УНИВЕРСИТЕТ ИТМО

Загарских А.С., Хорошавин А.А.

Александров Э.Э.

ВВЕДЕНИЕ В РАЗРАБОТКУ КОМПЬЮТЕРНЫХ ИГР

Санкт-Петербург
2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

Загарских А.С., Хорошавин А.А.,

Александров Э.Э.

ВВЕДЕНИЕ В РАЗРАБОТКУ КОМПЬЮТЕРНЫХ ИГР

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 09.04.03 Прикладная информатика в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры

Санкт-Петербург

2020

Загарских А.С., Хорошавин А.А., Александров Э.Э., Введение в разработку компьютерных игр– СПб: Университет ИТМО, 2019. – 79 с.

Рецензенты: Карсаков А.С. к.т.н. Старший научный сотрудник национального центра когнитивных разработок, доцент факультета цифровых трансформаций

В пособии приводятся краткие теоретические сведения и изложения содержания практических занятий, методические указания и требования к курсовому проекту по дисциплине “Основы разработки компьютерных игр” направления подготовки магистратуры 09.04.03 “Прикладная информатика” по профилю “Технологии разработки компьютерных игр”.
Учебное пособие фокусируется на разработке компьютерных игр в среде Unreal Engine 4. На момент написания данного пособия актуальная версия Unreal Engine 4.24.

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

© Университет ИТМО, 2020

© Загарских А.С., Хорошавин А.А., Александров Э.Э., 2020

Содержание

Введение **4**
Занятие 1 Основные инструменты и ключевые элементы Unreal Engine 4 ..... **6**
Занятие 2 Программирование на Blueprints. Часть I **19**
Занятие 3 Программирование на Blueprints. Часть II **30**
Занятие 4 Работа с материалами. Material Editor **42**
Занятие 5 Работа с контентом в UE 4 II. Animation Blueprint **49**
Занятие 6 Программирование на Blueprints. Часть III **55**
Занятие 7 Искусственный интеллект. Инструменты написания AI в UE4: State Machine, Behaviour Tree, Navigation **61**

Введение

Данное учебное пособие предназначено для слушателей курса “Основы разработки компьютерных игр” и фокусируется на разработке компьютерных игр в среде (игровом движке) Unreal Engine 4. Курс не покрывает непосредственно особенности программирования на C++ в данной среде, он направлен на систему визуального программирования Blueprints и посвящен основным инструментам Unreal Engine 4.

Пособие может быть использовано как для самостоятельного изучения разработки компьютерных игр в среде Unreal Engine 4, так и служить практическим руководством в процессе изучения курса, так как содержит ключевые моменты лекционных и практических занятий.

Авторы данного пособия рекомендуют использовать его следующим образом.

Если вы студент, последовательно следуйте указаниям каждого занятия, параллельно ищите дополнительную информацию в интернете и смотрите в официальную документацию движка по ссылке https://docs.unrealengine.com/. Если вы пропустили занятие, используйте соответствующую тему, чтобы покрыть пробелы в своих знаниях. Если вы просто стремитесь быстрее узнать все самые важные инструменты движка, то постарайтесь прочитать и проработать данное пособие за пару дней в формате интенсива. Таким образом, есть вероятность преодолеть входной порог сложности Unreal Engine 4 и отправиться в вольное плавание по поиску дополнительной информации.

Если вы преподаватель, обратите внимание, что занятия данного пособия покрывают все необходимые знания для работы с основными инструментами Unreal Engine 4. Все уроки расположены в том порядке, в котором их полезнее всего преподносить студентам. Не обязательно использовать данное пособие как прямое руководство к действию – вы можете разнообразить лекции и практики своими собственными примерами.

Для выполнения практических заданий необходимо скачать движок Unreal Engine 4 с официального сайта: https://www.unrealengine.com/en-US/. На момент написания данного пособия актуальна версия Unreal Engine 4.24. Движок является бесплатным программным обеспечением с несколькими видами лицензий. Оплата будет необходима лишь в том случае, если ваш проект подразумевает монетизацию.

Практические задания пособия содержат рекомендованную последовательность действий для работы в среде Unreal Engine 4 при разработке компьютерной игры. В результате успешного выполнения всех заданий обучающийся получит материал, необходимый для создания ключевого проекта курса, представляющего собой компьютерную игру. Необходимым условием защиты проекта является наличие написанных пользовательских нестандартных блюпринт-классов.

Словарь сленговых терминов и сокращений

- Игровой движок – комплекс программных средств, использующийся для разработки интерактивных программ.
- Темплейт – предустановленный образец, шаблон.
- Ассет – цифровой объект, преимущественно состоящий из однотипных данных.
- ПКМ – правая кнопка мышки.
- ЛКМ – левая кнопка мышки.
- Блюпринт – программный класс, реализованный в правилах языка визуального программирования Blueprints.
- Плюсовый проект – проект, реализованный на языке программирования C++.
- Уровень – виртуальное пространство, сохраненное в файле.
- Биллборд – цифровое изображение, которое всегда поворачивается в сторону виртуальной камеры.
- Инстанс класса – единичный экземпляр программного класса.
- Нода – небольшая панель, содержащая данные или программные операции.
- Ивент – название вида программных функций.
- Таймлайн – тип ноды, позволяющий задать зависимость изменения значения переменной от времени.
- Пин ноды – элемент ноды, через который происходит соединение нескольких нод.
- Стейт-машина – то же, что и конечный автомат.
- Меш – это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике.

Занятие #1 Основные инструменты и ключевые элементы Unreal Engine 4.

1.1 Результат занятия

- уметь создавать проект в Unreal Engine 4;
- уметь ориентироваться в среде Unreal Engine 4;
- уметь создавать новый проект;
- уметь пользоваться панелью Viewport;
- уметь пользоваться панелью World Outliner;
- уметь пользоваться панелью Details;
- уметь пользоваться панелью Content Browser;
- уметь создавать классы Blueprint, добавлять компоненты к классам, применять изменения в экземплярах класса на блюпринт класс;
- уметь сохранять уровни в Unreal Engine 4.

1.2 Теоретические сведения

При нажатии на ярлык Unreal Engine 4 откроется меню с предложением создать проект определенного типа. Выберите **Games** и нажмите кнопку **Next** (рисунок 1).

P.S. Тип проекта определяет предустановленные настройки проекта.

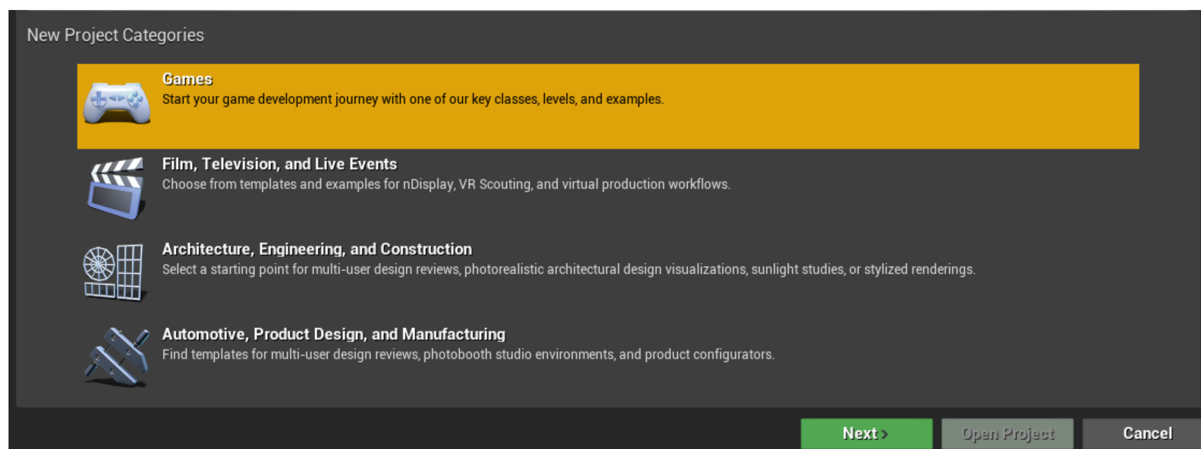

Рисунок 1. Меню выбора типа проекта в Unreal Engine 4.

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 2).

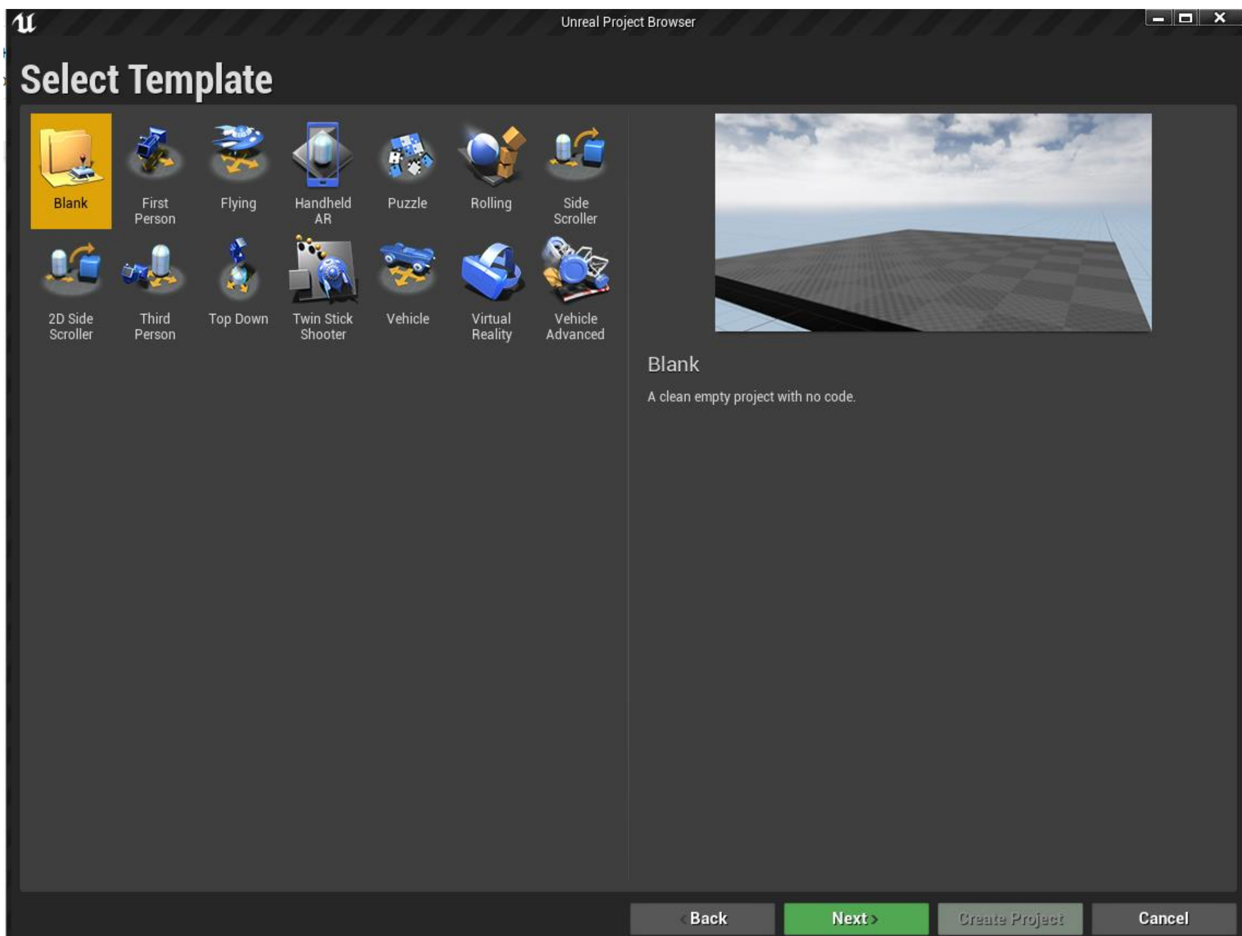

Рисунок 2. Меню выбора темплейта(шаблона) проекта в Unreal Engine 4.

Последним шагом является выбор настроек проекта. Самой важной настройкой является выбор блюпринт- или С++ проекта. Все настройки вы сможете изменить в ходе разработке игры, но с данной опцией лучше определиться с самого начала, так как трансформация блюпринт-проекта в плюсовый и обратно может вызвать ошибки в скриптах и блюпринт-классах (рисунок 3).

Итак, у вас открылся пустой (**Blank)** проект. Рассмотрите основные панели и окна Unreal Engine 4, в которых вы будете проводить большую часть времени работы над игрой. Под **окном** в среде Unreal Engine 4 понимается большая область, в которой располагаются интерактивные элементы среды Unreal Engine 4.

Окно **Viewport** – это окно, через которое вы можете обозревать уровень. В нём вы можете обозревать виртуальное пространство, словно режиссер на съемочной площадке, включая все объекты вашего уровня, в том числе те, что не видны в самой игре – например, вспомогательные иконки (billboard) объектов. Для переключения режима отображения между разработчиком и **Game view** (рисунок 4) нажмите на клавишу **G**.

