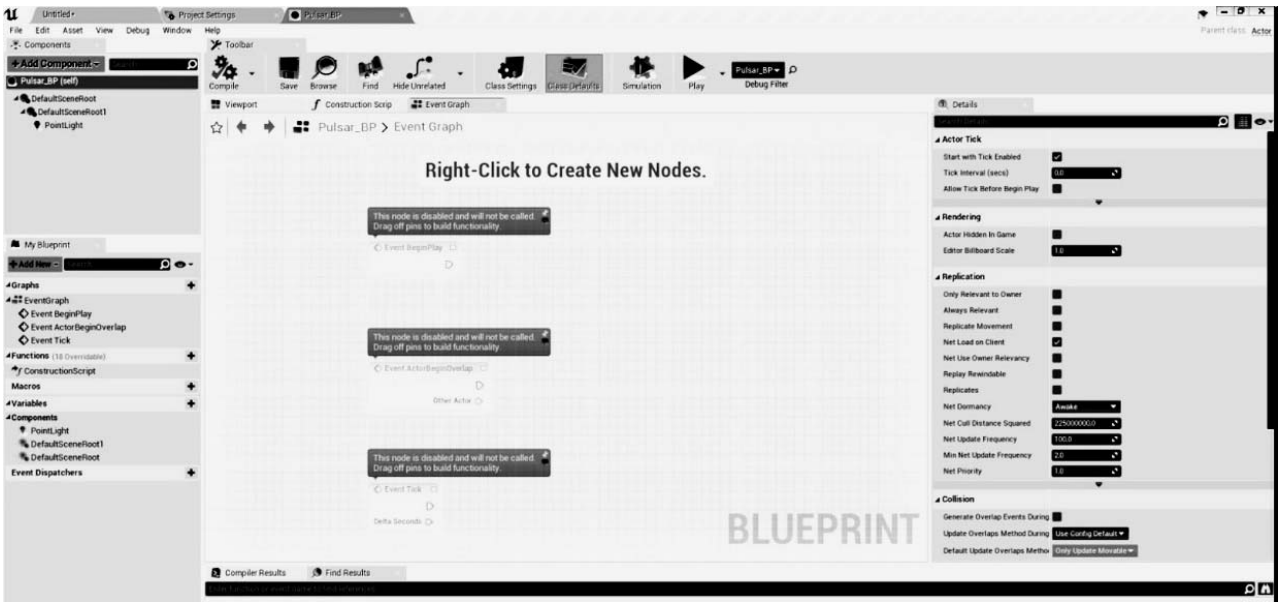

Рисунок 1.16 – Окно Blueprint Editor

Нажав на вкладку окна viewport, можно посмотреть, как выглядит блюпринт-класс (рисунок 1.17).

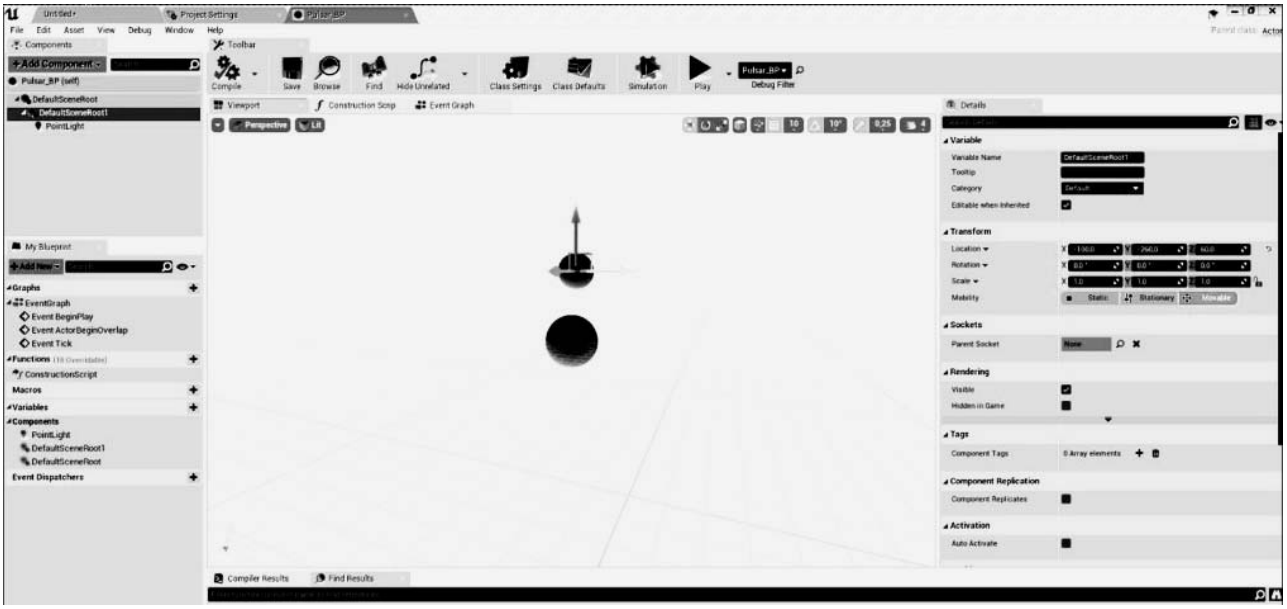

Рисунок 1.17 – Окно Viewport

Важно понимать, что данное окно Viewport является контекстным для выбранного класса и не отображает блюпринт-класс на всём уровне.

В случае с Unreal Engine версий 4.24 может наблюдаться следующее поведение. Присутствуют два компонента **Default Scene Root**. Вы можете удалить нижний по иерархии компонент Default Scene Root. Чтобы избежать поведения с двумя Default Scene Root компонентами, сначала создайте блюпринт-класс из «пустого» Actor, а затем в окне **Blueprint Editor** добавьте нужные Вам компоненты.

Нажмите кнопку **Compile**, чтобы применить изменения (рисунок 1.18).

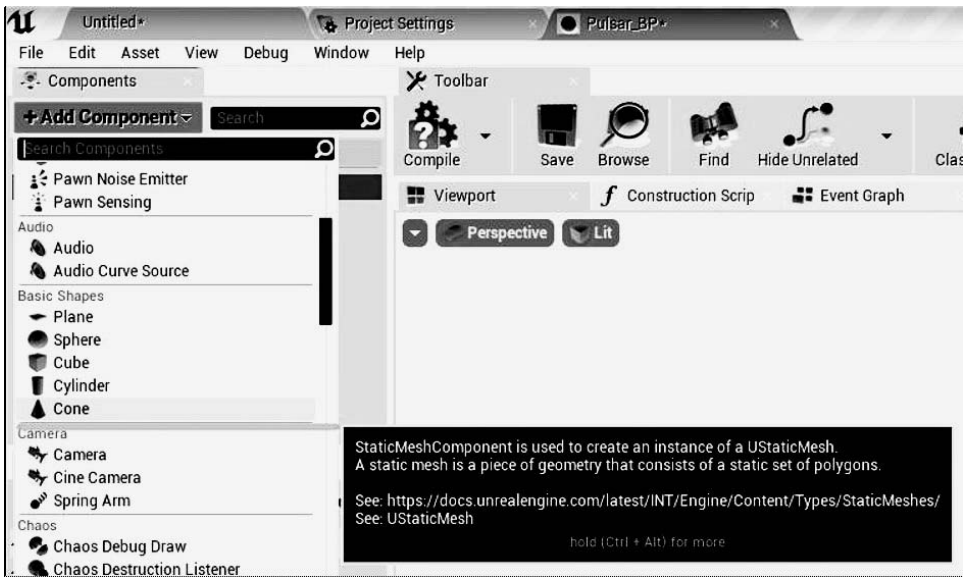

Рисунок 1.18 – Добавление компонента Cone из панели Components

Добавьте компонент Cone (примитив, конус). Поднимите компонент Point Light выше, чтобы было заметно влияние источника света на конус. Скомпилируйте блюпринт (рисунок 1.19). Есть вероятность, что у конуса будет назначен Default Material (выглядит, как шахматная сетка) – это нормально на данный момент.

Рисунок 1.19 – Приблизительное изображение окна Viewport блюпринт-класса Pulsar_BP

Обратите внимание, что на уровне все экземпляры блюпринтов теперь имеют конус. Таким образом, можно быстро вносить изменения во все экземпляры блюпринт-класса, расставленные на всех уровнях, редактируя сам класс. В данном примере изменены цвет и интенсивность компонента point light, а также пропорции конуса (компонент cone). Вы можете изменять пропорции конуса (и любого другого объекта), редактируя значения параметров Transform в панели Details.

Иногда необходимо посмотреть, как изменения на объекте (экземпляр блюпринта) вписываются в окружение уровня. Чтобы изменения объекта применились на блюпринт-класс, нажмите кнопку **Edit Blueprint** и выберите **Apply Instance Changes to Blueprint**. Если всё сделано верно, остальные экземпляры (другим словом, инстансы) класса автоматически обновятся с учётом новых изменений (рисунок 1.20).

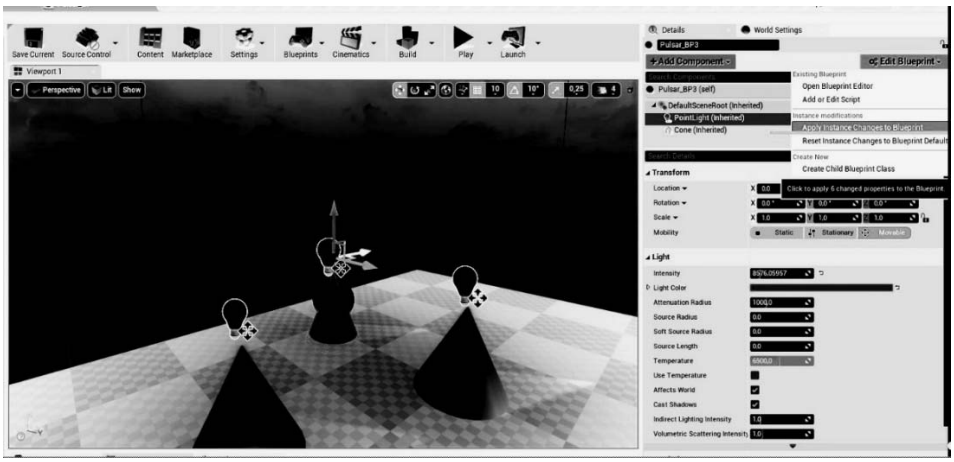

Рисунок 1.20 – Выбор пункта Apply Instance Changes to Blueprint

Сохраните уровень, нажав Ctrl + Shift + S или через меню File-> Save Current. В открывшемся окне создайте папку Maps, дайте название уровню и нажмите кнопку Save (рисунок 1.21) [1, 2].

Рисунок 1.21 – Использование только латинских букв

2 Программирование на Blueprints. Часть I

Цель работы: изучить архитектуру Blueprint проекта UE4 и порядок выполнения блюпринт-класса: ноды Begin Play и Tick Update; изучить Level; научиться обращаться к компонентам Blueprint и менять их свойства.

Игра – это **контент**, **связанный** между собой **логикой** и запрограммированный на определенные действия.

Для этого можно использовать **C++** и **Blueprints**. Blueprints – это система визуального программирования в UE 4. Соединяя проводами **ноды, события, функции, переменные,** вы можете создать комплексный геймплей.

Несмотря на кажущуюся наивность блюпринтов, они подчиняются ООП, компонентной архитектуре, паттернам программирования.

Откройте **Blueprint editor** блюпринт-класса, созданного на прошлом уроке (конус с синим источником света) (рисунок 2.1).

Перейдите в окно **Event Graph** (рисунок 2.2).

Перед Вами откроется окно, в котором и будет содержаться основная часть логики Вашего класса. Вы можете увидеть три серых блока: **Event BeginPlay, Event ActorBeginOverlap, Event Tick –** эти блоки и называют нодами (рисунок 2.3).

Рисунок 2.1 – Блюпринт-класс Pulsar_BP

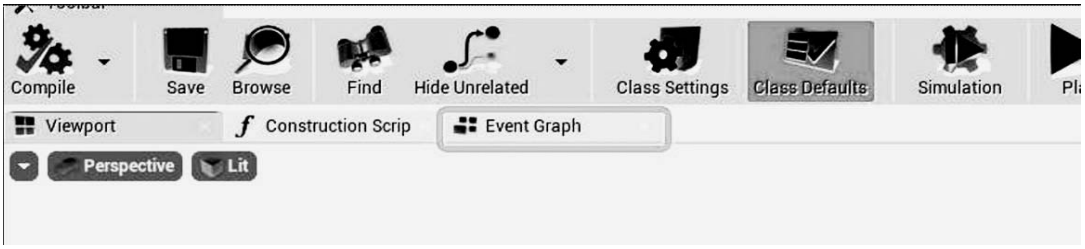

Рисунок 2.2 – Вкладка Event Graph, в окне которой создаётся логика класса

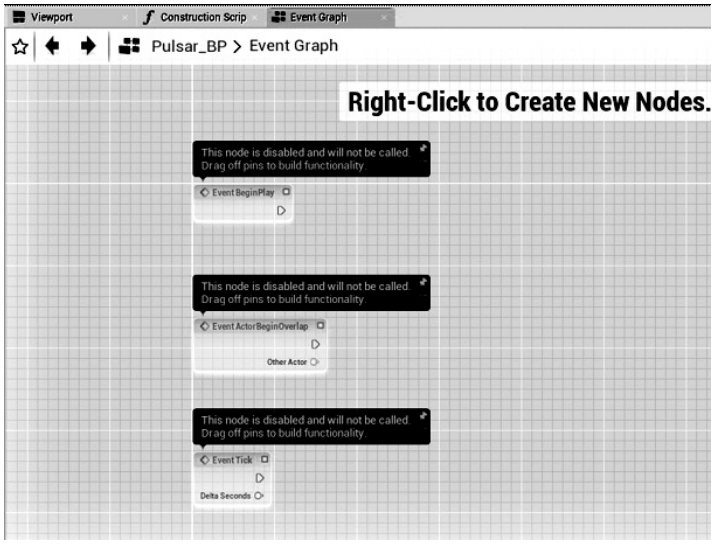

Рисунок 2.3 – Окно Event Graph при создании любого блюпринт-класса

Нажмите **ПКМ** в свободном месте, чтобы создать свою собственную ноду. Вы увидите множество рассортированных по своему функционалу нод. Выведите текст Hello в встроенную консоль. Для этого наберите в строке поиска **Print**

и выберите из предложенных ноду **Print String** (рисунок 2.4).

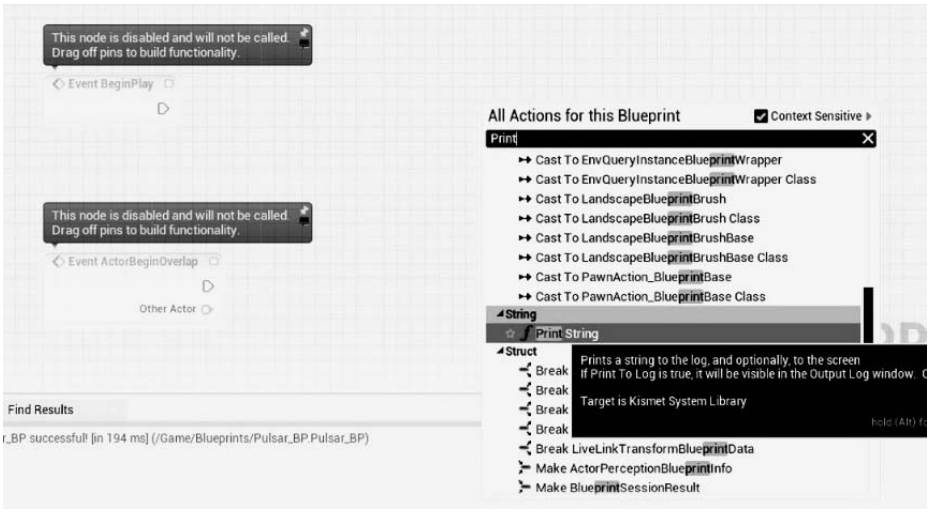

Рисунок 2.4 – Контекстное меню выбора ноды

Вы увидите, как появилась нода Print String. Теперь «вытяните» соединение из треугольного символа ноды **BeginPlay** и соедините с входящим пином ноды **Print String**. Треугольный символ – это пин выполнения **execution pin**. Выполнение функции, события и кода в целом происходит слева направо по соединению **execution** (белому проводу). Событие **BeginPlay**, как следует из его названия, выполняется **один раз при старте игры** (рисунки 2.5 и 2.6).