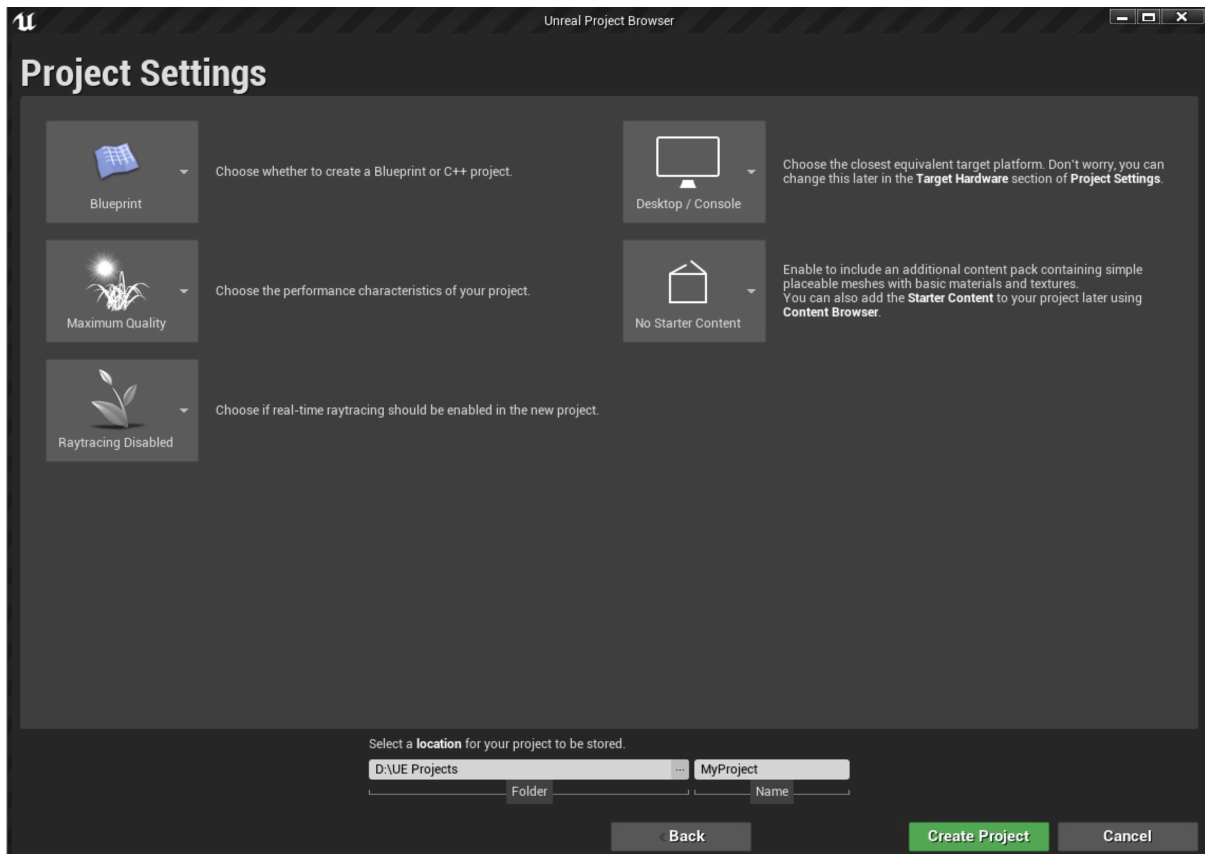

Рисунок 3. Окно выбора настроек проекта. Обратите внимание, что вы можете заранее определить некоторые графические настройки проекта, например функцонал Raytracing.

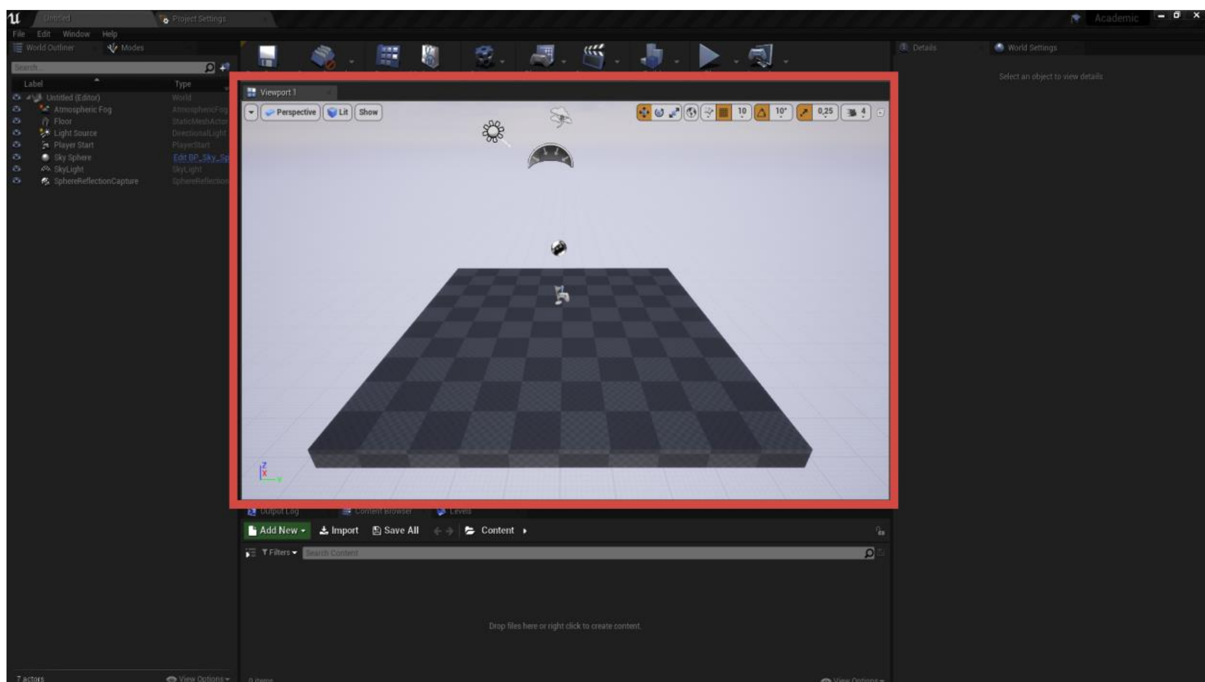

Рисунок 4. Окно Viewport в Unreal Engine 4.

Вы можете осматриваться по сторонам, 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 Unreal Engine 4 будем понимать небольшую область экрана (зачастую вертикальную), в которой располагаются интерактивные элементы среды Unreal Engine 4. 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 5). Обратите внимание, что вы можете использовать все классические сочетания клавиш, характерные для множества программ. Вы можете 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.

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 6). В Панели Details также перечислены компоненты выбранного объекта. Обратите внимание, что при выборе отдельного компонента список свойств может изменяться.

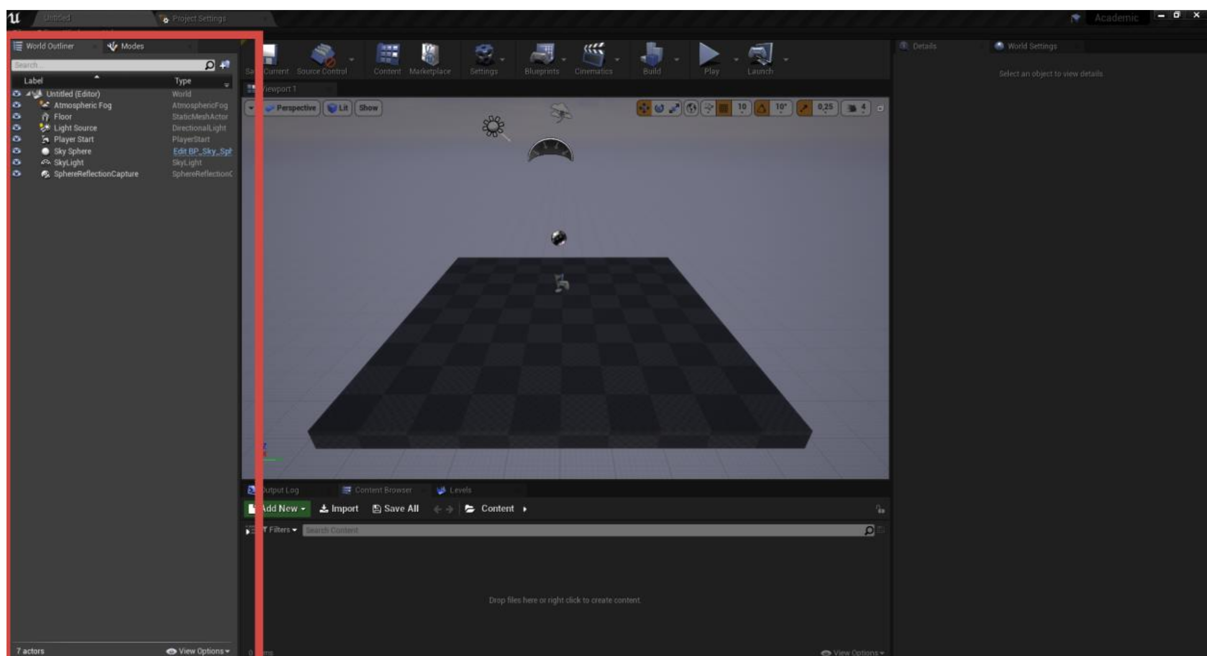

Рисунок 5. Панель World Outliner отображает все объекты текущего уровня. В данной панели доступны опции сортировки и поиска объектов.

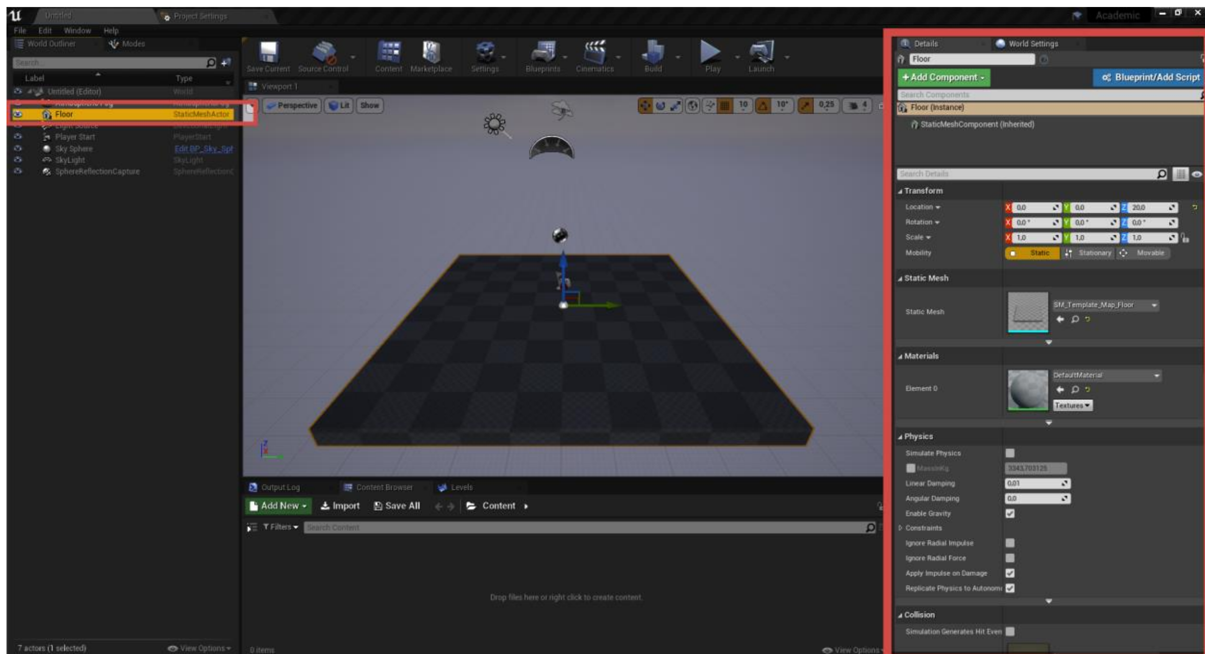

Рисунок 6. Панель Details демонстрирует все свойства выбранного на данный момент объекта.

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 7).

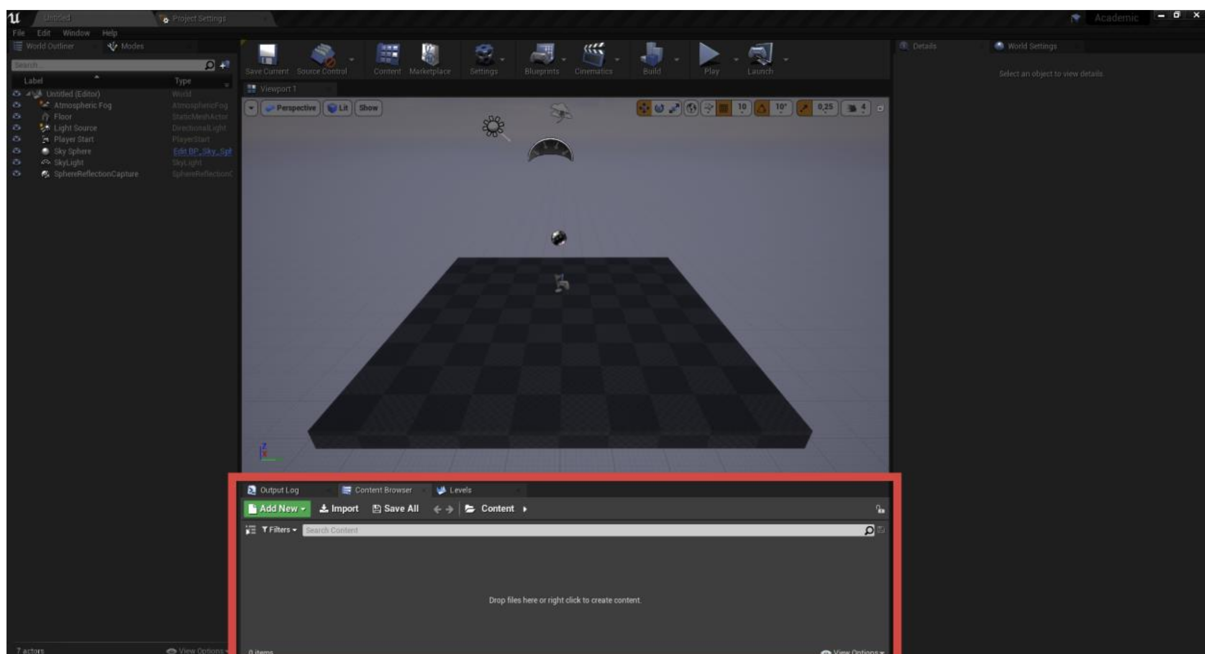

Рисунок 7. Панель Content Browser, показывает список файлов проекта в Unreal Engine 4. В конкретном случае, эта панель пуста.

Blueprints - это система визуального программирования в UE 4. Соединяя

проводами (лапшой) **ноды, эвенты, функции, переменные,** вы можете создать комплексный геймплей.

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 8).

Добавьте объект **Empty Actor** на уровень. Просто перетащите его иконку с вкладки **Basic** панели **Modes** (рисунок 9) в окно **Viewport**. Вы только что поставили пустой объект, он будет выделен сферой биллбордом (не забывайте про клавишу **G**) - данная сфера видна только для разработчика. Иногда объекты, которые могут находиться на уровне, в Unreal Eninge 4 называют акторами или актёрами. В данном пособии будут использоваться слова объект Unreal Engine 4 и актор.

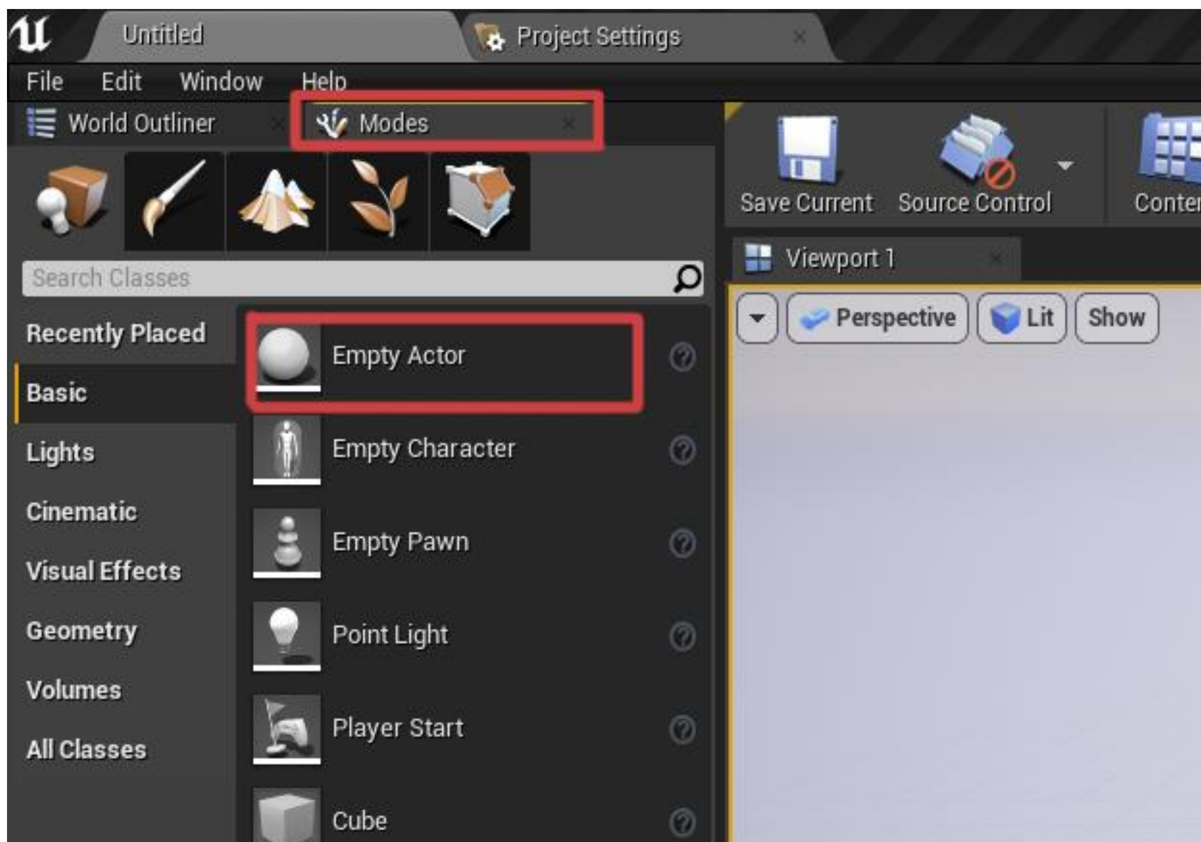

Рисунок 8. Панель Modes, поставьте объект Empty Actor на уровень.

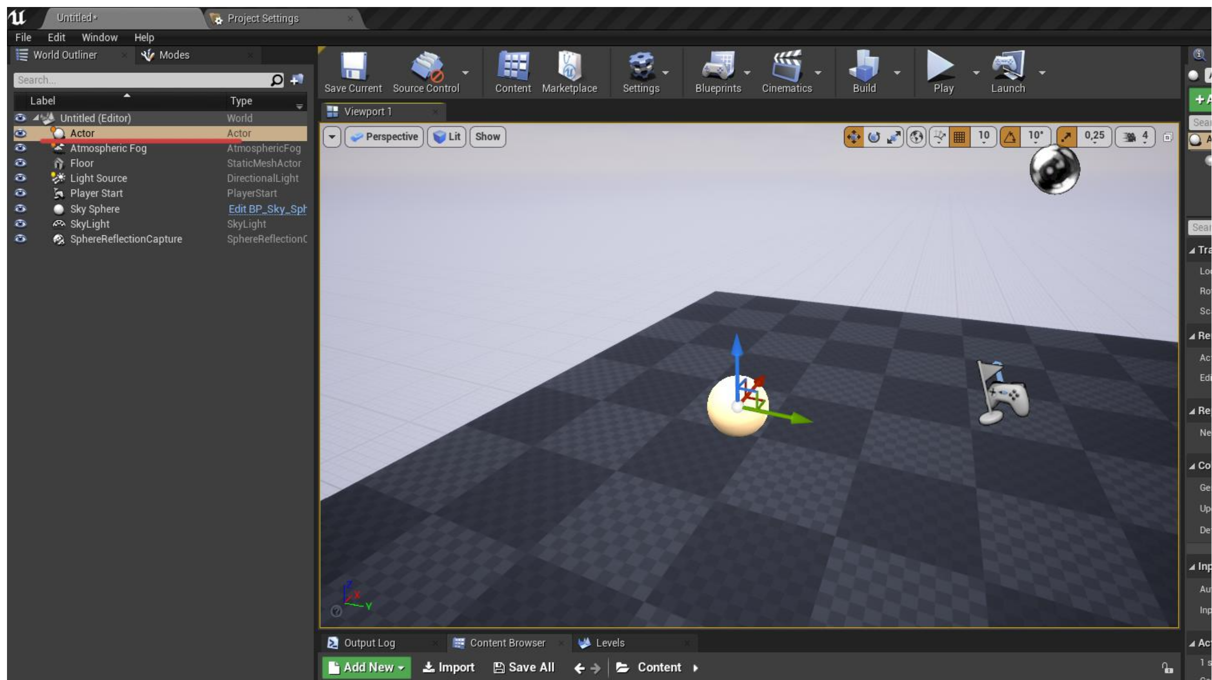

Рисунок 9. Empty Actor на уровне.

Актёры - это главные объекты игры, с которыми можно взаимодействовать, или которыми можно управлять. Это может 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 10).

Рисунок 10 Панель инструментов Transform.

Перейдите в окно компонентов панели **Details,** выбранного **Empty Actor**, нажмите на кнопку **Add Component** - в предложенном списке компонентов выберите **Point Light** (рисунок 11**).** Подробнее про этот и другие компоненты можно почитать в официальной документации Unreal Engine 4.

Чтобы превратить наш объект с компонентом point light в блюпринт-класс, нажмите на синюю кнопку **Blueprint/Add Script** (рисунок 12**).** Выберите папку, в которой будет сохранен ваш блюпринт-класс (рисунок 13).

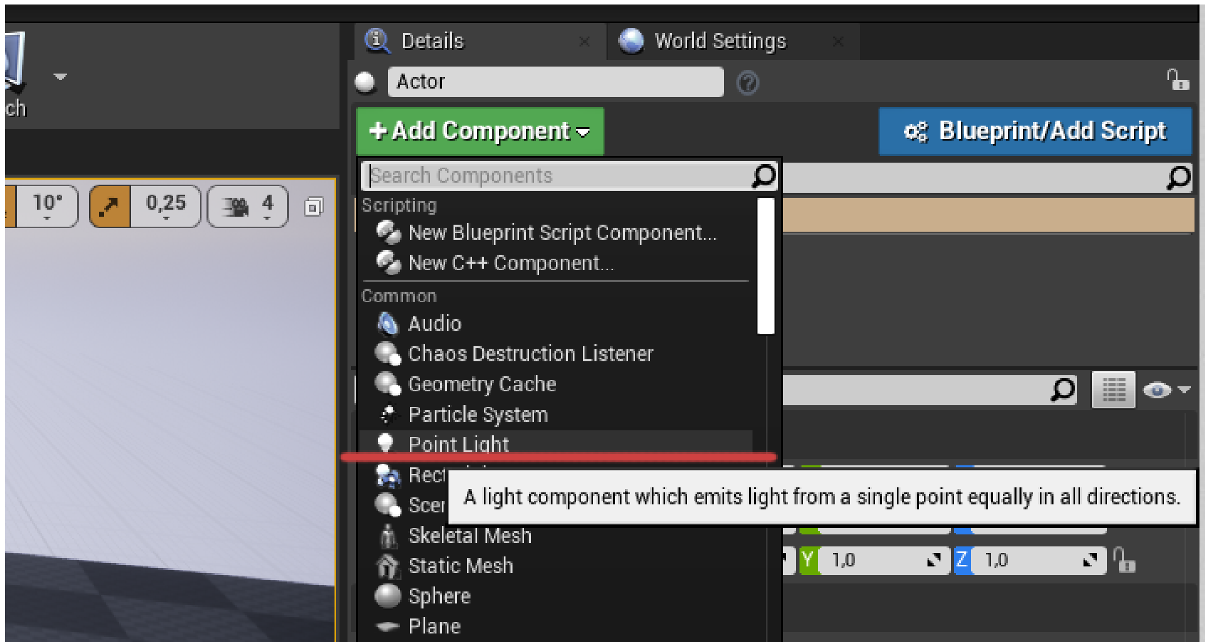

Рисунок 11. Добавление компонента Point Light к объекту Empty Actor.

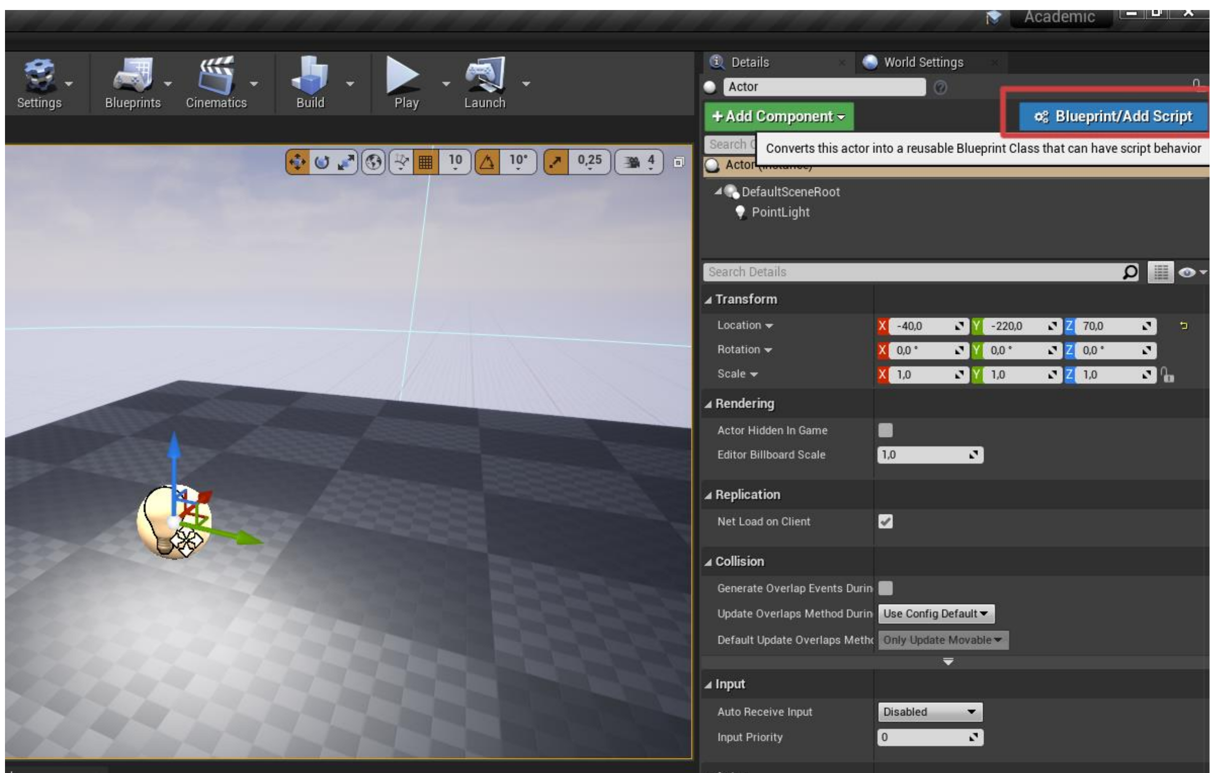

Рисунок 12. При нажатии на синюю кнопку Blueprint/Add Script откроется окно сохранения объекта в виде блюпринт класса.