Рисунок 2.5 – Логическая конструкция, выводящая текст Hello

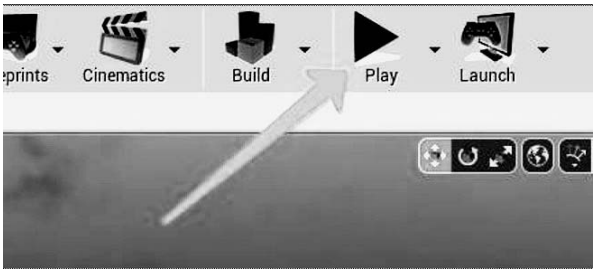

Рисунок 2.6 – Кнопка Play, запускающая выполнение всех экземпляров блюпринт-классов

Подключите ноду Print String к ноду Event Tick и запустите проект (рисунок 2.7).

Рисунок 2.7 – Конструкция, выводящая текст Hello

Множественный и непрекращающийся вывод значения аргумента **Print String** подсказывает, что нода **Event Tick** выполняется каждый кадр (frame).

Level Blueprint – это специализированный тип блюпринта, который действует как глобальный event graph для своего уровня. Каждый уровень в Вашем проекте имеет собственный Level Blueprint, созданный по умолчанию, но который можно редактировать, однако Вы не можете создать собственный класс Level Blueprint – в этом нет необходимости.

Чтобы открыть **Level Blueprint**, нажмите кнопку **Blueprints** и выберите из списка **Level Blueprint** (рисунок 2.8).

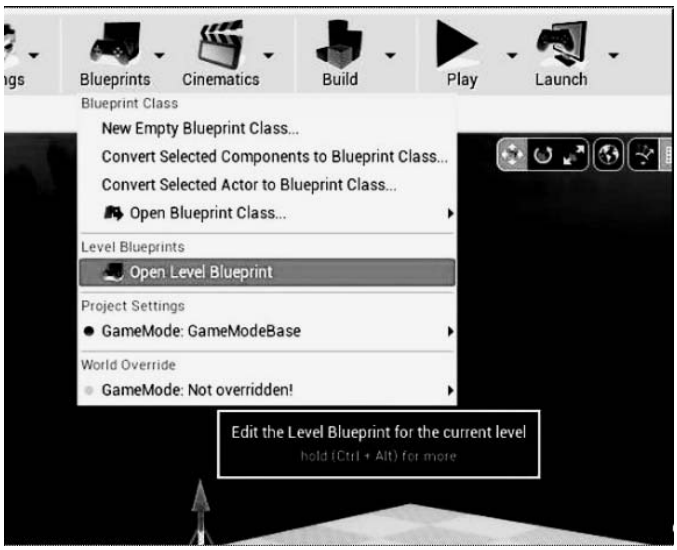

Рисунок 2.8 – Доступ к окну Level Blueprint

В **Level Blueprint** логично располагать функции и ивенты, которые являются специфичными для конкретного уровня. Level Blueprint может быстро получить доступ по ссылке к любому блюпринту или актёру на уровне:

- выберите блюпринт Pulsar_BP в World Outliner;
- перейдите в Event Graph Level Blueprint'a;
- зажмите клавишу R (reference) и нажмите ЛКМ.

В общей архитектуре кода Unreal Engine классы Game State и Game Mode играют важную роль, однако в основном для онлайн-игр. Для одно-пользовательских игр значимость этих классов снижается. Поскольку создание сетевых игр не рассматривается в данном курсе, подробную информацию по Game Mode и Game State Вы можете узнать в официальной документации.

Зайдите в Event Graph Pulsar_BP, измените параметры компонента Point Light через код и перетащите компонент **Point Light** в поле **Event Graph** (рисунок 2.9). Он будет выглядеть как нода с одним выходным пином.

Вытяните пин Point Light и наберите в поиске **Set Light Color** (рисунок 2.10). Поиск названий нужных параметров нарабатывается с опытом.

Обратите внимание, что, следуя логике, Вы можете посмотреть названия нужных Вам параметров, нажав на компонент. Вы можете попробовать вводить **Set Intensity**, **Set Attenuation Radius** и прочие в поиске свойств компонента в

Event Graph, чтобы изменять или получать их значения (рисунок 2.11).

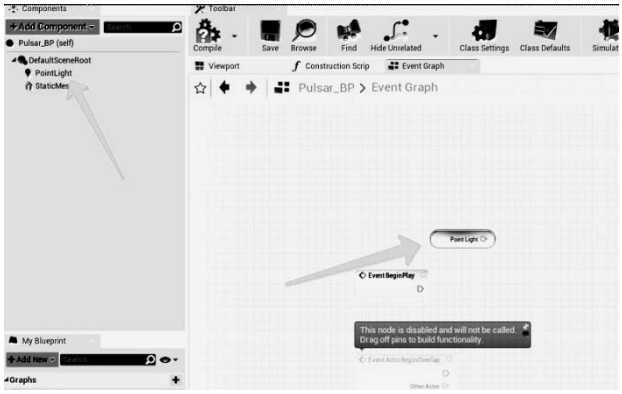

Рисунок 2.9 – Компонент из панели Components в окне Event Graph

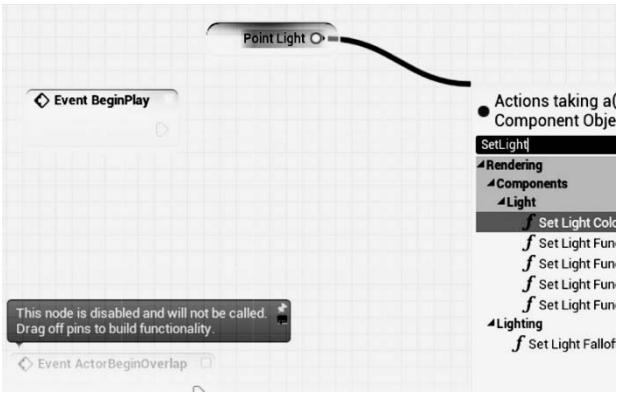

Рисунок 2.10 – Использование поисковой строки для нахождения параметров компонентов

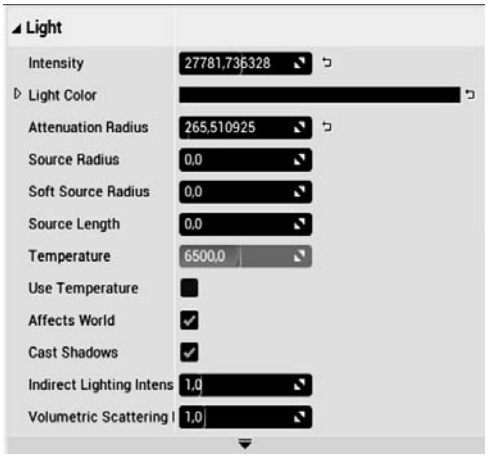

Рисунок 2.11 – Основные параметры (или свойства) компонента Point Light

Свяжите ноды Begin Play и SetLight Color, выберите желаемый цвет, нажав на маленький черный квадрат, репрезентирующий черный цвет (по умолчанию) (рисунки 2.12 и 2.13). Скомпилируйте и запустите проект.

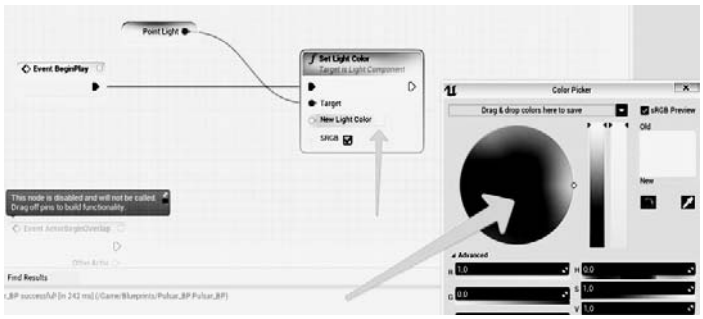

Рисунок 2.12 – Окно выбора цвета для New Light Color в палитре

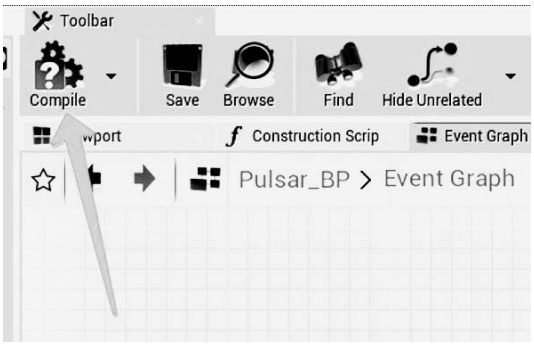

Рисунок 2.13 – Подтверждение выбора

Цвет источника света в Pulsar_BP должен поменяться на выбранный.

Обратите внимание, что Вы можете «разобрать» большинство пинов составных типов данных на простые типы данных. Нажмите на пин New Light Color ПКМ и выберите Split Pin. При желании Вы сможете указывать цвет в формате вектора RGBA. Это может пригодиться при генерации случайных цветов.

Теперь сделайте так, чтобы источник света плавно изменял свою интенсивность с 0 до 5000 со старта игры в течение 10 с. Для этого можно использовать ноду Timeline.

Добавьте ноду Timeline, нажав ПКМ и набрав в поиске Add Timeline. Вы можете дать собственной название ноде таймлайна. Обратите внимание, что у ноды Timeline имеется несколько входящих пинов для execution «проводов» (рисунок 2.14).

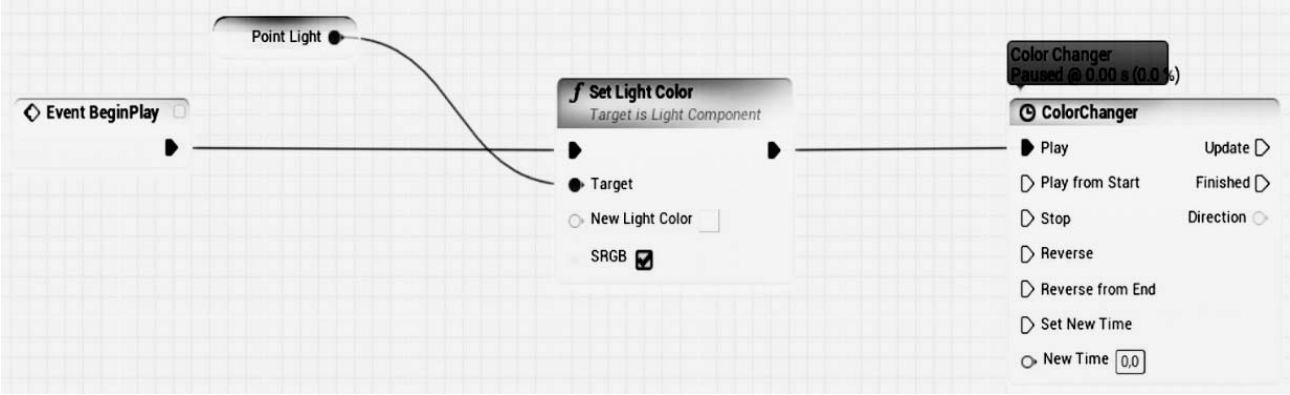

Рисунок 2.14 – Нод Timeline с именем ColorChanger

Щелкните 2 раза по ноде Timeline – у вас откроется окно редактирования кривых Curve Editor. Вы можете сделать кривые под несколько типов данных, под float, vector, color и даже event. Выберите значок f+, чтобы добавить кривую для float значений (рисунок 2.15).

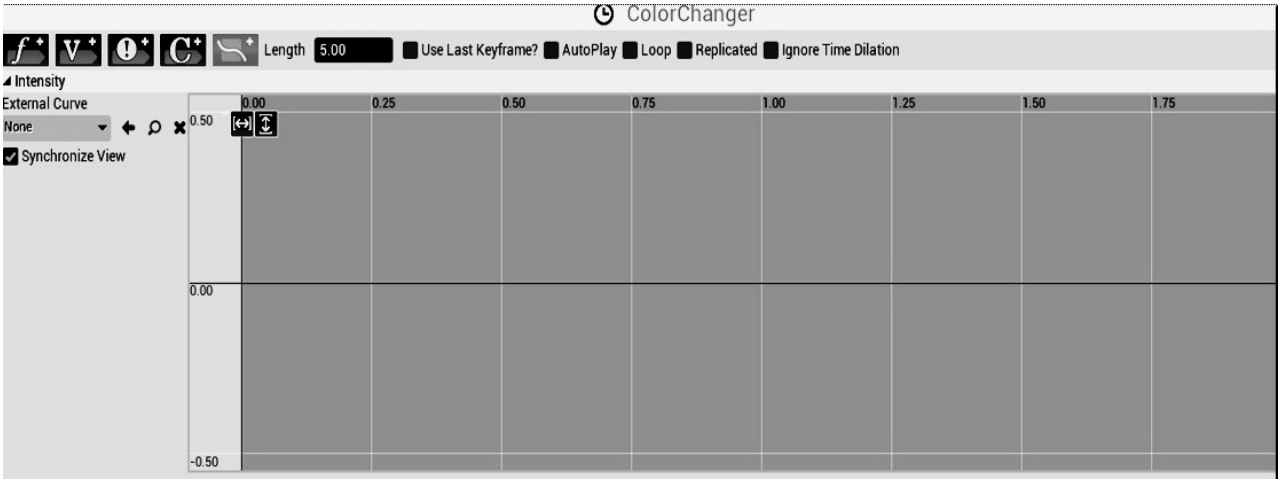

Рисунок 2.15 – Окно редактирования кривых

Нажмите ПКМ на любом участке кривой и выберите пункт **Add Key to CurveFloat_0**. Добавляя таким образом ключи, Вы можете построить

собственный график изменения дробной величины.

Используйте кнопки масштабирования по ширине и высоте (рисунок 2.16, *а*), чтобы увидеть полноценную картину того, как выглядит кривая. Также можно указывать точное время и значение точки кривой в соответствующих полях. Обратите внимание, что можно изменить продолжительность во времени интерполяции значения по кривой в поле **Length** (по умолчанию 5 с).

Кроме того, можно изменить характер интерполяции между ключевыми точками (ключами, keys) кривой, нажав на выбранную точку ПКМ (рисунок 2.16, *б*).

а) *б*)

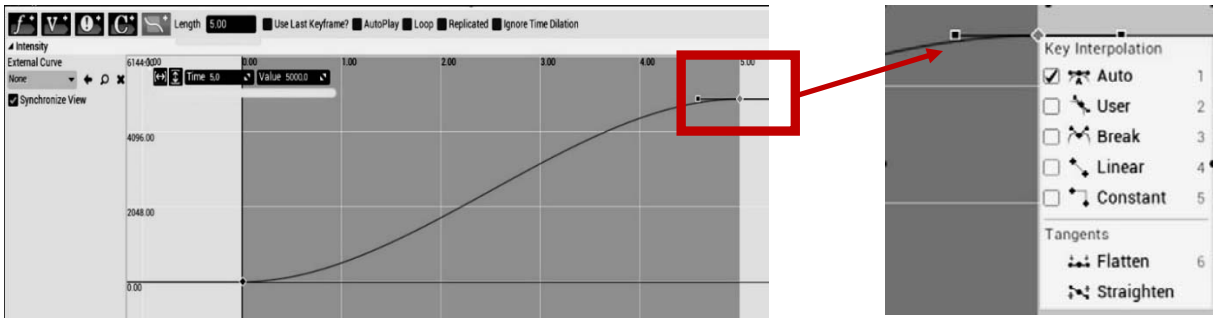

Рисунок 2.16 – Изменения значений переменных во времени в соответствии с заданным графиком

Создайте кривую продолжительностью в 10 с, с начальным ключом в Time 0 и Value 0 и вторым ключом в Time 10 и Value 5000. Не забудьте увеличить Length до 10. Установите интерполяцию на обеих точках в режим Auto.

Вернитесь в Event Graph, обратите внимание, что у нода Timeline появился выходящий пин зеленого цвета с названием Вашей кривой. Выходной пин выполнения Update выполняется каждый кадр в течение интерполяции значения по всей кривой. Соедините его с нодой **Set Intensity** компонента **Point Light**. Соедините выходной аргумент **Intensity** (или название кривой) с аргументом **New Intensity** (рисунок 2.17).