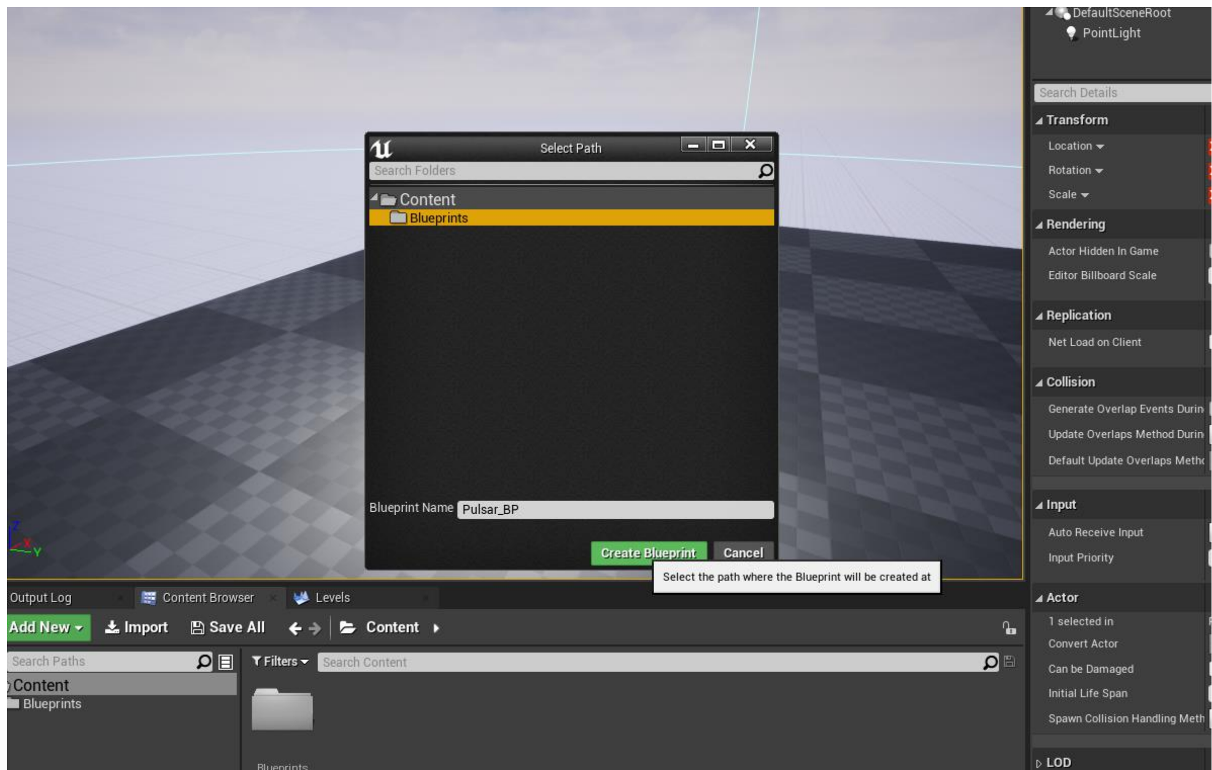

Рисунок 13. Окно выбора пути сохранения блюпринт класса.

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.

P.S. 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 14).

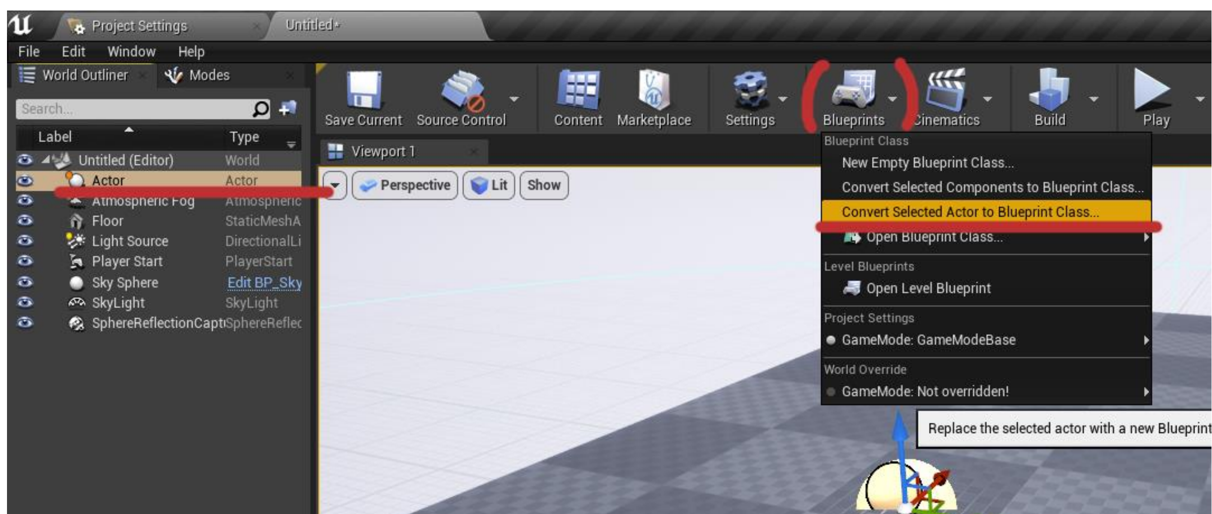

Рисунок 14. Нажав на кнопку Blueprints, вы можете создать блюпринт.

Вы можете перетаскивать блюпринт класс из окна Content Browser на ваш уровень, создавая экземпляры выбранного класса (рисунок 15).

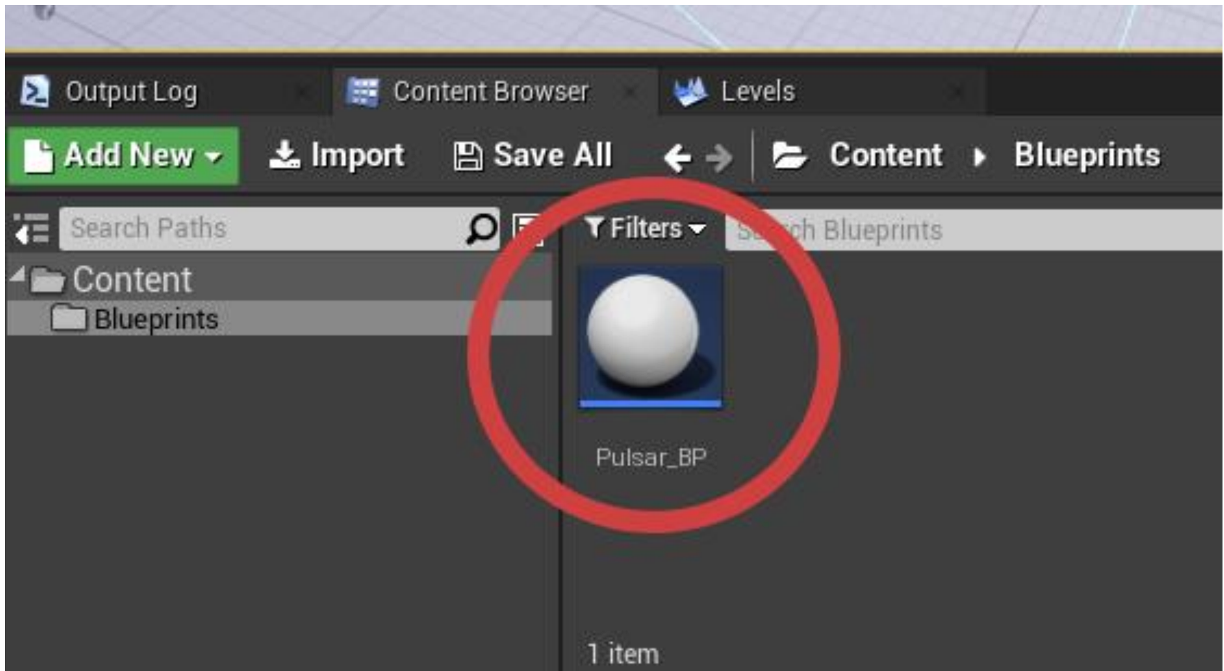

Рисунок 15. Нажмите ПКМ на иконку вашего блюпринт класса и перетащите её в область окна Viewport. Так вы создадите экземпляр вашего класса.

При двойном нажатии на файл блюпринта вы перейдете в отдельное окно - **Blueprint Editor** (рисунок 16).

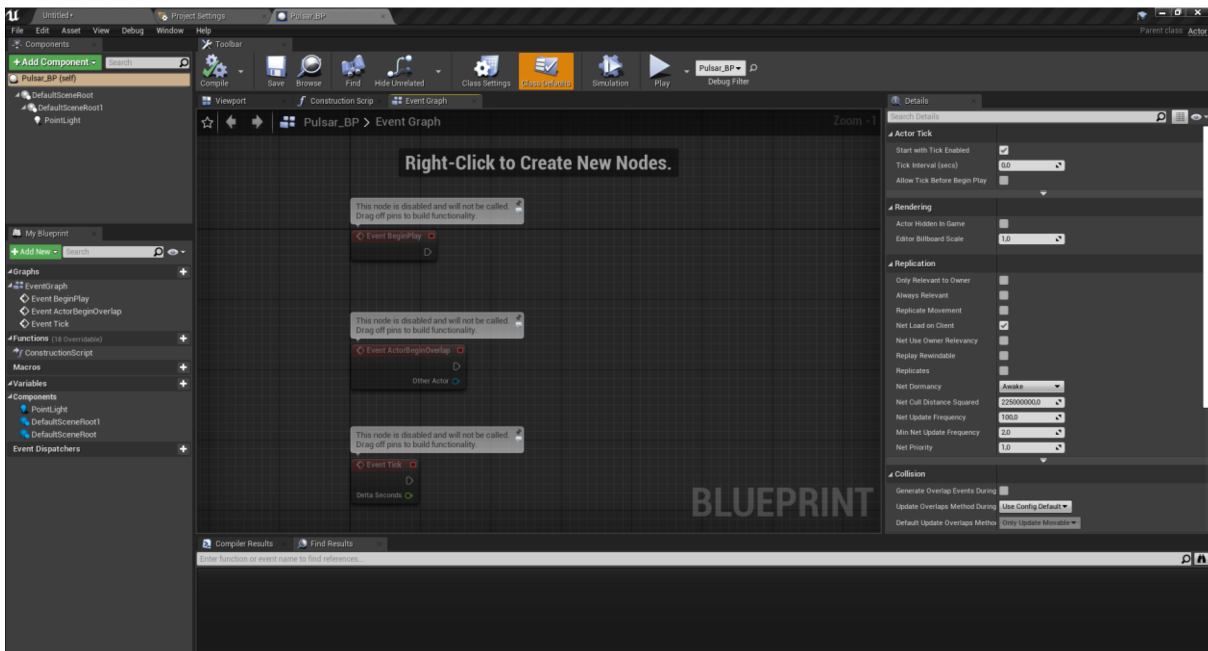

Рисунок 16. Окно Blueprint Editor.

Нажав на вкладку окна viewport, можно посмотреть, как выглядит ваш блюпринт класс (Рисунок 17).

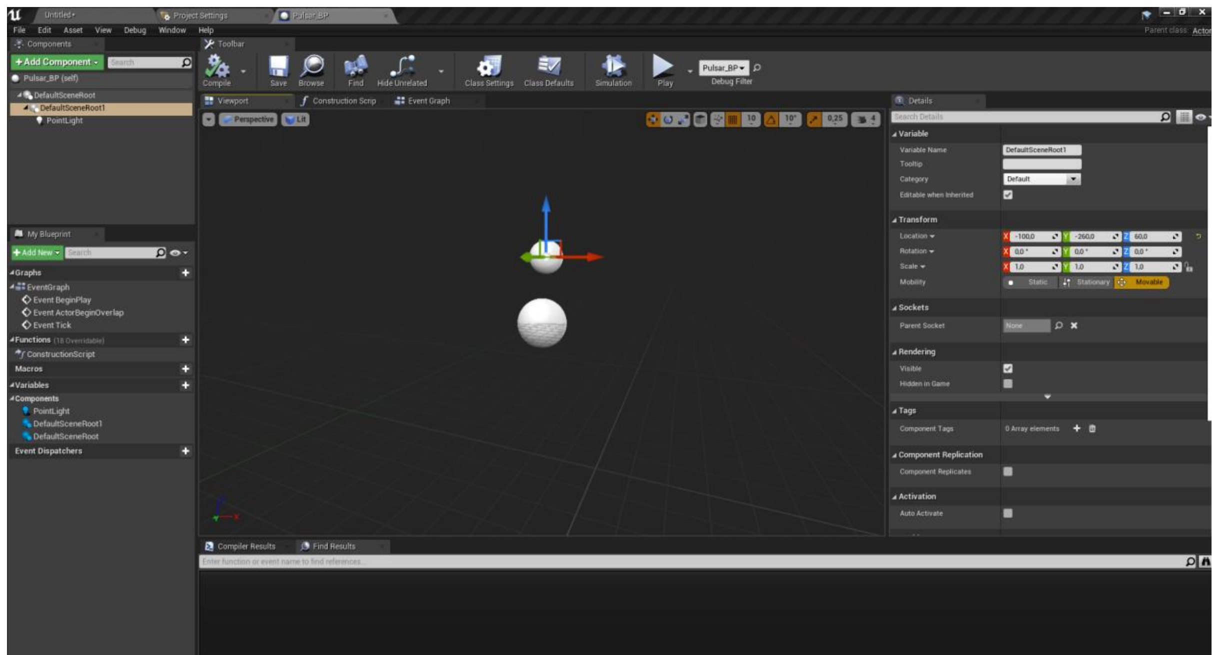

Рисунок 17. Окно Viewport. Важно понимать, что данное окно Viewport является контекстным для выбранного класса и не отображает блюпринт класс на всём уровне.

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 18).

Добавьте компонент Cone (примитив, конус). И поднимите компонент Point Light выше, чтобы было заметно влияние источника света на конус. Скомпилируйте блюпринт (рисунок 19).

P.S. Есть вероятность, что у конуса будет назначен Default Material (выглядит как шахматная сетка) - это нормально на данный момент.

Обратите внимание, что на уровне все экземпляры блюпринтов теперь имеют конус. Таким образом, вы можете 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.*

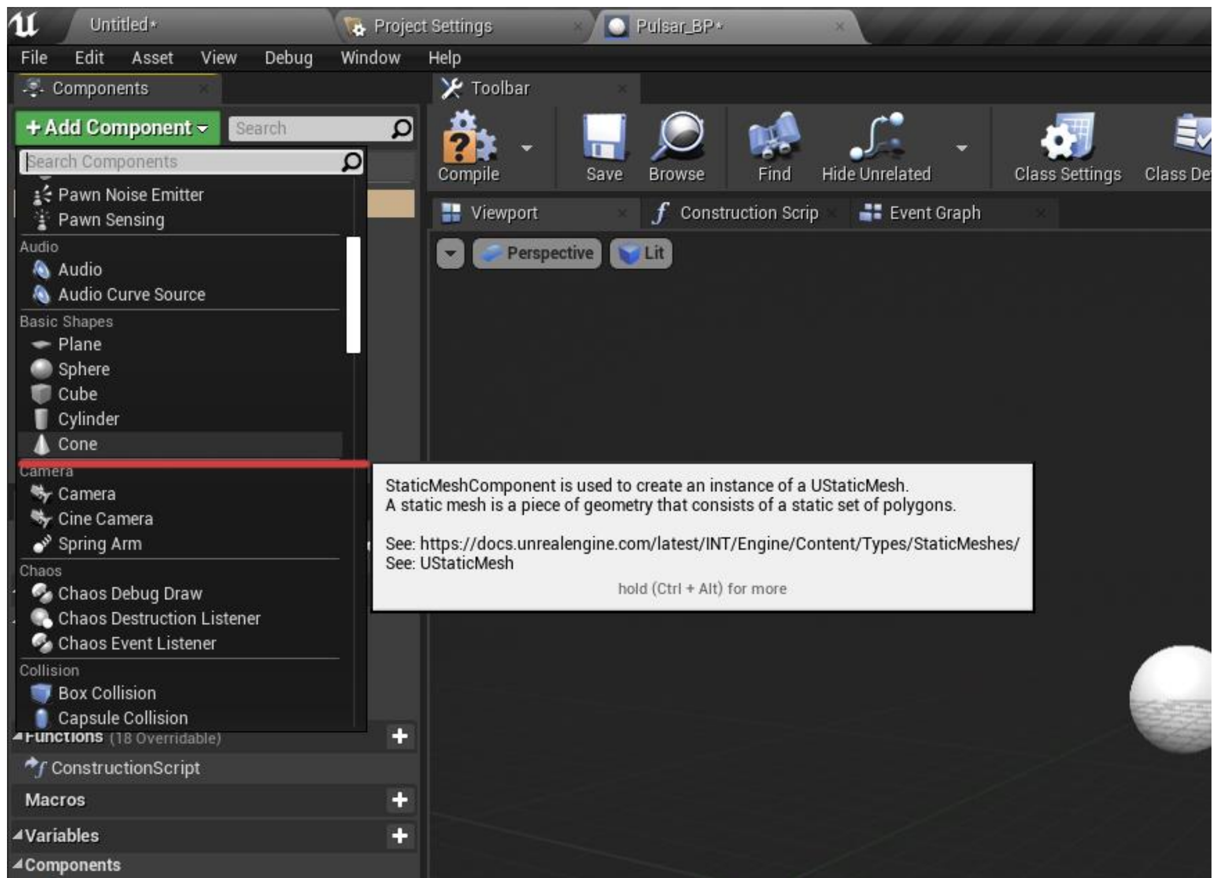

Рисунок 18. Добавьте компонент Cone из панели Components. Обратите внимание, что там же перечислены другие компоненты примитивы: куб, цилиндр, сфера и плоскость.

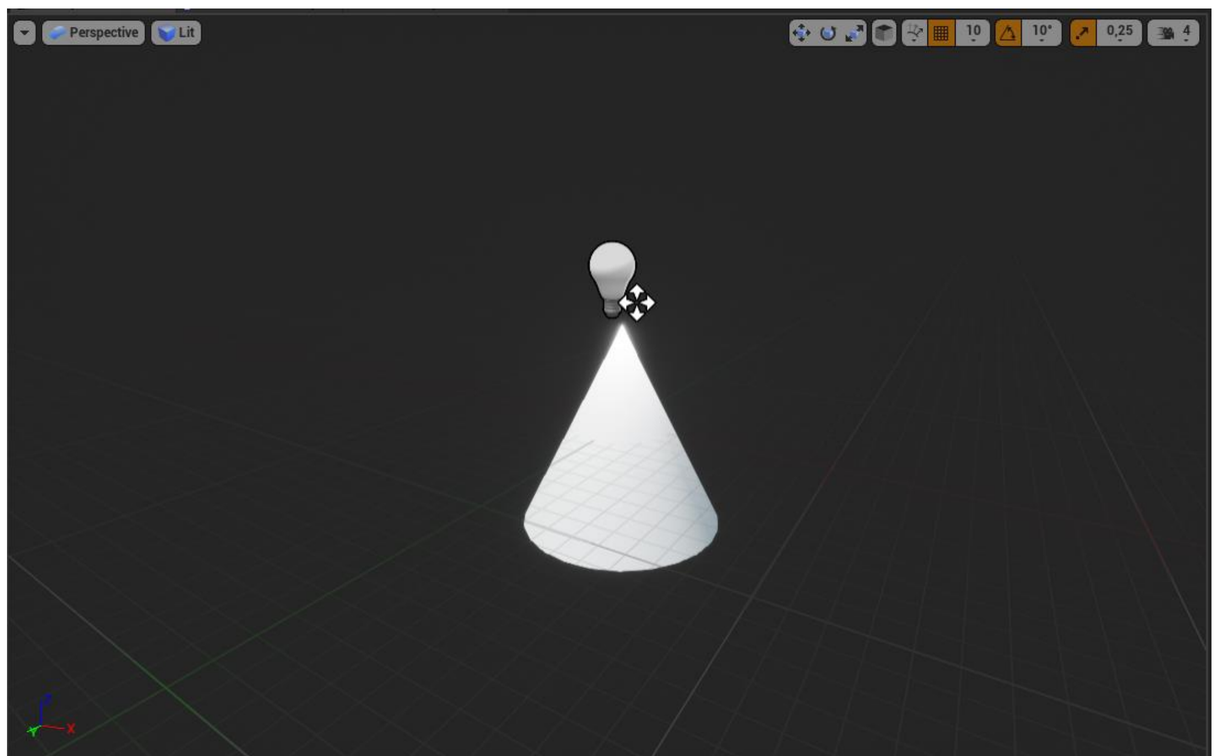

Рисунок 19. Приблизительное изображение окна viewport блюпринт класса Pulsar_BP.

Иногда необходимо посмотреть, как изменения на объекте (экземпляр блюпринта) вписываются в окружение уровня. Если вы хотите, чтобы изменения объекта применились на блюпринт-класс, нажмите кнопку **Edit Blueprint** и выберете **Apply Instance Changes to Blueprint**. Если всё сделано верно, остальные экземпляры (другим словом, инстансы) класса автоматически обновятся с учётом новых изменений (рисунок 20).

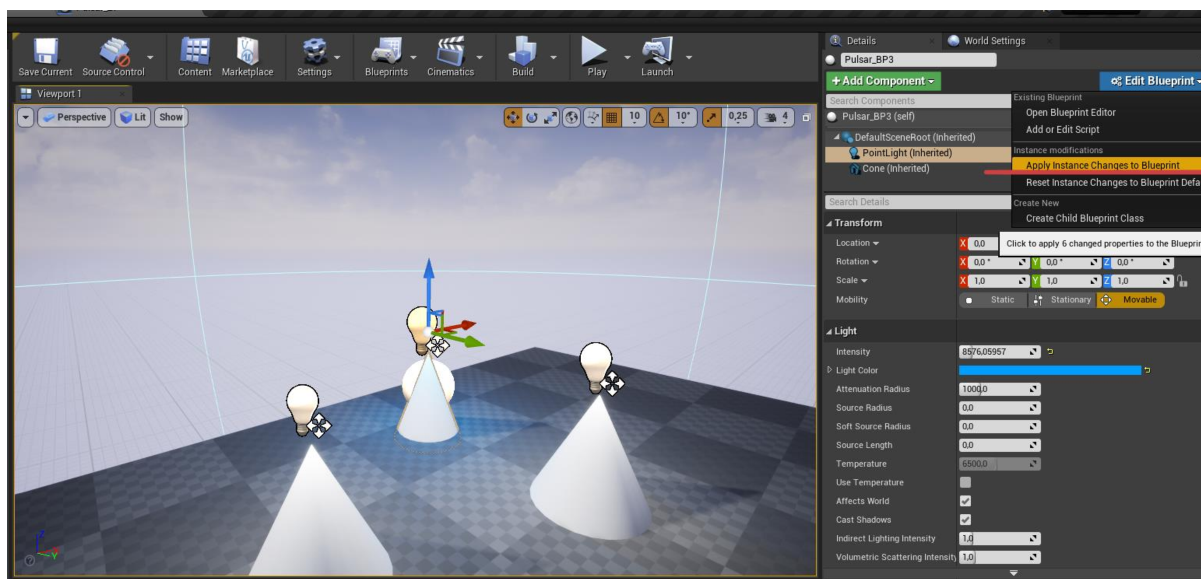

Рисунок 20. После нажатия на синюю кнопку Edit Blueprint выберите пункт Apply Instance Changes to Blueprint.

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 21).

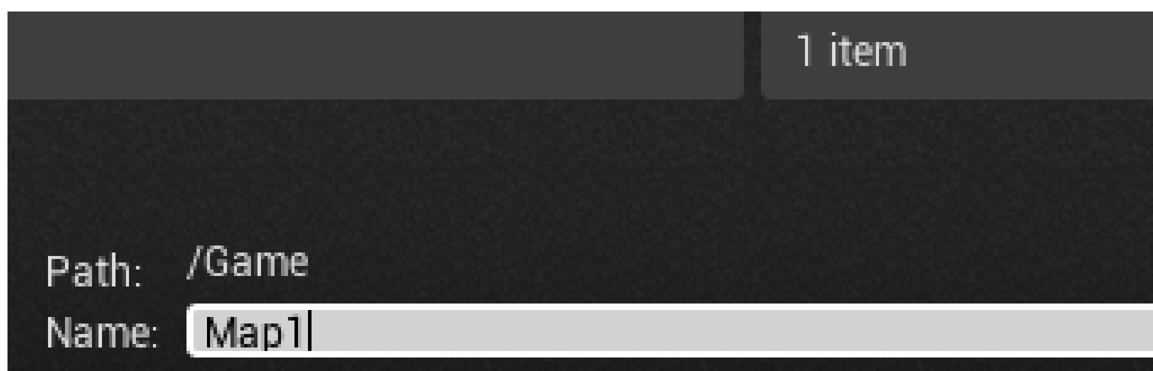

Рисунок 21. Помните, что при названии уровня безопаснее использовать только латинские буквы.

Занятие #2 Программирование на Blueprints. Часть I

2.1Результат занятия

- понимать архитектуру Blueprint проекта UE4;
- понимать порядок выполнения блюпринт-класса: ноды Begin Play и Tick Update;
- понимать Level Blueprint;
- уметь обращаться к компонентам блюпринта и менять их свойства.

2.2 Теоретические сведения

Игра — это не просто контент, из которого она состоит. Игра — это **контент**, **связанный** между собой **логикой** и запрограммированный на определенные действия.

Для этого у нас есть **C++** и система **Blueprints**. Blueprints - это система визуального программирования в UE 4. Соединяя проводами **ноды, эвенты, 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 в прошлом уроке (конус с синим источником света) (рисунок 22).

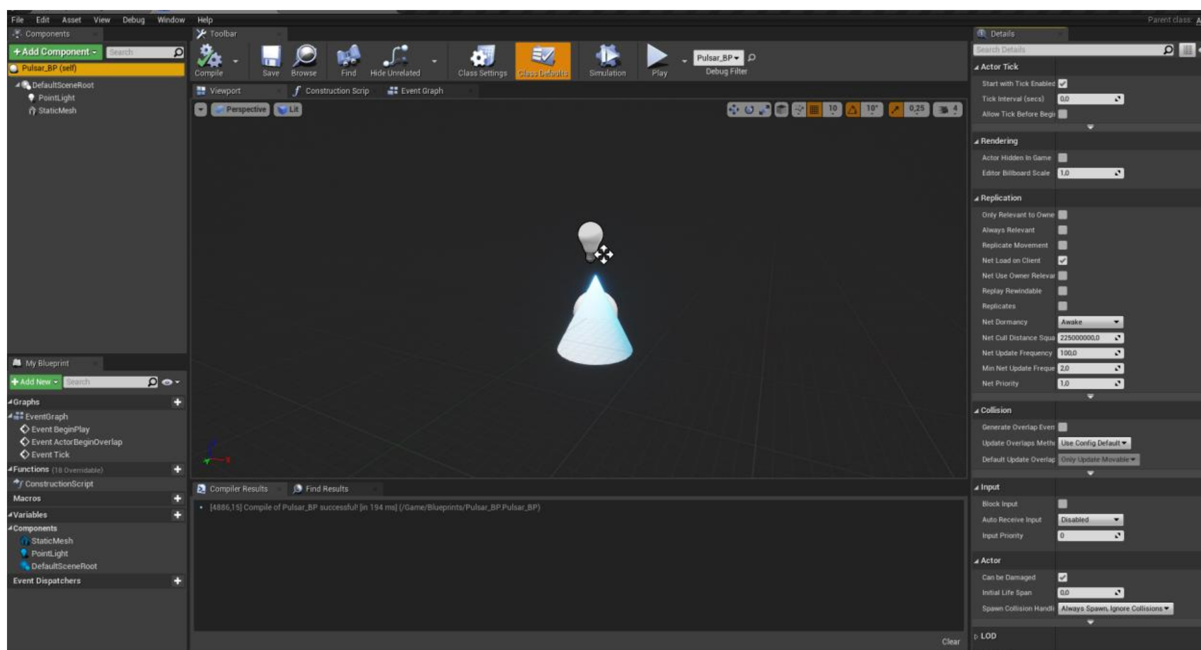

Рисунок 22. Блюпринт-класс Pulsar_BP.

Перейдите в окно **Event Graph** (рисунок 23).

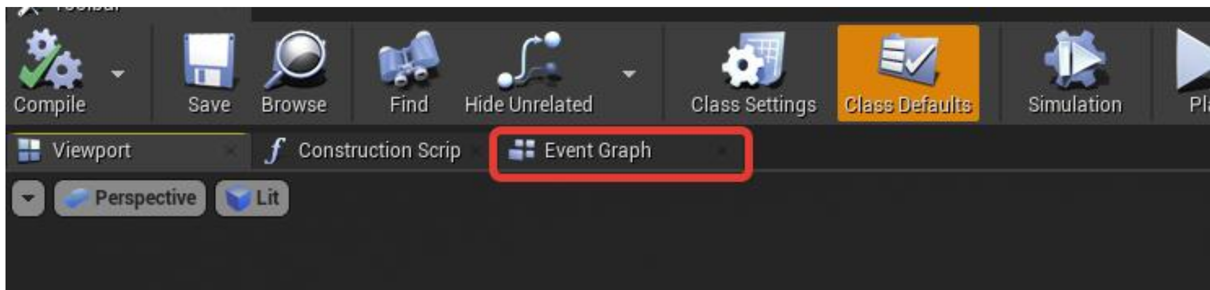

Рисунок 23. Расположение вкладок. Нажав на вкладку Event Graph откроется окно Event Graph, в котором создаётся логика класса.