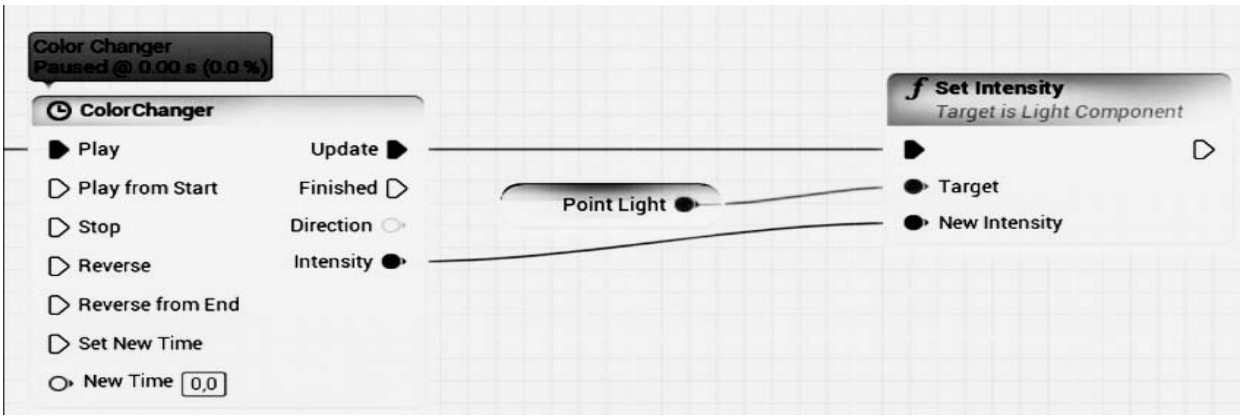

Рисунок 2.17 – Логика изменения интенсивности света в зависимости от кривой в ноде Timeline

Запустите проект, интенсивность света в Pulsar_BP должна плавно увеличиться до заданного уровня [1, 2].

Варианты задания

На основе Pulsar_BP сделайте так, чтобы на старте:

– Point Light интенсивность 0, но через 5 с после старта игры начинала пульсировать волнообразно 4 раза, с амплитудой интенсивности от 0 до 10000 в течение 10 с;

– интенсивность Point Light равномерно изменялась с 1000 до 10000 и обратно до 1000 в течение 6 с со старта игры. По окончании изменяется цвет Point Light на любой другой и повторно запускается таймлайн, цикл продолжается бесконечно;

– цвет Point Light красный. Интенсивность Point Light вырастает с 0 до 20000 в течение 3 с, затем меняется цвет на синий и убывает с 20000 до 0 в том же таймлайне. По возвращении интенсивности в 0 цвет меняется на красный. Цикл продолжается бесконечно.

3 Программирование на Blueprints. Часть II

Цель работы: создавать собственные события (events) в Blueprints и функции в Blueprints, переменные в Blueprints, добавлять package в проект и уметь детектировать коллизии в UE4.

В системе Blueprints Вы можете создавать свои собственные события (events) и функции. Это имеет смысл делать, чтобы избежать повторения однотипного кода и оптимизировать архитектуру классов вашей игры. Здесь и далее под словом событие будет подразумеваться event. Для того, чтобы создать собственное событие, нажмите ПКМ в Event Graph и наберите в поиске Add Custom Event. Дайте название событию (рисунок 3.1).

Вызов пользовательского события совершается подобно классическим правилам программирования – по написанию имени события (так же, как это происходит с вызовом функции). Вытяните соединение из события **Event BeginPlay** и отпустите, наберите в поиске имя события – в описанном примере **Start Effect** (рисунок 3.2).

Для добавления входных аргументов для событий щелкните на ноду события и в панели Details добавьте аргументы в поле Inputs. Вы можете дать названия аргументам, а также выбрать тип данных (рисунок 3.3).

Создайте следующий аргумент события: имя **New Color**, тип данных **Linear Color**. Свяжите появившийся пин в ноде ивента с входящим пином нода **Set Light Color** (рисунок 3.4). Обратите внимание, что можете изменять входной аргумент в ноде вызова события.

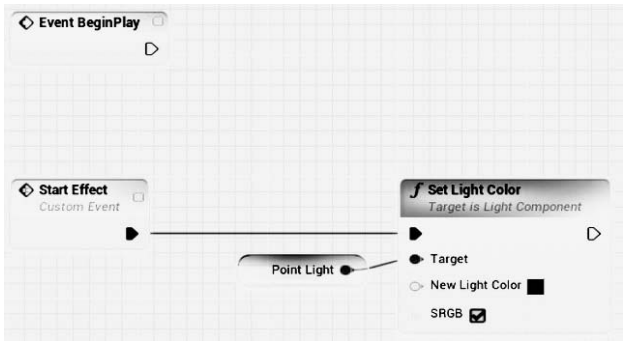

Рисунок 3.1 – Событие Start Effect и подключенная нода Set Light Color

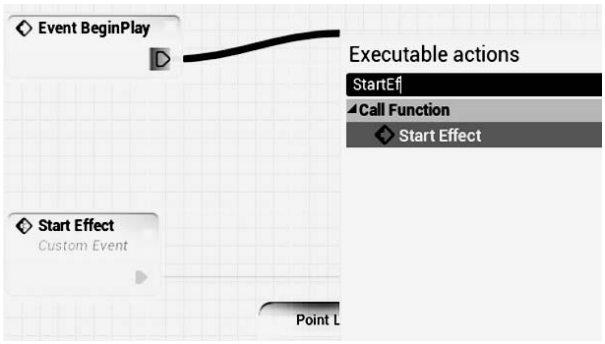

Рисунок 3.2 – Вызов события Start Effect в событии BeginPlay

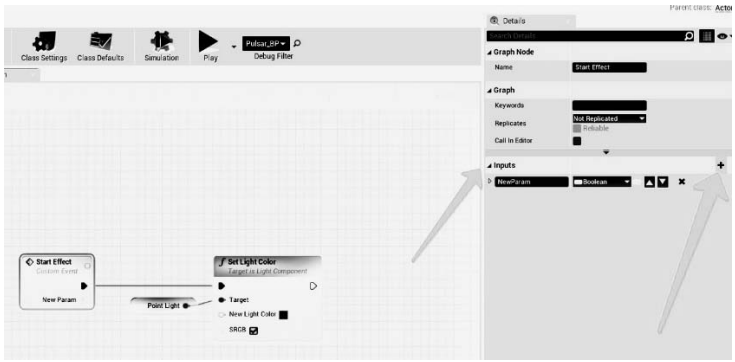

Рисунок 3.3 – Добавление входных аргументов для пользовательского события

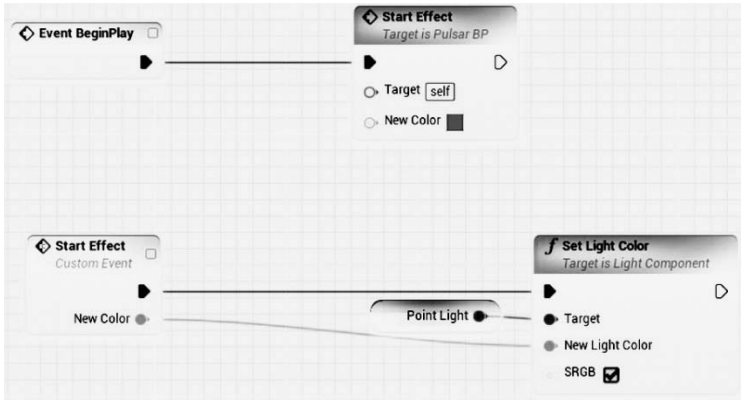

Рисунок 3.4 – Изображение аргумента события в ноде вызова

Чтобы создать собственную функцию, перейдите в панель **My Blueprint**, в панель **Functions**, нажмите на значок +, чтобы добавить новую функцию. Вы также можете дать ей собственное имя (рисунок 3.5).

В момент создания новой функции появится элемент с именем функции. Все функции описываются в отдельных окнах. Отличительная особенность функции от событий заключается в том, что функции не могут использовать некоторые ноды (например, Timeline), а также в отличие от событий могут иметь выходные аргументы. Входные и выходные аргументы функции указываются в панели **Details** при нажатии на ноду функции (рисунок 3.6).

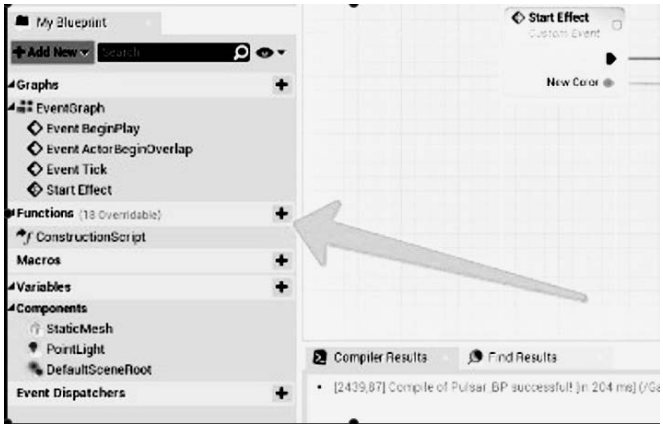

Рисунок 3.5 – Поле Functions со списком пользовательских функций

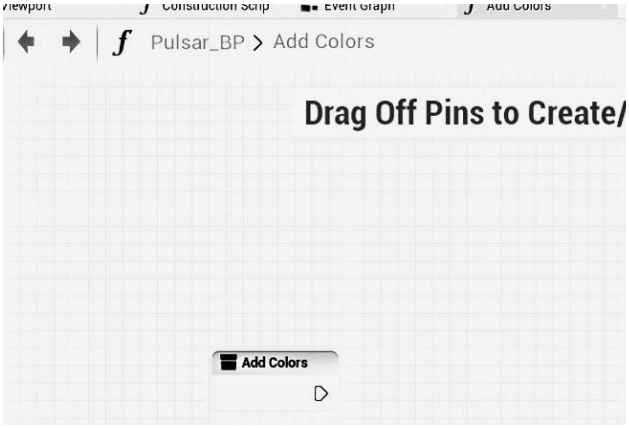

Рисунок 3.6 – Создание отдельной вкладки

Напишите функцию, которая будет складывать два цвета (**Linear Color**) и возвращать результирующий цвет (рисунок 3.7). Для этого создайте два входных аргумента с именем Color1 и Color2 типа данных **Linear Color** и выходной аргумент с именем ResultColor типа **Linear Color**.

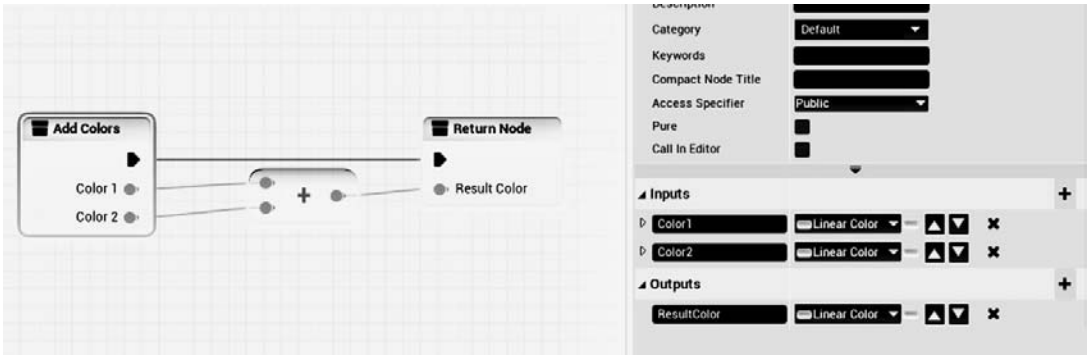

Рисунок 3.7 – Аргументы и типы данных функции

В Event Graph, вызов функции выполняется нодой с именем функции – так же, как и в ситуации с разработанными событиями и в ситуации классического программирования (рисунок 3.8).

Рисунок 3.8 – Вызов функции

Иногда не является обязательным наличие у функции execution соединения, для этого включите флаг **Pure** в панели **Details** функции. В таком случае вид

вызова функции преобразится (рисунок 3.9).

Рисунок 3.9 – Pure-функция без execution-соединения

В панели Details функции можно указать **Access Specifier** (уровень доступа) в виде классических **Public, Private и Protected**, создавать собственные переменные и давать им уровень доступа (рисунок 3.10).

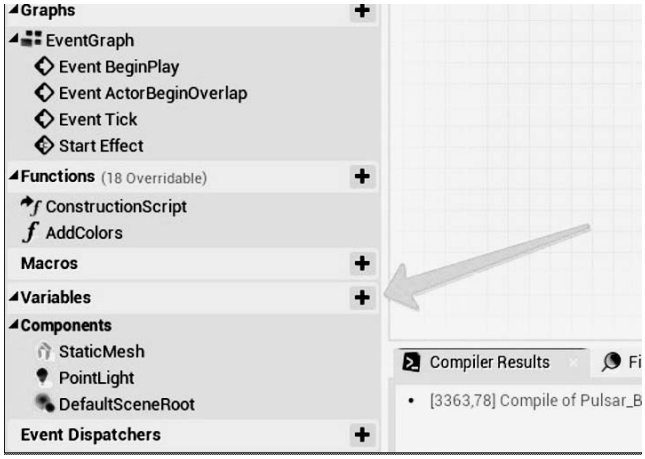

Рисунок 3.10 – Пользовательские переменные на вкладке Variables

UE4 поддерживает все основные типы данных (целые числа, дробные числа, bool-значения, строковые значения и другие) и множество комплексных типов данных, такие как векторы, списки, цвета и пр.

Давайте привнесем интерактивность в проект и добавим управляемого персонажа. После создания Blank проекта можно добавить стандартные ассеты других темплейтов предложенных в начальном шаге создания проекта.

Добавьте контент от темплейта (шаблона) **Third Person,** нажав на зеленую кнопку **Add New** окна **Content Browser,** а затем выбрав пункт **Add Feature or Content Pack** (рисунки 3.11 и 3.12).

После добавления контента появятся новые папки с файлами. В папке Third Person BP -> Blueprints можно найти блюпринт уже настроенного персонажа. Перетащите его на уровень, однако при запуске игры он будет неподвижен. Это связано с тем, что на данный момент в стандартном **GameMode** чистого проекта управляемым персонажем по умолчанию установлена парящая камера. В папке

Third Person Character находится нужный **Game Mode** для персонажа от третьего лица (рисунок 3.13).

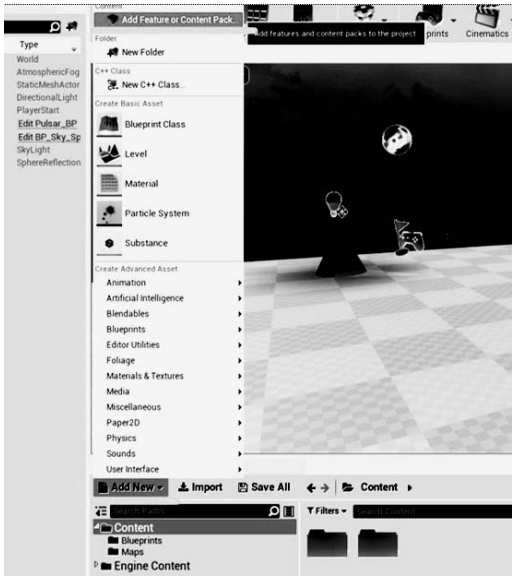

Рисунок 3.11 – Добавление дополнительных стандартных ассетов

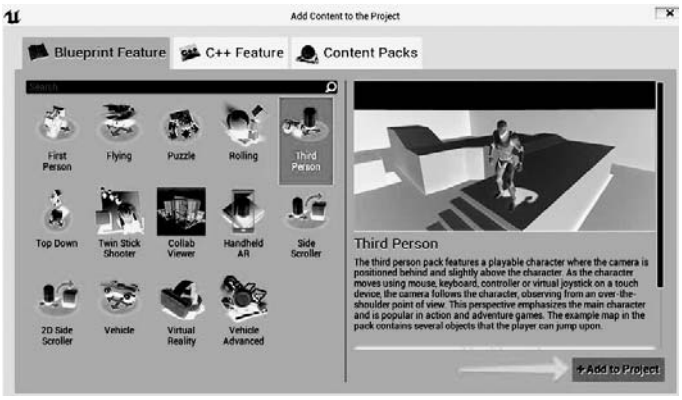

Рисунок 3.12 – Стандартные ассеты UE4

Рисунок 3.13 – Блюпринт-класс стандартного управляемого персонажа импортированного контента

Чтобы установить этот **GameMode,** нажмите на кнопку **Blueprints** верхней панели **Project Settings: Game Mode-> Select GameModeBase Class-> ThirdPersonGameMode** (рисунок 3.14).