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 24).

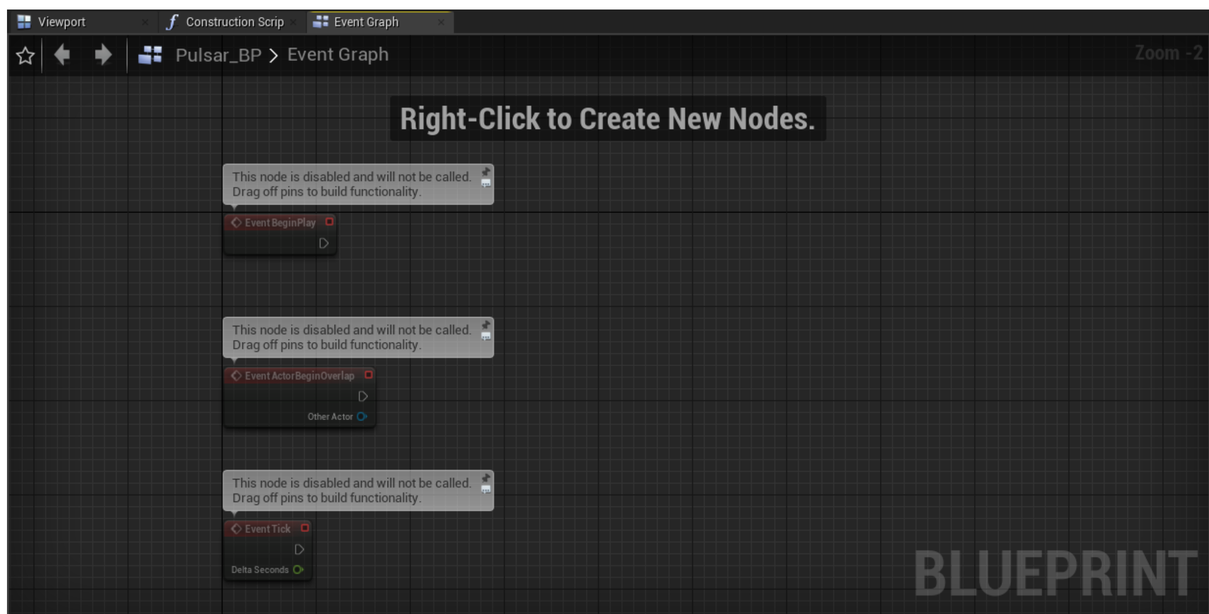

Рисунок 24. Так выглядит окно Event Graph любого ново-созданного блюпринт-класса. Обратите внимание на автоматически созданные ноды BeginPlay Event Tick и Event ActorBeginOverlap.

Нажмите **ПКМ** в свободном месте, чтобы создать свой собственный нод. Вы увидите множество рассортированных по своему функционалу нодов. Со временем вы запомните названия нодов и будете их создавать просто, набирая их название в строке поиска. Выведите текст Hello в встроенную консоль. Для этого наберите в строке поиска **Print** - и выберите из предложенных нод **Print String** (рисунок 25).

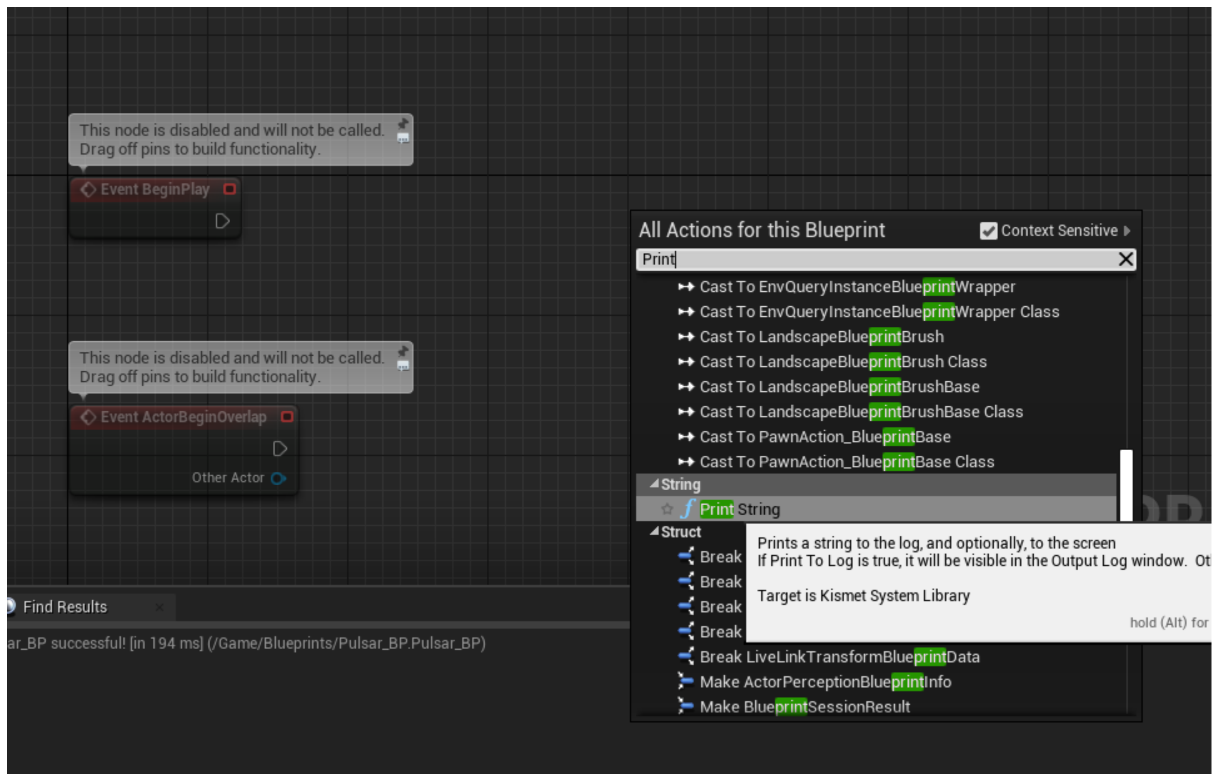

Рисунок 25. Контекстное меню выбора нода. В верху данного меню имеется строка поиска нод. Введите в строку поиска слово print, чтобы найти нод Print String.

Вы увидите, как появился нод Print String. Теперь “вытяните” соединение из треугольного символа нода **BeginPlay** и соедините с входящим пином нода **Print String**. Треугольный символ - это пин выполнения - **execution pin**. Выполнение функции, эвента и кода в целом происходит слева-направо по соединению **execution** (белому проводу). Ивент **BeginPlay**, как следует из его названия, выполняется **один раз при старте игры** (рисунок 26).

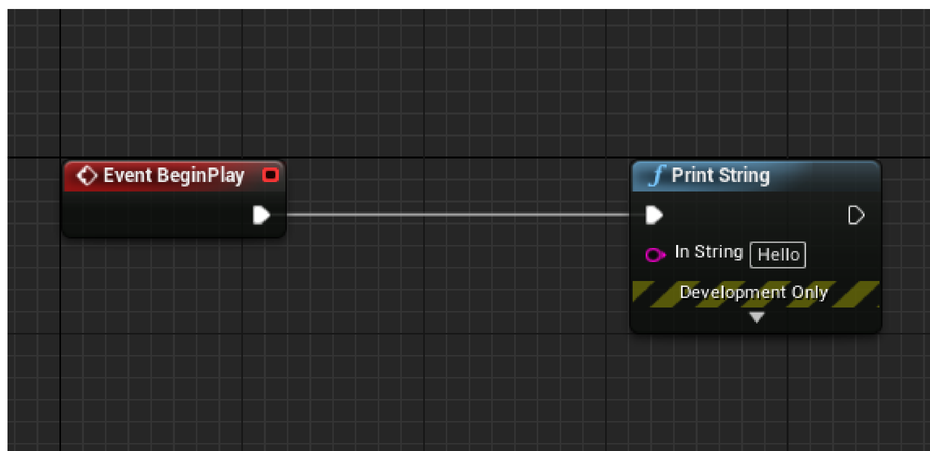

Рисунок 26. Логическая конструкция, выводящая текст Hello во время запуска проекта.

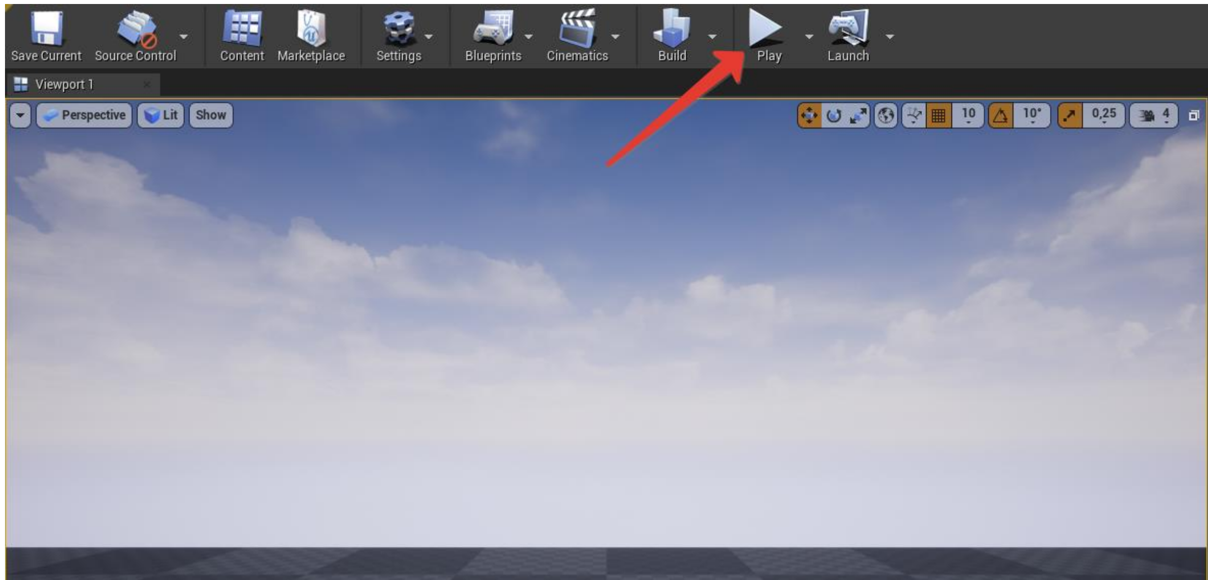

Рисунок 27. Кнопка Play запускает выполнение всех экземпляров блюпринт классов, находящихся на данном уровне.

Подключите нод Print String к ноду Event Tick и запустите проект (рисунок 28).

Рисунок 28. Логическая конструкция выводящая текст Hello без остановки после запуска проекта.

Множественный и непрекращающийся вывод значения аргумента **Print String** подсказывает нам, что нод **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 свой 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 29).

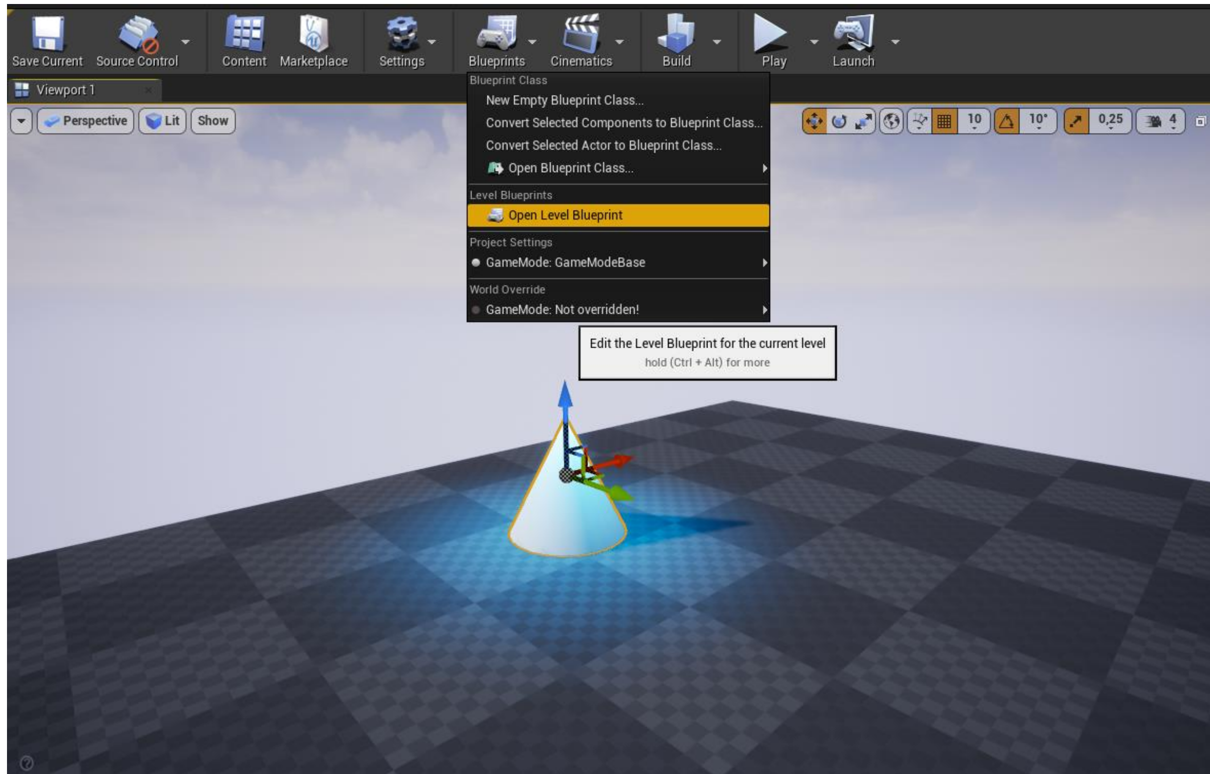

Рисунок 29. Доступ к окно Level Blueprint можно получить нажав на кнопку Blueprints.