Теперь при запуске проекта можно управлять персонажем, однако их будет двое. Это связано с тем, что **ThirdPersonGameMode** по умолчанию создает блюпринт персонажа в позиции актёра **Player Start.** Вы можете либо удалить прошлого персонажа, либо найти параметр **Auto Posses Player** в блюпринте персонажа и поставить его в значение **Player 0**. Таким образом, персонаж, выставленный на уровне, автоматически будет являться управляемым персонажем

по умолчанию (рисунок 3.15).

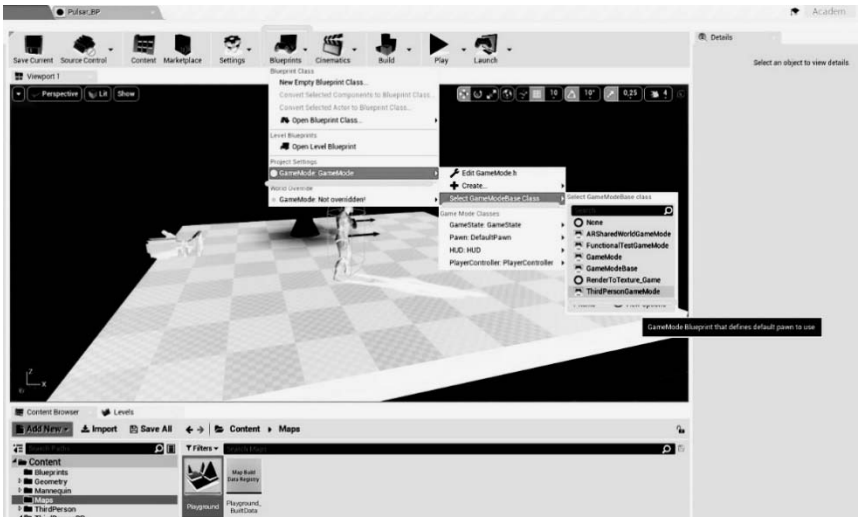

Рисунок 3.14 – Доступ к базовым настройкам логики проекта

Сделайте так, чтобы при приближении персонажа конус менял цвет на случайный. Добавьте **Sphere-**коллайдер в **Pulsar_BP** (рисунок 3.16).

Рисунок 3.15 – Поисковая строка объект ThirdPersonCharacter

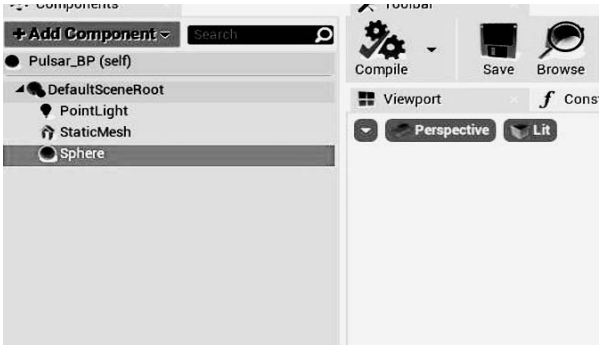

Рисунок 3.16 – Добавление компонента Sphere к классу Pulsar_BP

Убедитесь в том, что настройки данного Sphere-коллайдера установлены в состояние **Overlap All Dynamics** (рисунок 3.17).

Открыв **Event Graph** класса **ThirdPersonCharacter Blueprint**, Вы увидите, что там написана часть логики, связанная с передвижением. Ориентируясь на подобные готовые блюпринты, Вы можете потренироваться и написать логику передвижения собственного персонажа. Выберите компонент **CapsuleComponent** персонажа, пролистайте панель **Details** до блока Events. Вы можете видеть набор зеленых кнопок – это различные ивенты, обрабатывающие события, связанные с коллизией данного **CapsuleComponent**. Каждый из этих ивентов предназначен для разных «характеров» коллизии (рисунок 3.18).

Выберите **Event OnComponentBeginOverlap**. Будет создана новая нода. Аргумент OtherActor будет записывать в себя ссылку на актёра, коллайдер которого персонаж пересечёт. Нода **Cast To** является нодой, приводящей актёра к определенному классу. Первый выходной пин этой ноды будет выполняться, если

приведение типа прошло успешно. Приведите (совершите Cast) OtherActor к Pulsar_BP (рисунок 3.19) и выведите имя объекта класса.

Рисунок 3.17 – Настройки коллизии

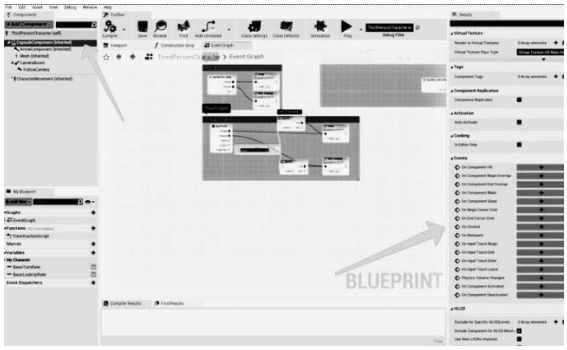

Рисунок 3.18 – Список доступных событий коллизии

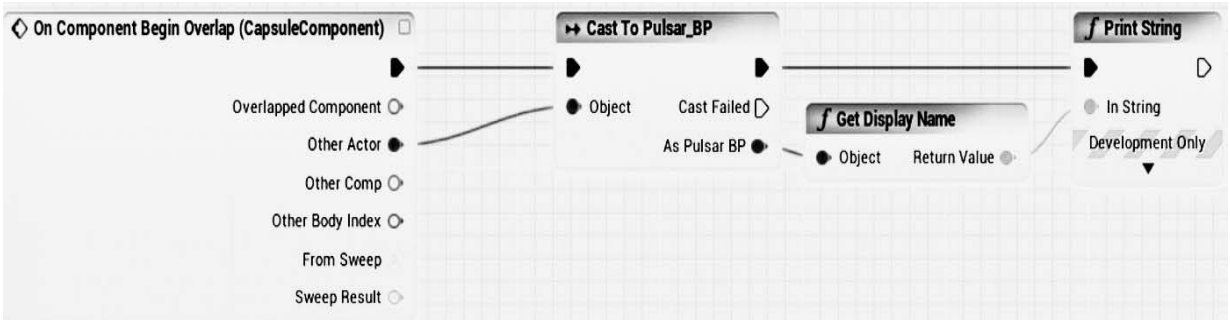

Рисунок 3.19 – Логика события при приближении персонажа

При приближении персонажа к объекту класса **Pulsar_BP** в консоль должно вывестись имя объекта (рисунок 3.20).

Рисунок 3.20 – Вывод имени класса Pulsar_BP в консоль

Реализуйте функцию **SetRandomColor** в **Puslar_BP** и вызовите её при приближении персонажа (рисунки 3.21 и 3.22).

Зачастую проще использовать теги и интерфейсы, чтобы определить объект, с которым происходит коллизия [1, 2].

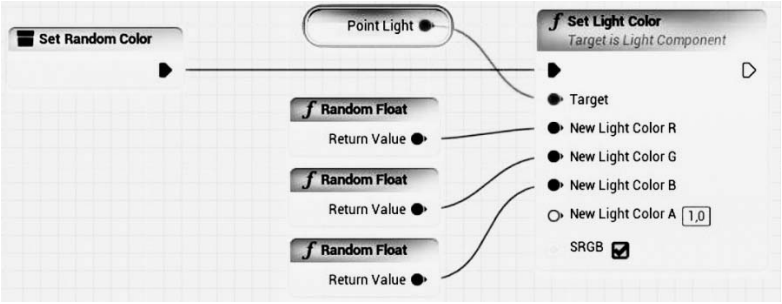

Рисунок 3.21 – Функция Random Color в Pulsar_BP, Random Float возвращает псевдослучайное значение от 0 до 1

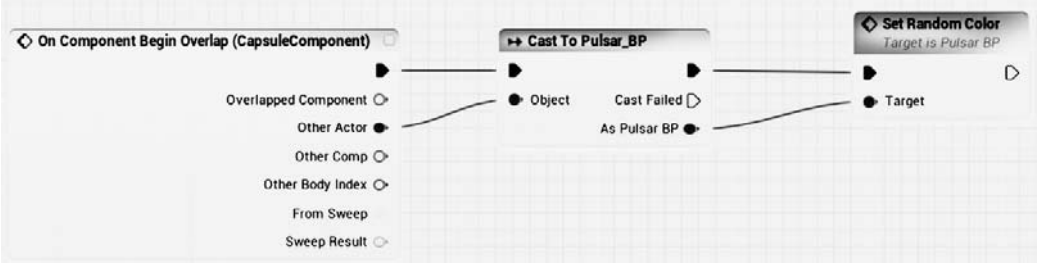

Рисунок 3.22 – Фрагмент кода из Third Character Blueprint – нода Cast To (при успешном приведении типа возвращает ссылку на объект класса, к которому совершается приведение, через которую можно вызвать все публичные функции и ивенты данного класса)

4 Работа с материалами. Material Editor

Цель работы: научиться создавать собственные материалы, создавать шейдеры для материалов и экземпляры материалов, применять материалы на сетки.

Material Editor – это нодовый редактор, позволяющий создавать шейдеры для материалов, которые могут быть применены для различной геометрии – static mesh, skeletal mesh – или быть использованы в системе частиц.

Создайте в проекте папку **Materials**, затем кликните ПКМ в папке, чтобы создать новый ассет. Выберите **Material** (рисунок 4.1).

Дайте название своему материалу (в данном примере LowPolyMaster) и откройте Material Editor двойным кликом по материалу. Откроется Material Editor. Центральная область – это непосредственно пространство, в котором выстраивается логика шейдера. В Material Editor'е используется свой набор нод. Сейчас там есть лишь одна главная выходная нода (рисунок 4.2).

Панель Viewport отображает то, как выглядит материал на тестовых мешах. Вы можете изменить превью-меш, нажав их иконки в правом нижнем углу Viewport (рисунок 4.3).

Панель Details показывает свойства выбранного нода, если ни один из нодов не выбран, отображаются общие свойства данного шейдера. В общих свойствах можно настроить тип материала (поверхность, пост-процесс и т. д.), тип

шейдинга, вид прозрачности и пр. (рисунок 4.4).

Рисунок 4.1 – Создание ассета типа Material

Рисунок 4.2 – Главное окно Material

Рисунок 4.3 – Панель Viewport

Панель Stats отображает статистику по материалу. Обратите особое внимание на то, что материал не может содержать в себе более 16 текстурных семплов (Textures Samples) (рисунок 4.5).

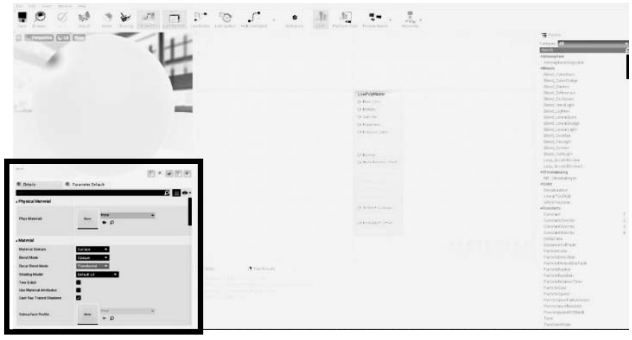

Рисунок 4.4 – Панель Details

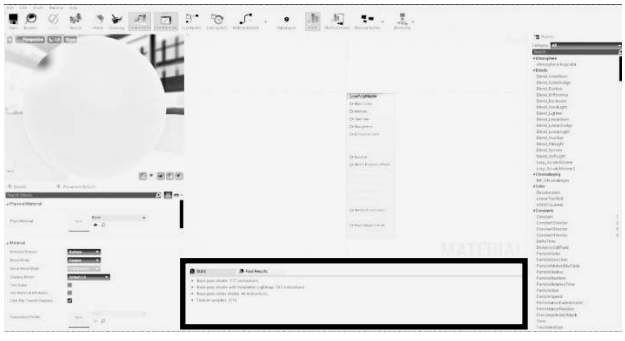

Рисунок 4.5 – Панель Stats

Панель Palette содержит список всех доступных нод для Material Editor. Вы можете перетаскивать ПКМ нужные ноды в Graph или же щелкнуть по свободному пространству в графе и найти нод, используя строку поиска (рисунок 4.6).

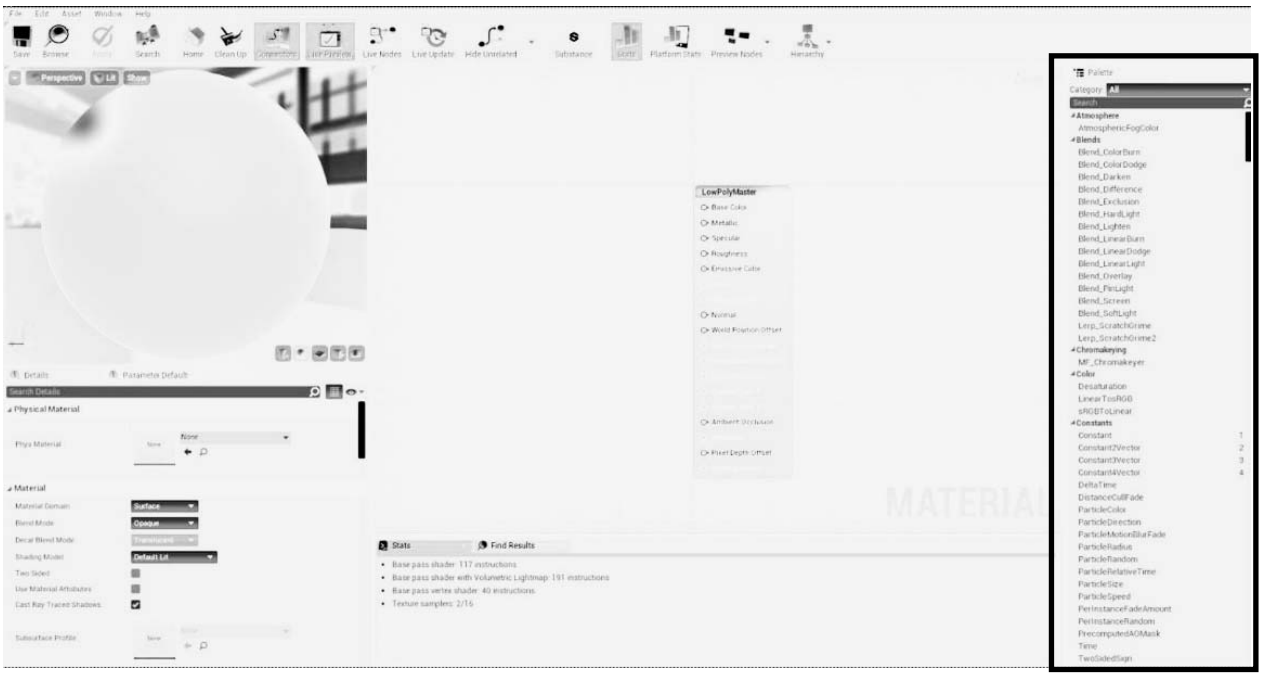

Рисунок 4.6 – Панель Palette

Сделайте простой шейдер, на основе которого будет собран материал с несколькими простыми параметрами – основным цветом (**Base Color или Albedo**), металлическим (**Metallic**) и шероховатостью(**Roughness**). В дальнейшем будут использоваться именно англоязычные термины для описания параметров материала, т. к. именно эти термины широко используются при коммуникации в индустрии видеоигры.

Большой нод с именем материала и списком пинов (**Base Color, Metallic, Specula, Roughness**). Нажмите ПКМ на пин Base Color и выберите всплывающий пункт **Promote To Parameter –** создастся нод типа Vector4, хранящий цвет RGBA. Вы можете совершить подобные действия с каждым из пинов или создать ноды вручную, нажав ПКМ в свободном месте (рисунок 4.7).

Обратите внимание, что значения отдельных нод по умолчанию можно изменять в панели **Details** после нажатия ЛКМ на нужную ноду (рисунок 4.8).

Внимательно изучите часть официальной документации Unreal Engine 4, посвященную Material Graph. В ней содержится много подсказок относительно оптимизации и подходов к проектированию универсальных материалов.

Нажмите кнопку Apply, чтобы скомпилировать шейдер материала. В Content Browser щелкните ПКМ по созданному материалу и выберите **Create Material Instance** (первый пункт). Назовите свой экземпляр материала. В данном примере ConeMaterial. Созданный экземпляр материала будет автоматически создан в той же папке, в которой находился мастер материал (рисунок 4.9).

Открыв экземпляр материала, можно настроить его параметры. Отдельные экземпляры (инстансы) материала применяются к разным моделям. Никогда не применяйте разные оригинальные (родительские) материалы к Вашим мешам, используйте инстансы для оптимизации проекта.

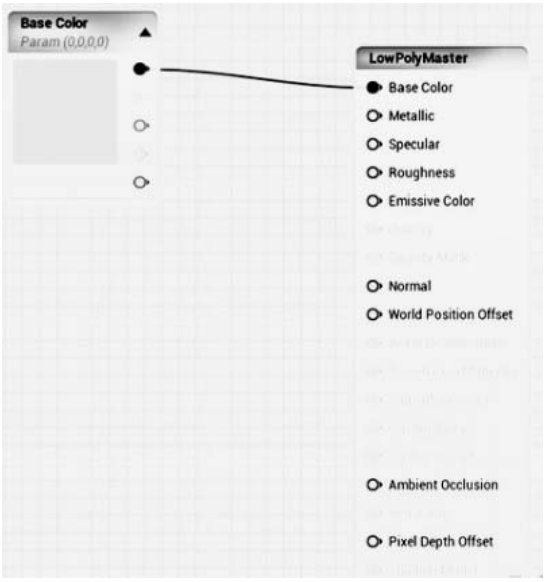

Рисунок 4.7 – Нода BaseColor

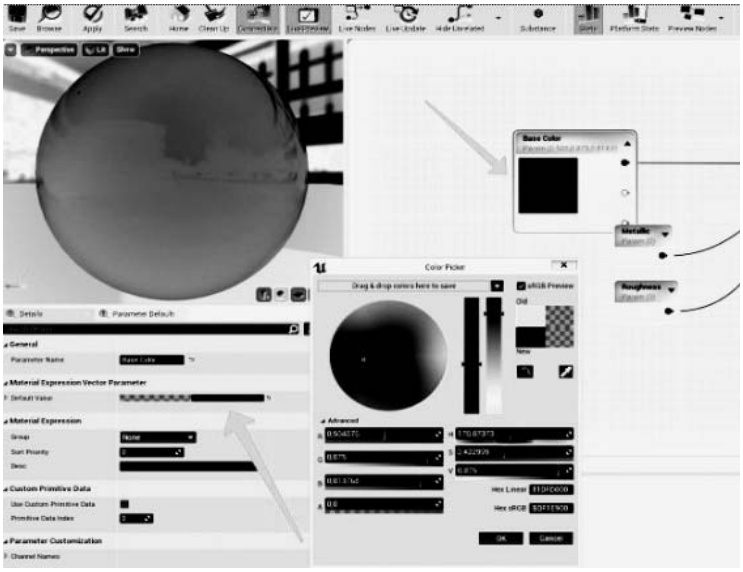

Рисунок 4.8 – Панель изменения значений переменных нодов в панели Details

Чтобы применить материал на меш, можно просто перетащить его из Content Browser на выбранный меш. Вы также можете указать необходимый материал из окна Blueprint или через панель Details в основном окне Unreal Engine (рисунок 4.10).

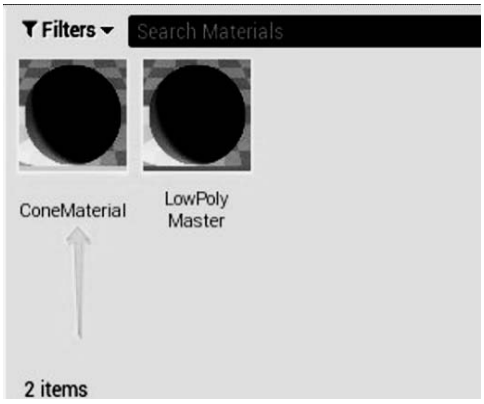

Рисунок 4.9 – Созданный материал

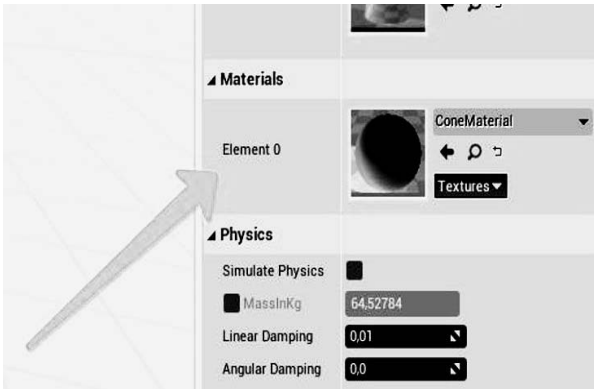

Рисунок 4.10 – Указание материала через поле Materials выбранного меша

Созданные вручную ноды типов данных в **Material Graph** (например, Vector или **Texture Sample)** не являются переменными по умолчанию. Для этого нужно кликнуть на нужную ноду и выбрать **Convert To Parameter** (рисунок 4.11).

Подробнее про **Material Editor** вы можете узнать в официальной документации [1, 2].

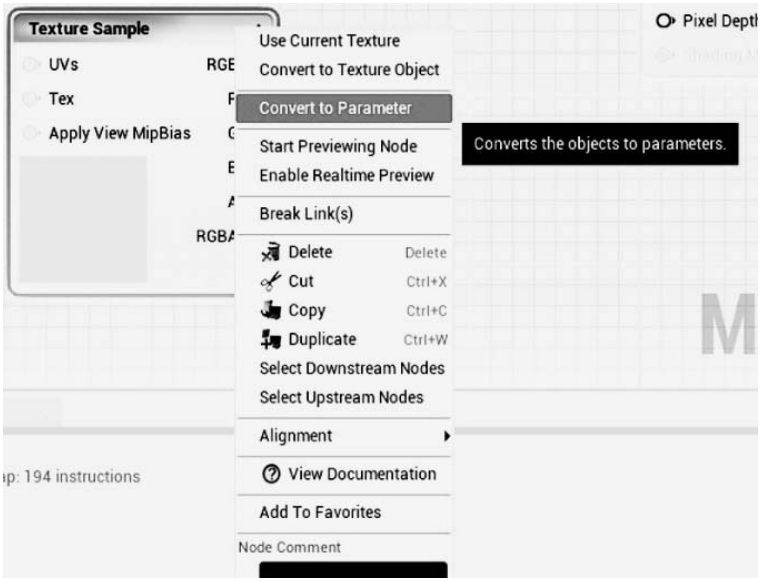

Рисунок 4.11 – Конвертация ноды в переменную через выбор опции Convert To Parametr

Варианты задания

1 Сделать полноценный PBR-материал с текстурными картами Albedo, Normal Map, Displacement Map.

2 Сделать материал, цвет Albedo которого плавно меняется с черного на белый в реальном времени.

5 Работа с контентом в UE 4 II. Animation Blueprint

Цель работы: научиться импортировать контент (собственный или из Marketplace) и создавать блюпринт-класс из нескольких акторов.

Импорт ассетов может происходить несколькими способами: можно перетаскивать ассеты в соответствующие папки или использовать кнопку **Import** в окне **Content Browser**. Вы также можете воспользоваться **Unreal Marketplace** – магазином контента для разработчиков UE 4. Там можно найти бесплатный контент.

Воспользуемся бесплатным контент-паком из Marketplace: Ancient Treasures за авторством Dekogon Studious. Найдите данный ассет на странице https://www.unrealengine.com/marketplace/.

После импорта ассетов перейдите в папку Meshes и поставьте на сцену сундук и крышку от него. Часто возникает ситуация, когда необходимо создать блюпринт-класс из нескольких актёров. Это логично в случае с сундуком, если в дальнейшем нужно написать логику открытия и закрытия крышки сундука. Выделите **StaticMeshActor** сундука и его крышки в **World Outliner**, затем выберите **Convert Selected Components to Blueprint Class** меню Blueprints.

Назовите блюпринт сундука **Chest_BP** (рисунок 5.1).

Рисунок 5.1 – Опция Convert Selected Components to Blueprint Class

Данный блюпринт сундука будет использоваться в следующем занятии.

Animation Blueprint – это специализированный блюпринт, предназначенный для управления анимациями **Skeletal Mesh**, т. е. мешей персонажей.

Стандартный манекен Unreal Engine уже имеет набор анимаций и запрограммированный Animation Graph, на данном занятии необходимо создать Animation Blueprint, но в качестве контента будем использовать стандартные анимации манекена.

Чтобы создать Animation Blueprint, нажмите ПКМ в свободном месте, выберите вкладку **Animation**, далее **Animation Blueprint**. Будет открыто окно, в котором нужно указать **Skeleton**, к которому будет приписан блюпринт. Выберите скелет стандартного манекена – **UE4_Mannequin_Skeleton** и нажмите **OK** (рисунок 5.2).

Рисунок 5.2 – Окно выбора скелета меша

Откройте созданный **Animation** блюпринт. В окне **AnimGraph** размещаются

анимации, стейт-машины и прочие ноды, обрабатывающие анимации, и описываются правила перехода между ними. В окне **Event Graph** пишется логика, в ней могут использоваться специфические для анимаций ноды. В окне Animation Graph доступна вкладка Event Graph, в которой можно писать специфическую логику анимаций (рисунок 5.3).

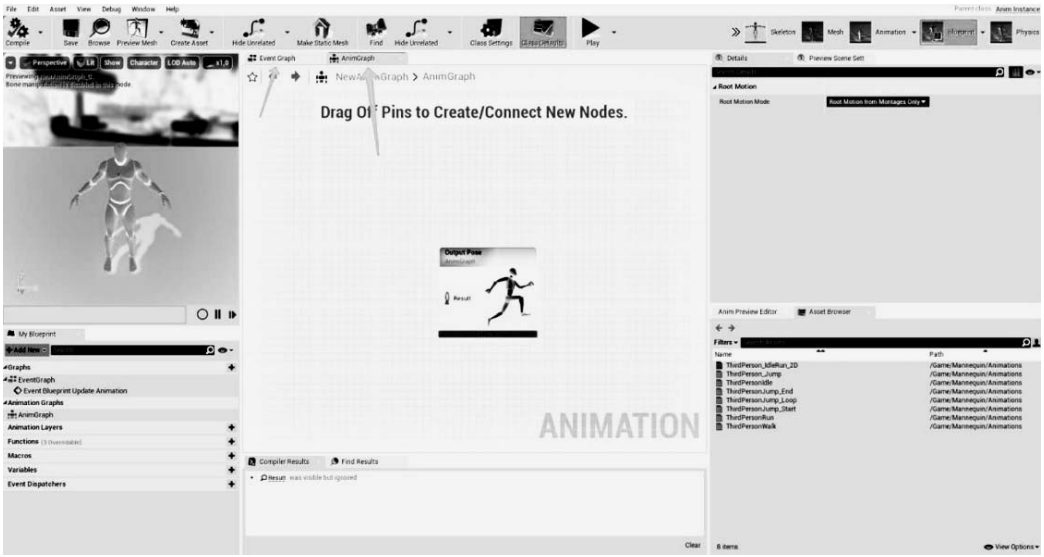

Рисунок 5.3 – Окно Animation Graph

Находясь в окне **AnimGraph**, создайте ноду **State Machine**. Для этого щелкните ПКМ в свободном месте и введите в поиске нод **Add New State Machine**. Вы можете изменить название State Machine в панели Details. Свяжите выход стейт-машины и вход ноды **Output Pose**. **Output Pose** – это всегда выходная нода любого Animation Blueprint. Желтоё предупреждение **Warning!** сообщает о том, что созданная стейт-машина пуста (рисунок 5.4).

Чтобы добавить состояния в стейт-машину, нужно щелкнуть 2 раза на ноду стейт-машины. Нода Entry является стандартной, и с нее начинается выполнение стейт-машины. Вы можете перетаскивать анимации из панели Asset Browser в окно стейт-машины. Вытягивайте ЛКМ связи из границ нод-анимаций (рисунок 5.5).

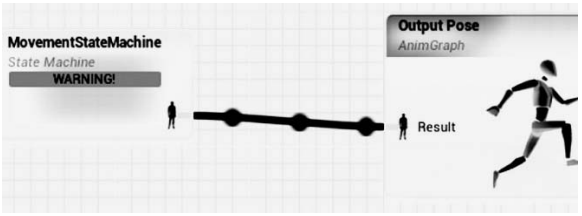

Рисунок 5.4 – Начальная State Machine

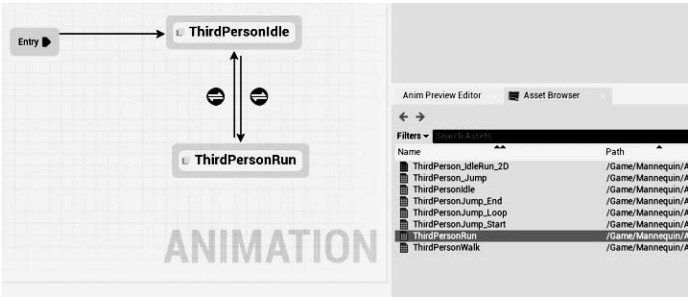

Рисунок 5.5 – Доступные анимации

Нажав двойным ЛКМ по связи, откройте окно для настройки логики связи перехода. Добавьте переменную типа float, дайте ей название (рисунок 5.6).

В данном примере имя переменной Anim_Velocity.

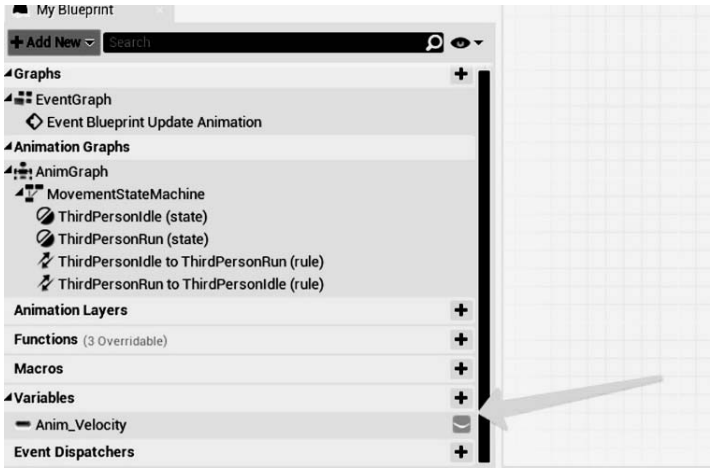

Рисунок 5.6 – Пользовательские переменные в блоке Variables

В связи, **выходящей** из анимации **ThirdPersonIdle**, добавьте проверку значения больше или равно 0,2 переменной Anim Velocity (рисунок 5.7).

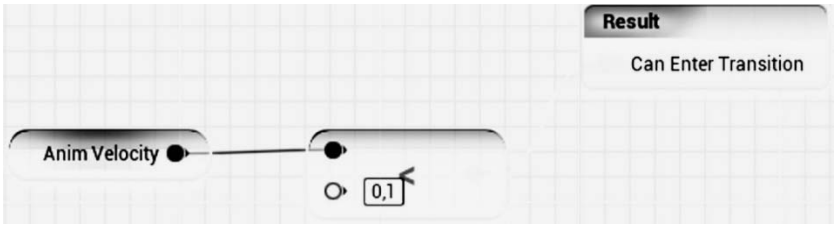

Рисунок 5.7 – Логика перехода анимации

В связи, **входящей** в анимацию **ThirdPersonIdle,** добавьте проверку значения меньше 0,1 переменной Anim Velocity (рисунок 5.8).