Перед вами предстанет обычный **Event Graph** с ивентами **Begin Play** и **Event Tick**. 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 актору на уровне:

1. выберите наш блюпринт Pulsar_BP в World Outliner;
2. перейдите в Event Graph Level Blueprint'а;
3. зажмите клавишу R (reference) и нажмите ЛКМ.

Подробнее про Level Blueprint читайте в официальной документации.

В общей архитектуре кода Unreal Engine классы Game State и Game Mode играют важную роль, однако в основном для онлайн-игр. Для однопользовательских 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. Давайте изменим параметры компонента Point Light через код. Перетащите компонент **Point Light** в поле **Event Graph**. (рисунок 30).

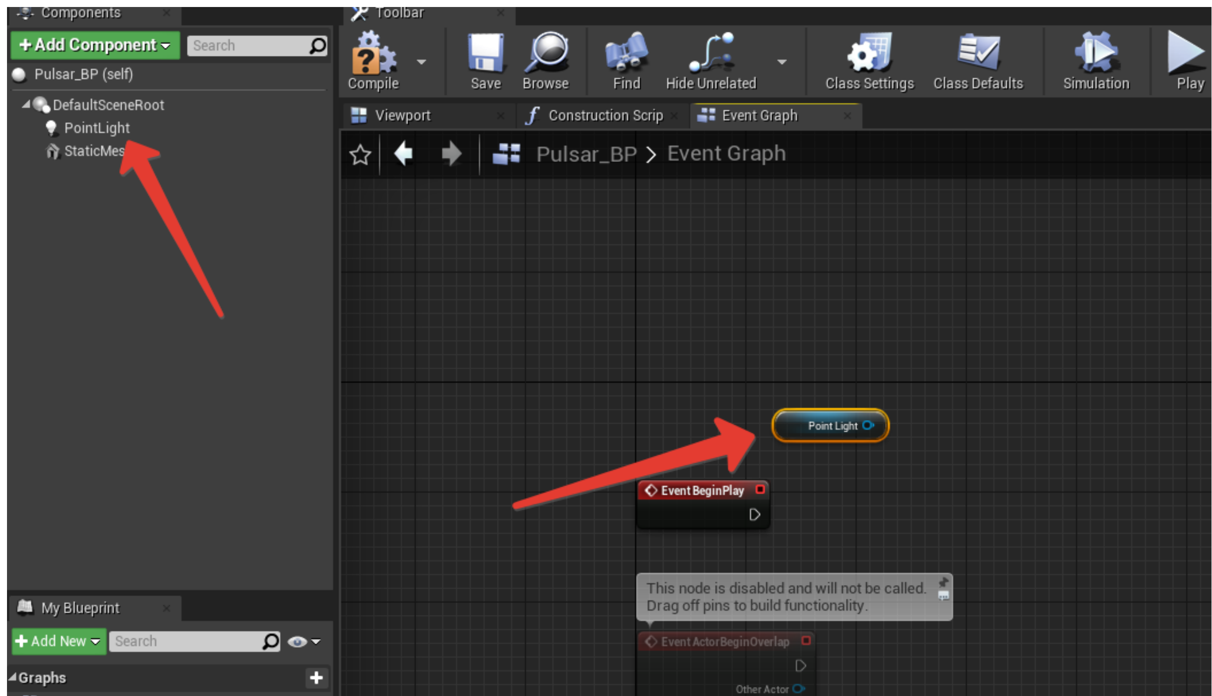

Рисунок 30. Перенесенный компонент из панели Components в окне Event Graph будет выглядеть как небольшой нод с одним выходным пином.

Вытяните пин Point Light и наберите в поиске **Set Light Color** (рисунок 31). Как и в случае с поиском нод, поиск названий нужный параметров нарабатывается “с опытом”.

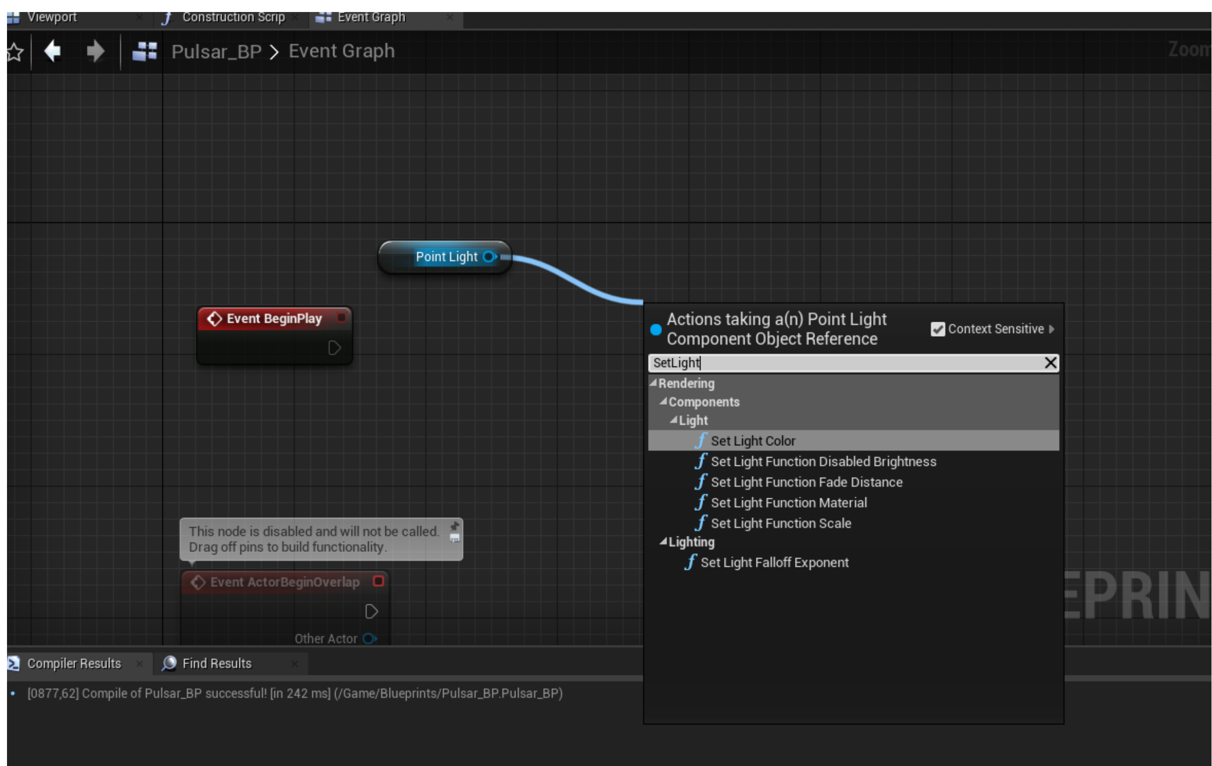

Рисунок 31. Используйте поисковую строку для нахождения параметров компонентов.

Обратите внимание, что, следуя логике здравого смысла, 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 32).

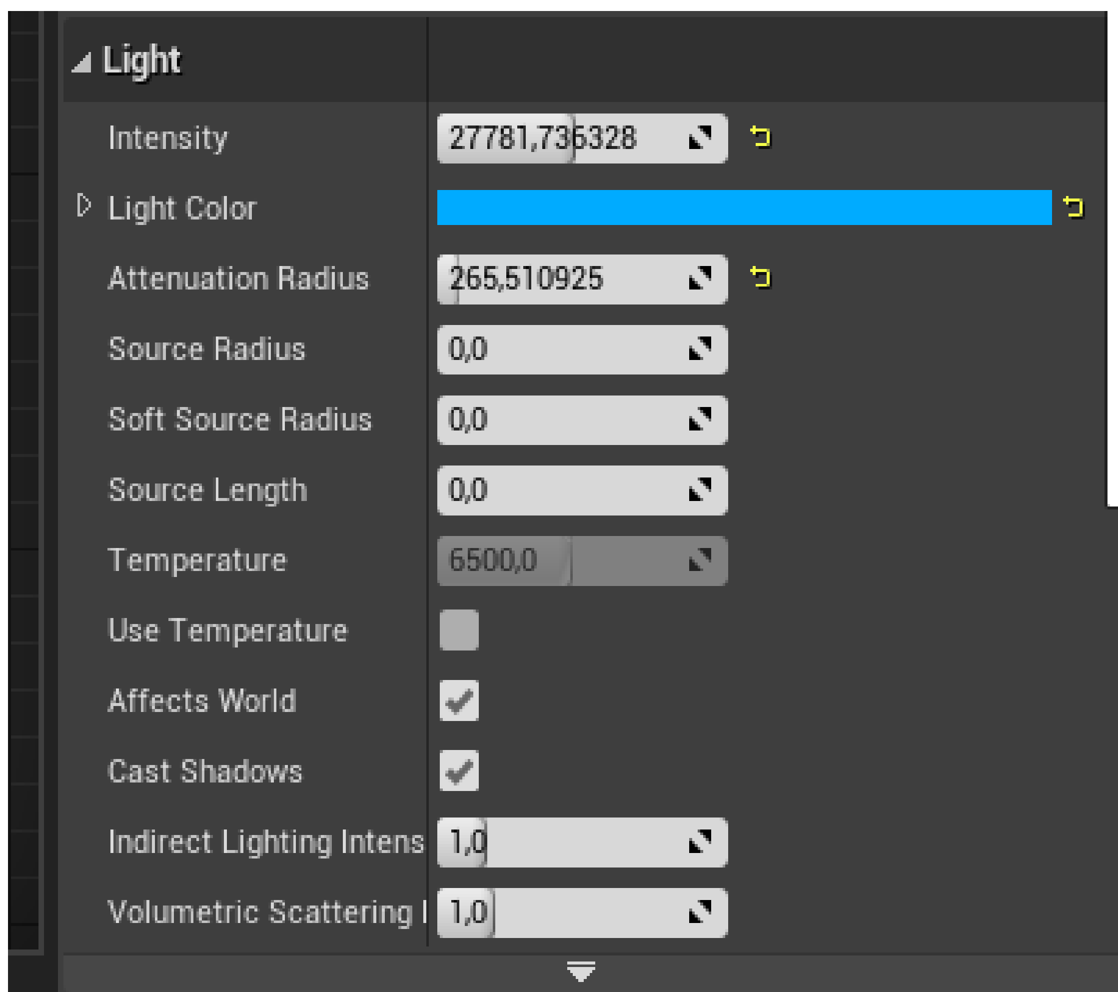

Рисунок 32. Основные параметры (или свойства) компонента Point Light. Вы можете обращаться к ним в окне Event Graph.

Свяжите нод Begin Play и нод SetLight Color, выберите желаемый цвет, нажав на маленький черный квадрат, репрезентирующий черный цвет (по-умолчанию) (рисунок 33,34). Скомпилируйте и запустите проект.

Цвет источника света в Pulsar_BP должен поменяться на выбранный вами.

P.S. 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 рандомных цветов.

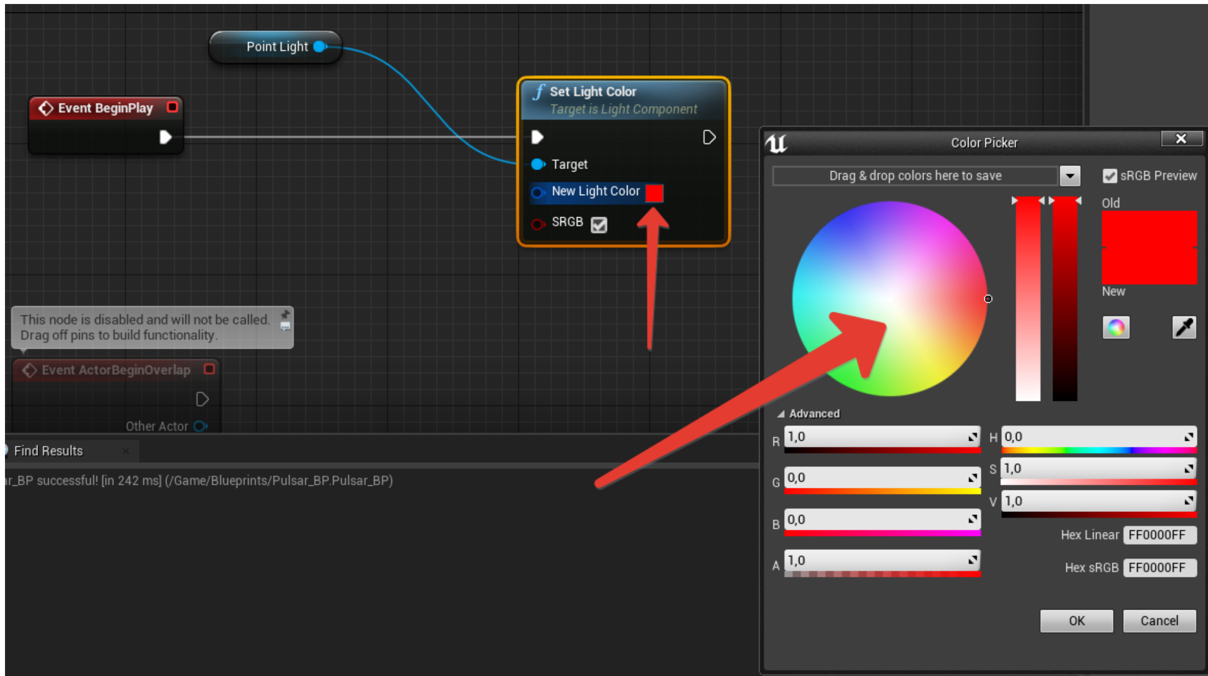

Рисунок 33. Выберите нужный цвет для аргумента New Light Color в палитре.