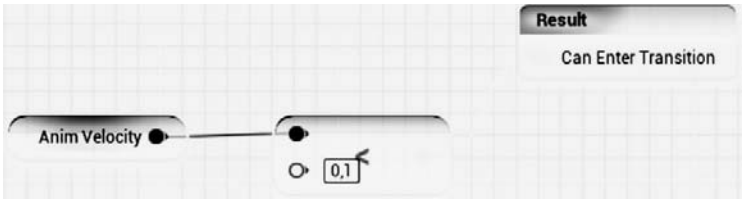

Рисунок 5.8 – Логика перехода анимации

Значению переменной Anim Velocity присвойте значение скорости персонажа, и в случае, если скорость больше определенного значения, то анимация будет переходить из состояния Idle в Running. Теперь считайте значение скорости с персонажа. Для этого перейдите в окно Event Graph. Нода Event Blueprint Update Animation является аналогом Tick Update из обычных блюпринтов. Нода TryGetPawn Owner получает ссылку на объект, к которому назначен данный Animation Blueprint.

Добавьте переменную **Owner Ref** типа **Actor (Reference)**, в нее запишите

ссылку на pawn через **Try Get Pawn Owner**.

В ноде **Update Animation** получите длину вектора **velocity** и присвойте её значение переменной **Anim_Velocity** (рисунок 5.9).

Рисунок 5.9 – Присвоение переменной Anim_Velocity значения длины вектора

Скомпилируйте **Animation Blueprint** и присвойте к **ThirdPersonCharacter** (рисунок 5.10). Запустите игру и проверьте смену анимаций при движении персонажа [1, 2].

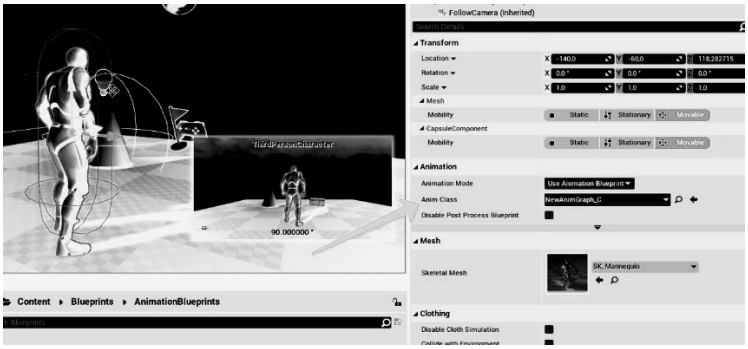

Рисунок 5.10 – Выбор Animation Blueprint в поле Animation, переменная AnimClass

6 Программирование на Blueprints. Часть III

Цель работы: научиться создавать Blueprint Interface и назначать его блюпринтам, вызывать методы Blueprint Interface, наследовать блюпринт-классы, подписываться на эвенты.

Интерфейс – это удобный способ обозначать методы, которые должны быть имплементированы у разных типов объектов. Чтобы создать блюпринт-интерфейс, добавьте новый ассет – выберите вкладку **Blueprints-> Blueprints Interface** (рисунок 6.1).

Открыв **Interface Blueprint**, можно перечислить все методы, а также их входные и выходные аргументы, которые можно будет использовать в качестве интерфейса (рисунок 6.2).

Перейдите в блюпринт сундука, созданный на прошлом занятии. Нажмите на **Class Settings** и добавьте свой Interface Blueprint в список интерфейсов

(рисунок 6.3). Обратите внимание, что созданный класс может содержать имплементацию нескольких интерфейсов.

Рисунок 6.1 – Путь создания интерфейса Blueprint

Рисунок 6.2 – Настройка интерфейса

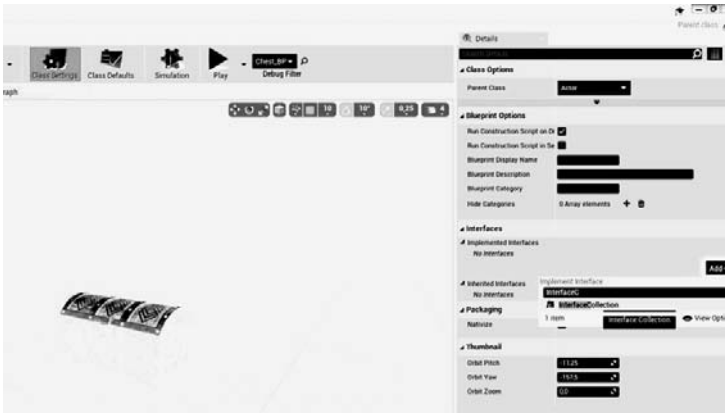

Рисунок 6.3 – Добавление интерфейсов к классу

Теперь можно реализовать события, описанные в Blueprint Interface, и вызывать их из другого блюпринта, не производя проверку типа блюпринта или не производя Cast (рисунок 6.4).

Реализуйте следующую игровую механику: при приближении игрока к

сундуку откроется крышка сундука. Пошаговая инструкция к достижению данного результата:

– добавьте Collision Sphere в блюпринт сундука;

– сделайте проверку столкновения с сундуком из блюпринта персонажа.

Рисунок 6.4 – Событие, созданное на основе интерфейса

Используйте нод **Does Implement Interface.**

В реализации интерфейса Activate Object запрограммируйте открытие крышки через ноду Timeline (рисунки 6.5 и 6.6).

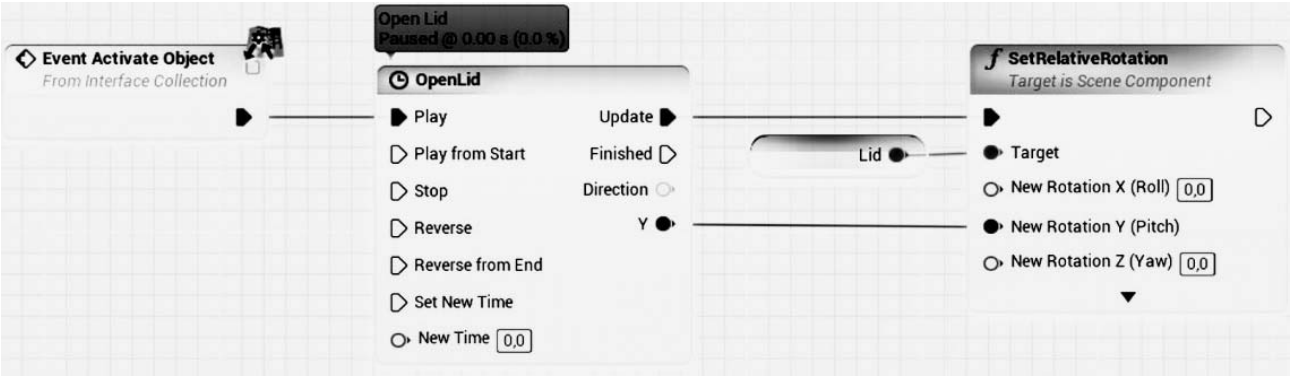

Рисунок 6.5 – Управле углом поворота крышки сундука

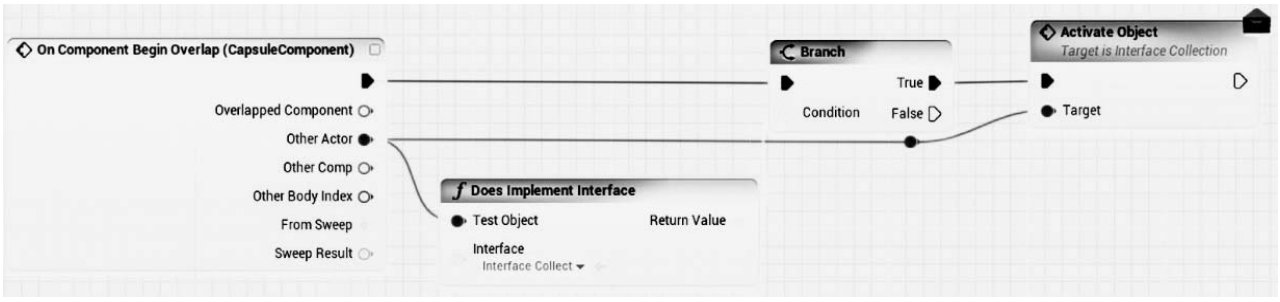

Рисунок 6.6 – Проверка нодой Does Implement Interface объекта на наличие определенного интерфейса

Обратите внимание, что вызов интерфейса не требует каста блюпринта в определенный тип. Таким образом, можно сильно упростить архитектуру кода игры. Не забывайте также, что нод **CastTo** является «тяжелым». Попробуйте добавить интерфейс Activate Object в блюпринт Puslar_BP.

Наследование в системе Blueprint работает так же, как и в классическом программировании. От любого созданного блюпринт-класса можно сделать дочерний блюпринт-класс, переопределить его методы и т. д. Чтобы создать дочерний блюпринт-класс, нажмите на желаемый блюпринт ПКМ и выберите **Create Child Blueprint Class** (рисунок 6.7).

Чтобы переопределить функции родительского класса в дочернем блюпринте, используйте кнопку **override** в блоке Functions (рисунок 6.8). Чтобы выполнить родительскую реализацию функции или события, щелкните ПКМ по событию в дочернем классе и выберите **Add Call to parent function.**

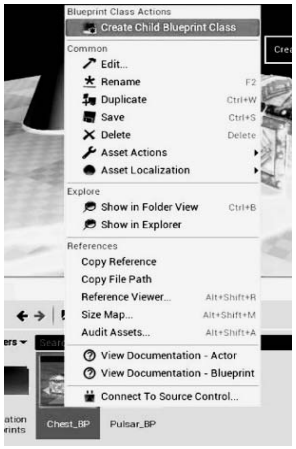

Рисунок 6.7 – Создание дочернего класса блюпринта через опцию Create Child Blueprint Class

Рисунок 6.8 – Список событий и функций, которые можно переопределить в дочернем классе, с помощью кнопки Override

Функцию делегатов в системе Blueprints выполняют **Event Dispatchers**, подписав одно или более событие на **Event Dispatcher**, можно вызвать эти события одновременно, как только **Event Dispatcher** будет вызван. Добавьте **Event Dispatcher**, нажав на + в разделе **Event Dispatchers** (рисунок 6.9).

Добавьте ноду вызова диспетчера по нажатию на клавишу E (рисунок 6.10). Затем подпишитесь на данный диспетчер из блюпринта Chest_BP (рисунок 6.11) [1, 2].

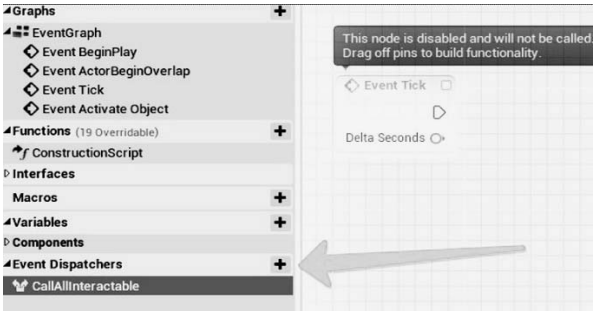

Рисунок 6.9 – Вызов Event Dispatchers

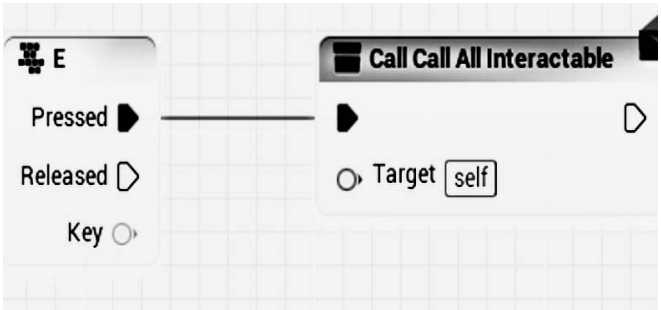

Рисунок 6.10 – Пример вызова ноды E при нажатии на клавишу клавиатуры латинской E

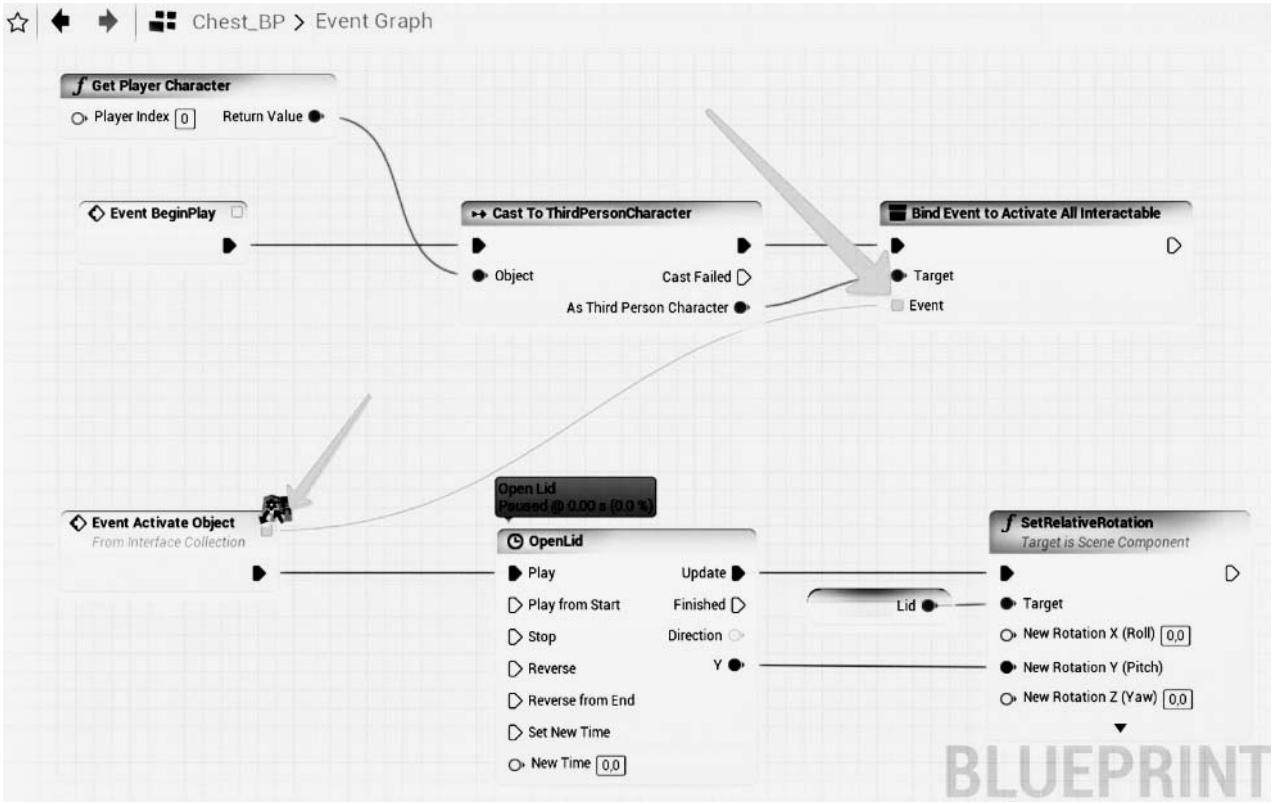

Рисунок 6.11 – Пример обозначения события при соединии их красным узлом с нодой Bind Event to

7 Искусственный интеллект. Инструменты написания AI в UE4: State Machine, Behaviour Tree, Navigation

Цель работы: научиться создавать NavMesh и использовать ноды AI, создавать Behaviour Tree и Blackboard, создавать патрулирующего персонажа, который останавливается на каждой точке патруля, и персонажа, преследующего игрока.

В играх под AI понимается реализация интеллектуальных (кажущихся интеллектуальными) поведений для различных агентов. Типы AI могут существенно различаться по:

– способу обработки задач;

– получения или восприятия информации о мире;

– интерпретации геометрического пространства мира;

– способу воспроизведения поведения, приближенного к человеческому.

Главным компонентом любого игрового AI является формальная система принятия решений, которая выполняет определенные действия, оценивая окружающую обстановку.

Основные способы реализации системы принятия решений:

– конечные автоматы (Finite-State machine);

– поведенческие деревья (Behavior tree);

– кривые выгоды (Utility AI).

Конечные Автоматы (FSM) – это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Программист задает набор состояний и переходов между ними. Самый проверенный временем способ создания ИИ.

Поведенческие деревья (Behaviour Trees) – это дальнейшее развитие идеи конечных автоматов. Основная идея заключается в том, что в один момент времени персонаж может находиться только в одном состоянии. Но за счет иерархической структуры и разнообразных узлов проектирование поведения сильно упрощается.

Utility AI – модель, которая выявляет доступные для ИИ действия и начисляет им очки в зависимости от складывающихся обстоятельств, тем самым выбирая наиболее выгодное.

Зачастую различные поведения AI можно разбить на отдельные состояния (states). Например, AI игрового агента может иметь состояния «Поиск врагов», «Поиск аптечки», «Покой», «Атаковать». Если мы определим правила переходов между этими состояниями, то эти состояния могут быть трансформированы в конечный автомат (Finite State Machine). Это простая техника, которая применима во множестве ситуаций. Стейт-машины исторически были одним из основных подходов к проектированию алгоритмов AI.

В среде Unreal Engine 4 можно реализовать свою систему FSM для AI либо использовать стандартный инструментарий для написания Behaviour Tree. В ситуации, когда планируется, что AI в игре будет комплексным, рекомендуется использовать Behaviour Trees (рисунок 7.1).

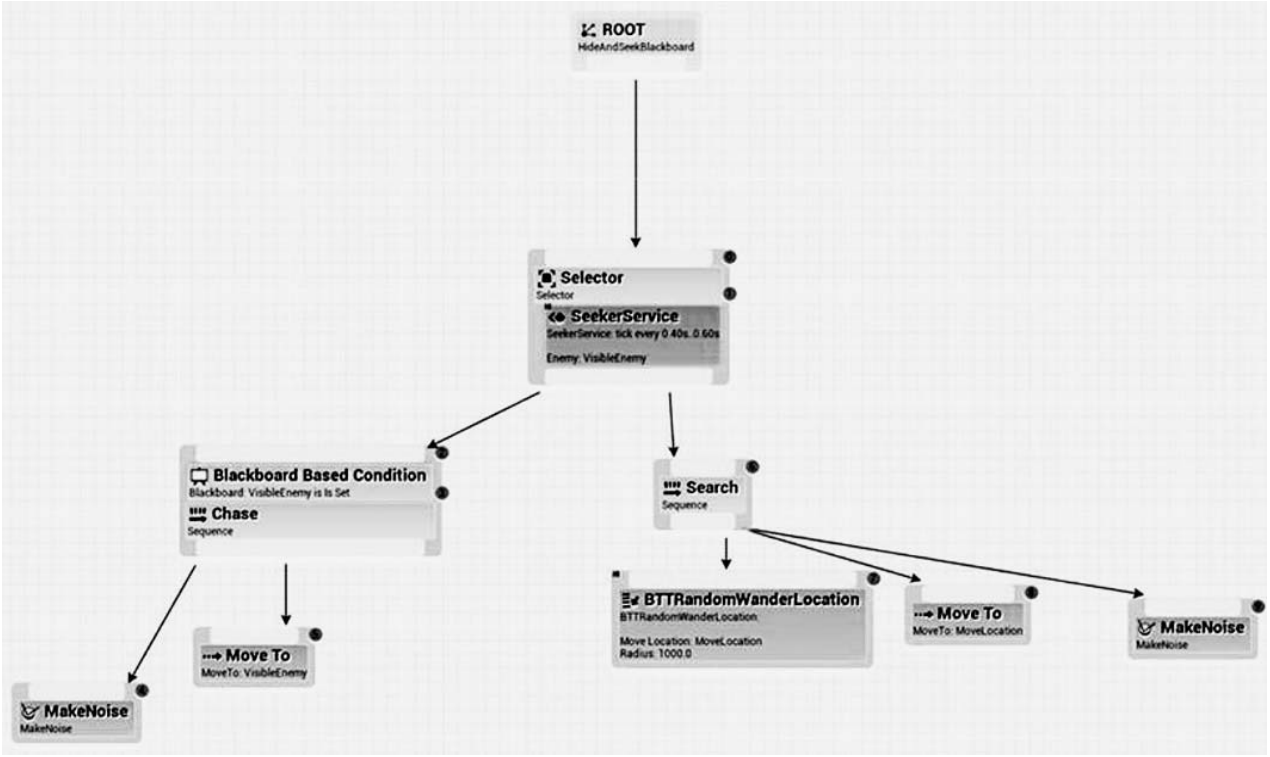

Рисунок 7.1 – Пример дерева поведения в Unreal Engine 4

Нода **Root** – уникальная нода дерева поведения. Начальная точка. К нему нельзя прикрепить декораторы и сервисы (рисунок 7.2).

Нода **Task** – листья дерева. Они сообщают, что «делать», и не имеют выходных соединений (рисунок 7.3).

Нода **Sequence** – этот узел выполняет своих «детей» слева направо и останавливает выполнение цепочки, если один из детей не может выполниться (рисунок 7.4).

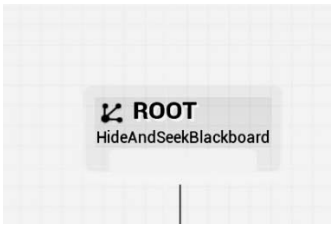

Рисунок 7.2 – Нода Root

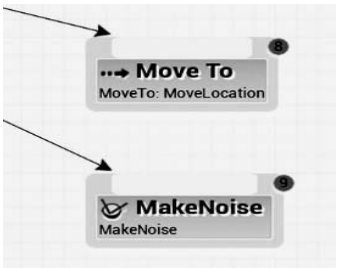

Рисунок 7.3 – Нода Task

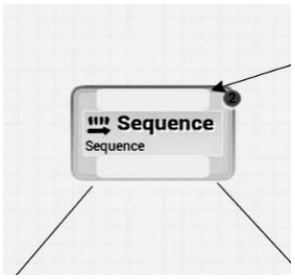

Рисунок 7.4 – Нода Sequence

Нода **Selector** выполняет своих «детей» слева направо и останавливает выполнение, когда один из «детей» успешно заканчивает свою операцию. Если один из «детей» успешно выполнился, то Selector возвращает true, если все «дети» фейлят, то возвращает false (рисунок 7.5).

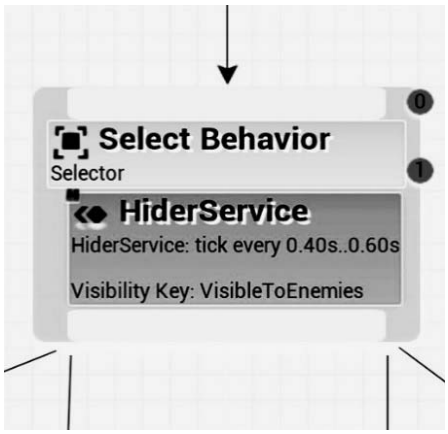

Рисунок 7.5 – Нода Selector

Нода **Decorators** – то же самое, что и условия. Они присоединяются к другим нодам и решают, будет ли выполняться ветка или лист дерева (рисунок 7.6).

Observers – это свойства декоратора, через которые можно связываться со значениями из Blackboard (блекборда). Также в Observers указываются условия прекращения выполнения узла или ветки дерева (рисунок 7.7).

Blackboards – память AI, которая хранит ключи и значения, используемые деревом поведения (рисунок 7.8).

Services – это модуль, присоединяемый к нодам. Он сообщает частоту

выполнения какой-либо ноды, если эта нода вообще выполняется. Они нужны для обновления проверок значений или обновления значений в Blackboard (рисунок 7.9).

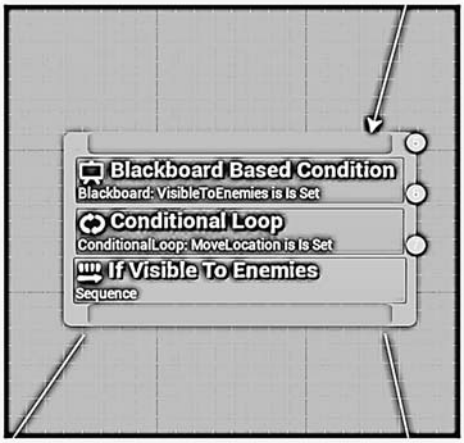

Рисунок 7.6 – Нода Decorator

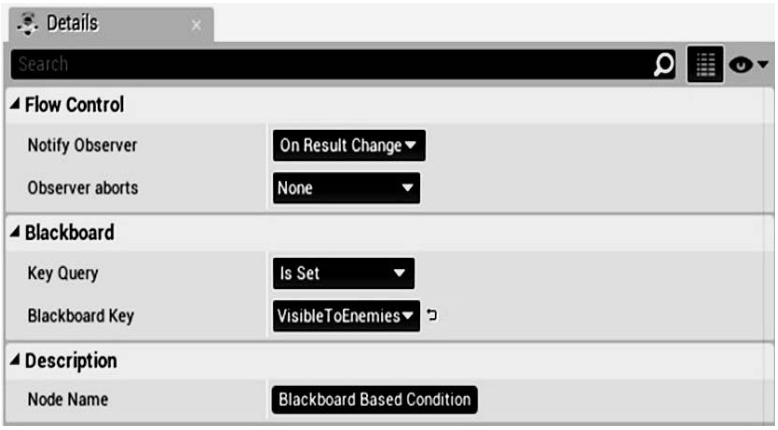

Рисунок 7.7 – Details панель Blackboard со списком Observers

Рисунок 7.8 – Примерный вид Blackboard

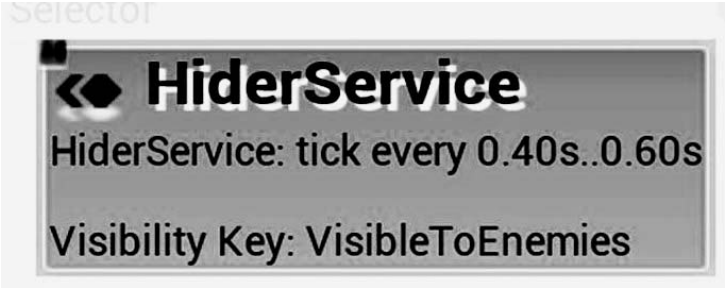

Рисунок 7.9 – Нода Services

AI может перемещаться по миру, только если он понимает, как это делать. В отличие от человека, AI может воспринимать мир вокруг себя в очень упрощенном формате. Процесс идентификации путей по локации называется **Pathfinding.** Три основных типа навигационных систем – это **Navigation Points, Navigation Grids** и **Navigation Meshes**.

Поставьте актёры Nav Mesh Bounds Volume на уровень (рисунок 7.10). Вы можете изменить масштаб актёра, чтобы покрыть нужную площадь уровня.

Откройте пункт Build и выберите пункт Build Paths, чтобы сгенерировать Nav

Mesh. При нажатии на клавишу P будет подсвечена область Nav Mesh, агенты, управляемые AI, смогут перемещаться только по зеленой зоне (рисунок 7.11).

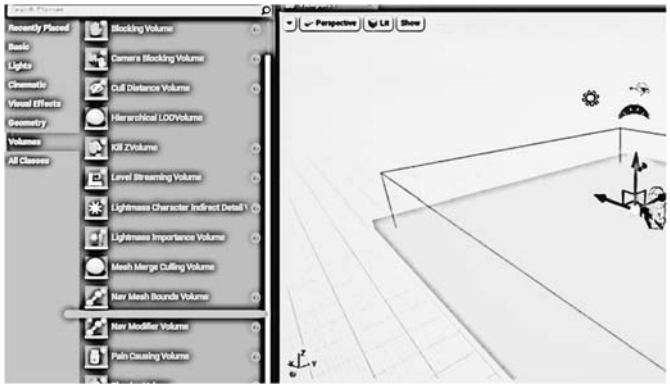

Рисунок 7.10 – Границы NavMesh Bounds Volume в окне Viewport

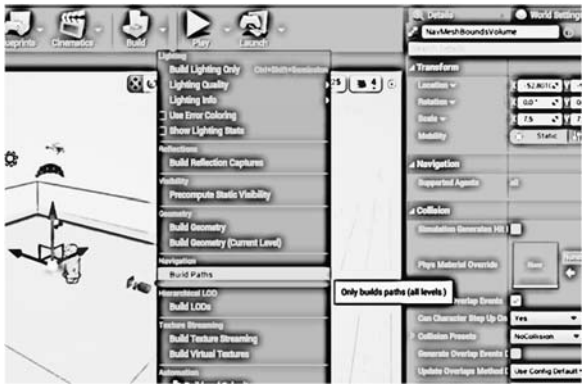

Рисунок 7.11 – Выбор опции Nav Mesh для создания Build Paths

Обычно для объектов управляемого AI используются классы Pawn и Character. Класс Character наследуется от Pawn, и есть смысл в его использовании, если Ваш агент – это классический гуманоидный персонаж или персонаж, который перемещается на ногах. Если у Вас игра про морские бои, где корабли управляются AI, то логичнее использовать Pawn актёра.

Добавьте Character Actor, назначьте ему скелетал меш (в данном примере стандартный манекен) и превратите его в Blueprint- класс. Откройте созданный блюпринт (рисунки 7.12–7.14).

Рисунок 7.12 – Созданный блюпринт-класс персонажа

Используя ноду **AI Move To**, создайте код, который заставляет агента преследовать игрока (см. рисунок 7.13).

Для того чтобы персонаж плавно поворачивался в направлении движения, проверьте следующие флаги в компоненте Character Movement – флаг **Orient Rotation To Movement** в **true** и **Use Controller Rotation Yaw** в **false** (рисунки 7.15 и 7.16).

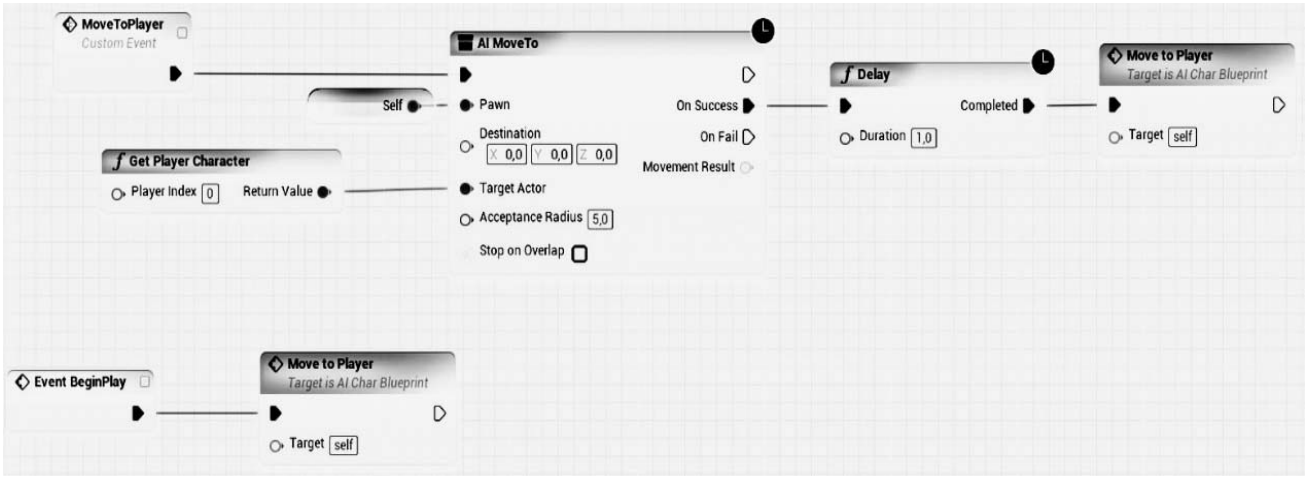

Рисунок 7.13 – Нода AI Move To

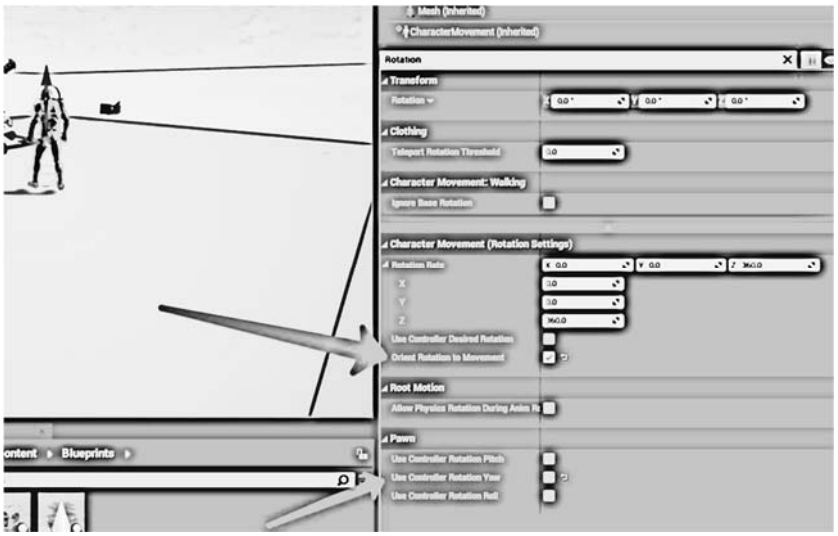

Рисунок 7.14 – Флаги, на которые необходимо обратить внимание, чтобы персонаж плавно поворачивался в направлении движения

Существует множество нод, специализирующихся на AI. Подробнее о таких нодах смотрите в официальной документации.

Напишите подобную логику следования персонажа в определенную позицию, но с использованием дерева поведения. Сделайте отдельную папку AI, переместите туда блюпринт агента, нажмите ПКМ и в пункте Artificial Intelligence выберите **Behavior Tree** и **Blackboard**. Создайте новый блюпринт для агента, который будет управляться деревом поведения, а не напрямую функциями из блюпринта (рисунок 7.15).

Класс **AI_Controller** является классом, в котором принято писать специфическую для агента логику. Часть нод, например, ноды, связанные с созданием дерева поведения, можно создать только в классе AI_Controller и наследуемых от него. Создайте дочерний класс **AI_Controller** (рисунок 7.16). Имеет смысл не писать логику в **AI_Controller**, а писать в дочерних классах, это поможет сделать компонентную систему контроллеров. Можно будет повторно

использовать контроллеры для разных типов персонажей-агентов.

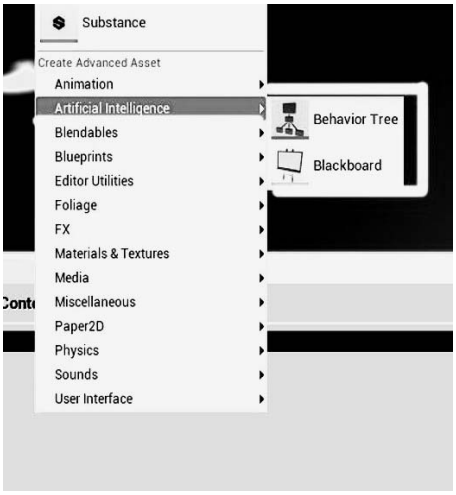

Рисунок 7.15 – Создание ассета Behaviour Tree и Blackboard

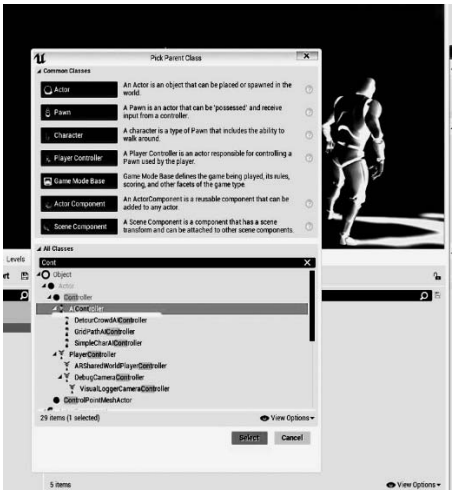

Рисунок 7.16 – Создание Blueprint-класса, наследуемый от AI_Controller

Добавьте созданный контроллер-класс в раздел Pawn агента (рисунок 7.17).

Откройте **Event Graph** созданного и присвоенного контроллера. Нода **Run Behavior Tree** запускает выполнение дерева поведения. Нода **Use Blackboard** подключает использование конкретного блекборда (рисунок 7.18).

Откройте **Behaviour Tree** и добавьте следующие ноды (рисунок 7.19): ноду **Sequence** и от нее ноды **MoveTo** и ноду **Wait**. Ноды добавляются ПКМ. Важно помнить, что ноды выполняются **сверху вниз, слева направо**. У данных нод есть стандартные аргументы. У ноды **MoveTo** – это переменная из Blackboard, к которой нужно двигаться (Vector-позиции или ссылка на объект). У ноды **Wait** это количество секунд ожидания.

Рисунок 7.17 – Для нахождения переменной AI_Controller Class в панели Details

Откройте созданный Blackboard, добавьте переменную, в которой будет храниться ссылка на игрового персонажа. Тип переменной Object, класс Actor. (рисунки 7.20 и 7.21).

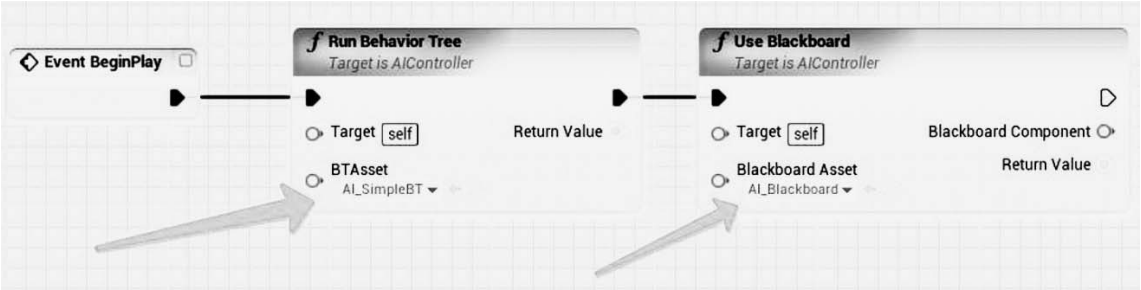

Рисунок 7.18 – Создание нодой Run Behaviour Tree экземпляра дерева поведения и его запуск, включение нодой Use Blackboard использования Blackboards

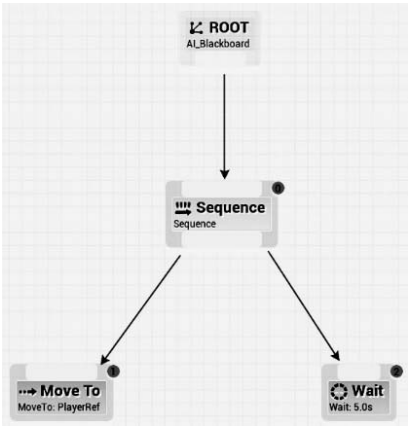

Рисунок 7.19 – Простое дерево поведения

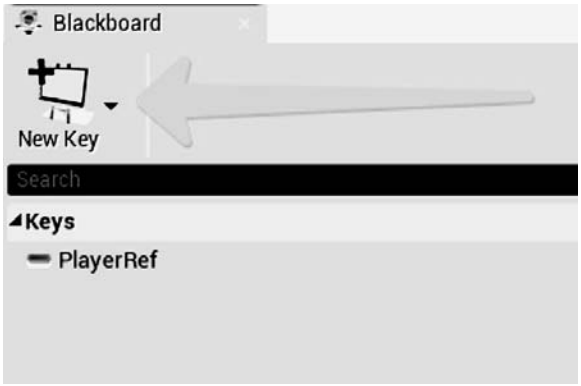

Рисунок 7.20 – Добавление новой кнопки

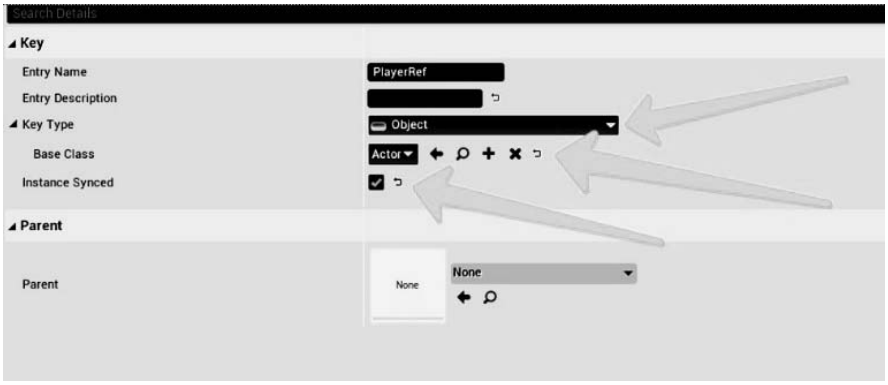

Рисунок 7.21 – Изменение характеристики переменной Object

Обратите внимание на флаг Instance Synced. Включенный флаг означает, что значение данной переменной будет одно во всех инстансах данного Blackboard. В случае с ссылкой на игрового персонажа логично поставить данный флаг в true.

Проверьте подставленное значение в **Behaviour Tree** нода **MoveTo** (по умолчанию первая переменная blackboard ставится автоматически во все пустые аргументы ноды дерева). Выбрав ноду Move To, в его параметрах следуетуказать переменную из Blackboard, которая будет использоваться нодой. В данном примере это переменная Player Ref (рисунок 7.22).

Чтобы установить значение в переменную Blackboard, используется нода **Set Value as *тип переменной***. Используйте ноду **Make Literal Name**, чтобы

указать имя ключа переменной из Blackboard (рисунок 7.23). Данные ноды можно писать, как в обычном блюпринте, так и в Controller-блюпринте.

При запуске проекта агент будет следовать за игроком и при достижении цели ждать 5 с, затем повторять дерево. Вы можете проследить выполнение дерева, открыв его, при запущенной игре (рисунок 7.24) [1, 2].

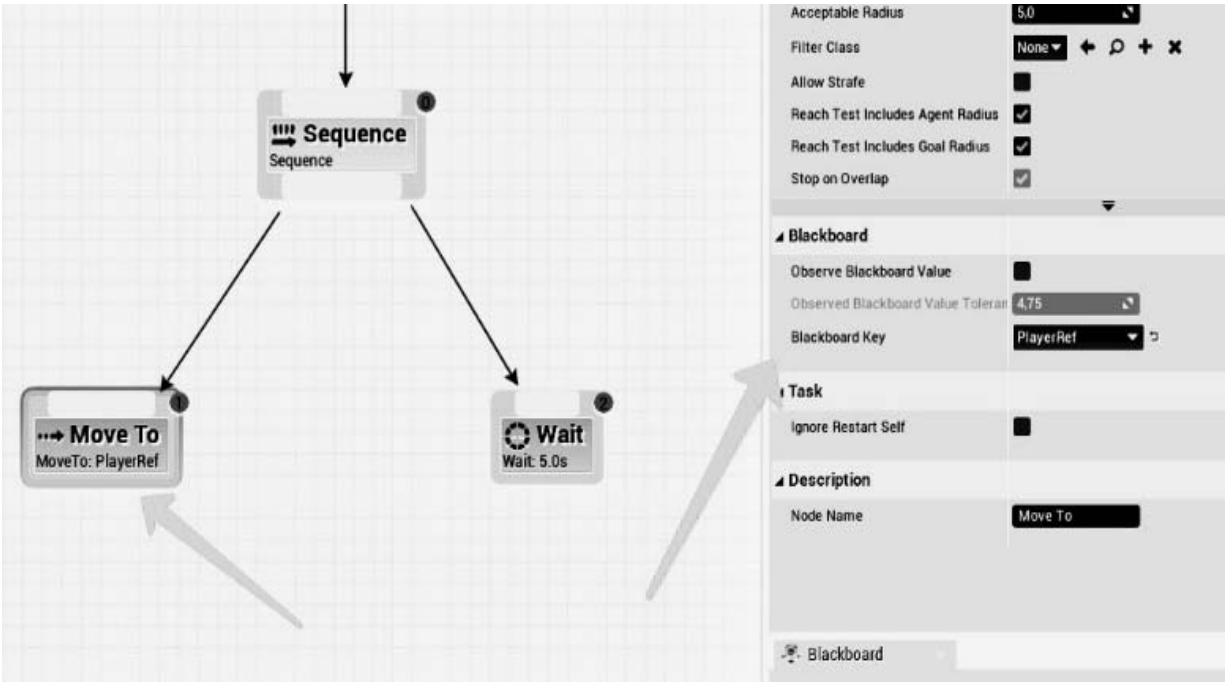

Рисунок 7.22 – Настройка ноды Move To

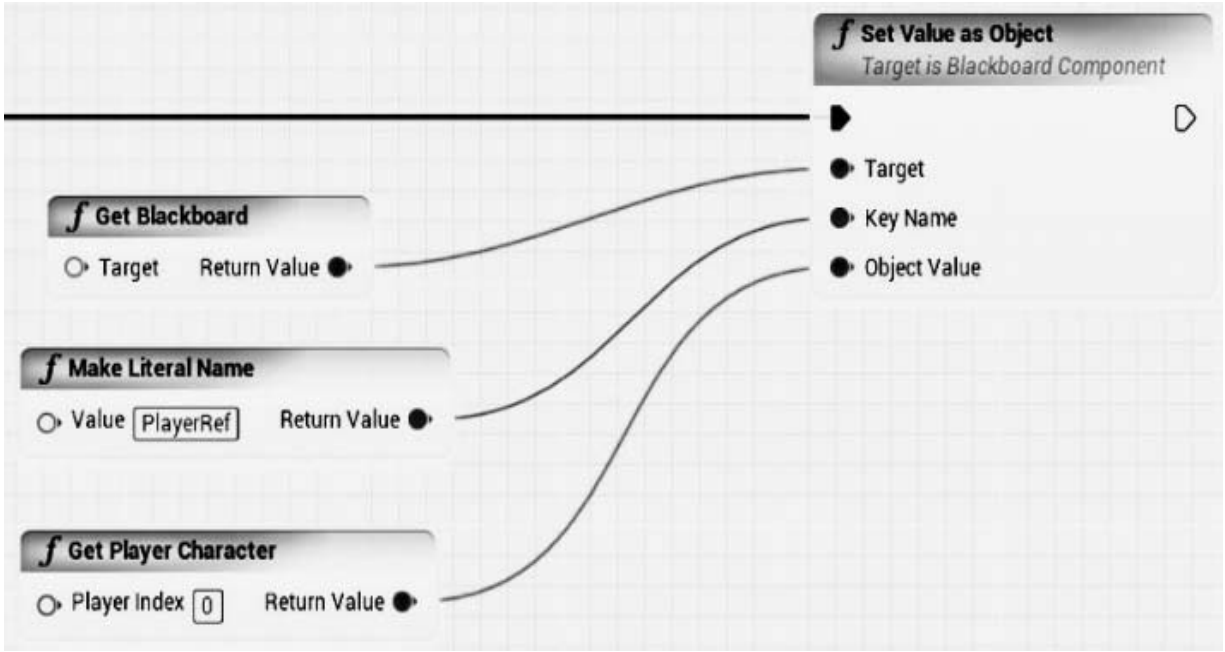

Рисунок 7.23 – Установка значения в переменную Blackboard

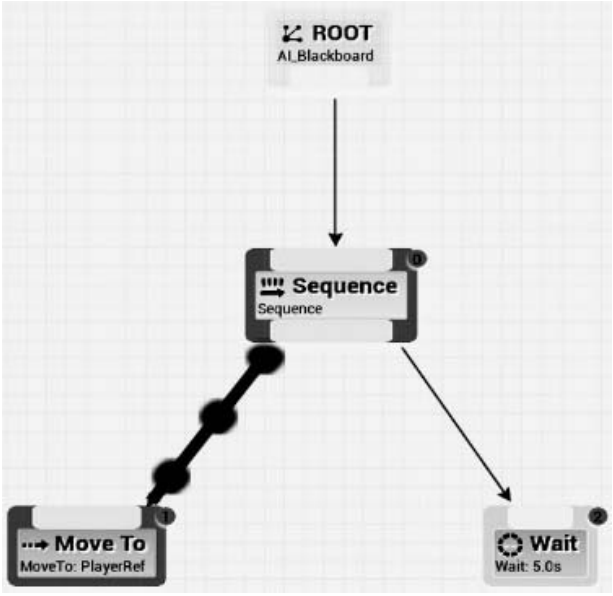

Рисунок 7.24 – Изображение ноды дерева поведения, которая выполняется в настоящий момент времени

Список литературы

1 **Куксон, А.** Разработка игр на Unreal Engine 4 за 24 часа / А. Куксон, Р. Даулингсока, К. Крамплер. – Москва: Эксмо, 2019. – 528 с.: ил.

2 **Загарских, А. С.** Введение в разработку компьютерных игр / А. С. Загарских, А. А. Хорошавин, Э. Э. Александров. – Санкт-Петербург: ИТМО, 2019. – 79 с.