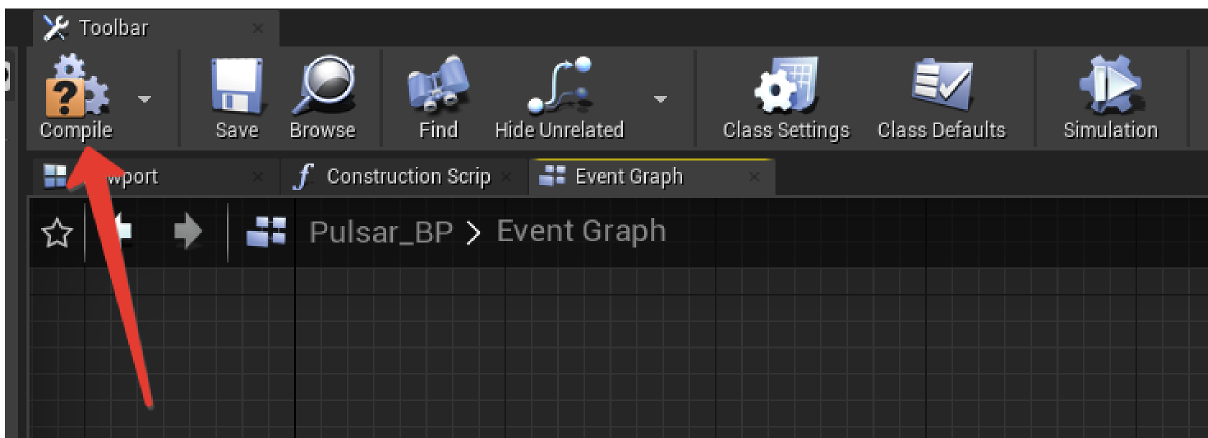

Рисунок 34. Не забывайте нажимать на кнопку Compile. При наличии ошибок иконка Compile изменится.

Теперь сделаем так, чтобы источник света плавно изменял свою интенсивность с 0 до 5000 со старта игры в течение 10 секунд. Для этого можно использовать полезный нод Timeline, скоро вы поймете, что этот нод часто используется в геймплейных и визуальный задачах.

Добавьте нод Timeline, нажав ПКМ и набрав в поиске Add Timeline. Вы можете дать собственной название ноду таймлайна. Обратите внимание, что у нода Timeline имеется несколько входящих пинов для execution “проводов” (Рисунок 35).

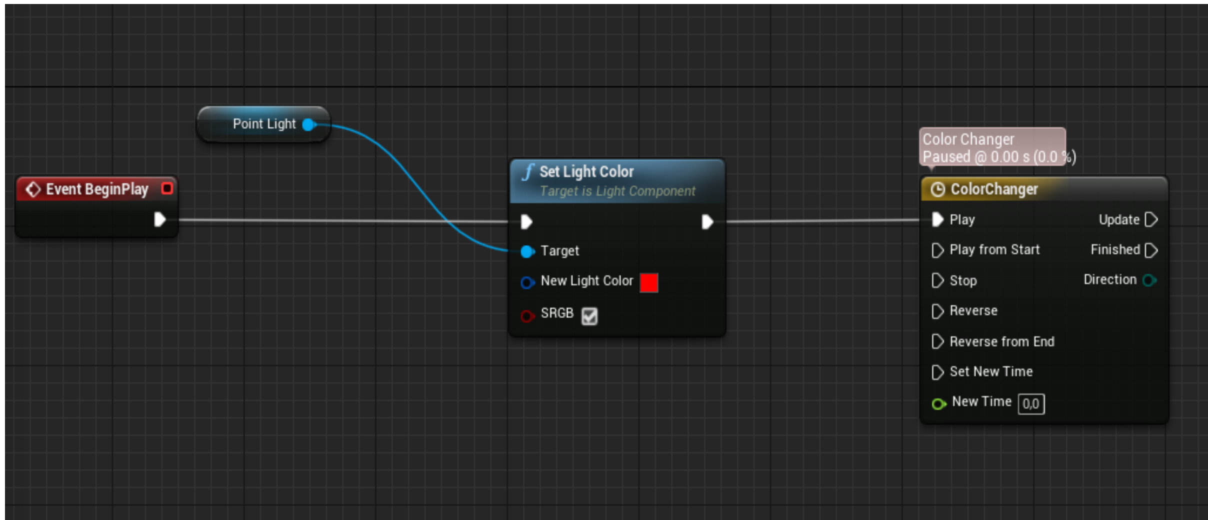

Рисунок 35. Нод Timeline с именем ColorChanger.

Щелкните два раза по ноду 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. Вы можете дать название кривой, для удобства (рисунок 36).

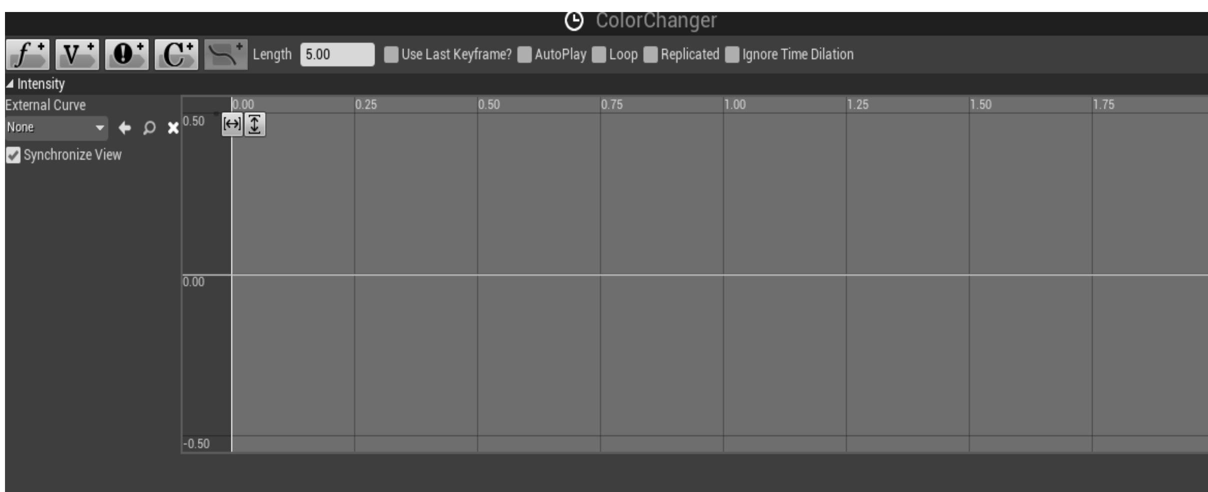

Рисунок 36. Окно редактирования кривых.

Нажмите ПКМ на любом участке кривой и выберете 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 37), чтобы увидеть полноценную картину того как выглядит кривая. Также вы можете указывать точное время и значение точки кривой в соответствующих полях. Обратите внимание, что вы можете изменить продолжительность во времени интерполяции значения по кривой в поле **Length** (по умолчанию 5 секунд).

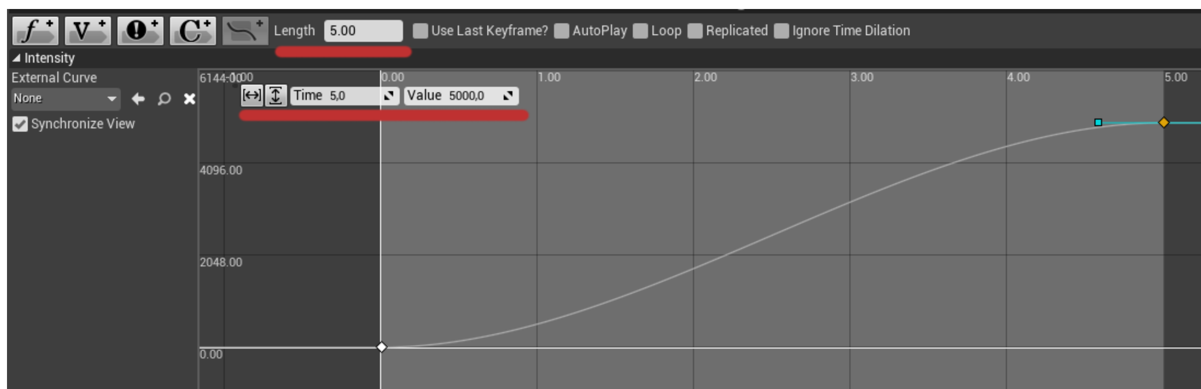

Рисунок 37. Вы можете редактировать продолжительность изменения переменной. Переменная Length. Выбрав конкретную точку на кривой, вы можете изменять её параметры времени и значения.

Вы можете изменить характер интерполяции между ключевыми точками (ключами, keys) кривой, нажав на выбранную точку ПКМ (рисунок 38).

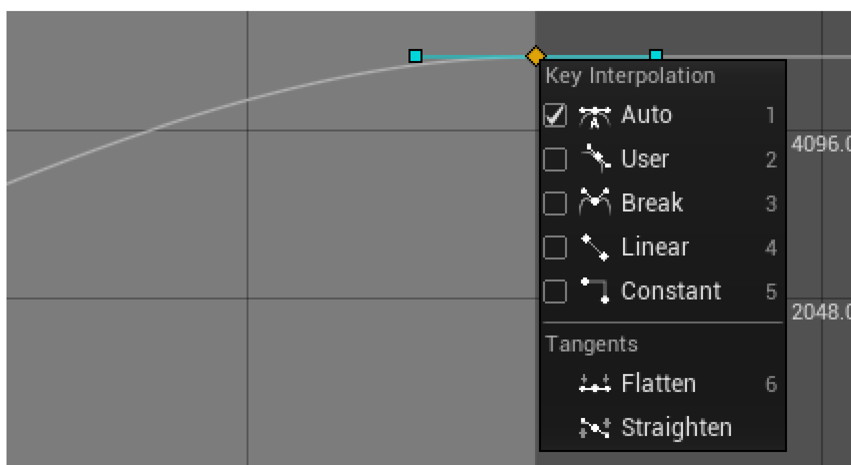

Рисунок 38. Изменение типа интерполяции конкретной точки. Вы можете выбрать несколько точек кривой и изменить интерполяцию для всех них одновременно.

Создайте кривую, продолжительностью в 10 секунд, 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 нодом **Set Intensity** компонента **Point Light**. Соедините выходной аргумент **Intensity** (или название вашей кривой) с аргументом **New Intensity** (рисунок 39).

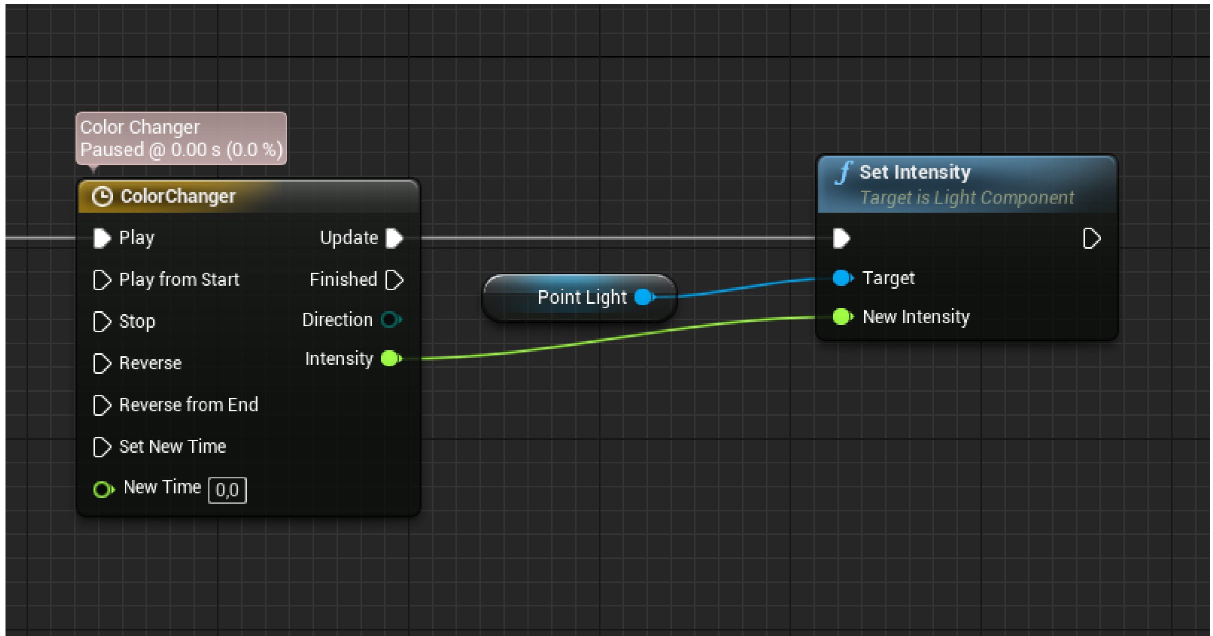

Рисунок 39. Логика изменения интенсивности света в зависимости от кривой в ноде Timeline.

Запустите проект, интенсивность света в Pulsar_BP должна плавно нарастать.

2.3 Варианты самостоятельного задания:

На основе Pulsar_BP сделайте так, чтобы на старте:

1. Point Light интенсивность 0, но через 5 секунд после старта игры начинает пульсировать волнообразно 4 раза, с амплитудой интенсивности от 0 до 10000 в течение 10 секунд.
2. Интенсивность Point Light равномерно изменяется с 1000 до 10000 и обратно до 1000 в течение 6 секунд со старта игры. По окончанию, изменяется цвет Point Light на любой другой и повторно запускается таймлайн, цикл продолжается бесконечно.
3. Цвет Point Light красный. Интенсивность Point Light вырастает с 0 до 20000 в течение 3 секунд, затем меняется цвет на синий и убывает с 20000 до 0 в том же таймлайне. По возвращению интенсивности в 0 цвет меняется на красный. Цикл продолжается бесконечно.

Занятие #3 Программирование на Blueprints. Часть II

3.1 Результаты занятия

- уметь создавать собственные события (events) в Blueprints;
- уметь создавать собственный функции в Blueprints;
- уметь создавать собственные переменные в Blueprints;
- уметь добавлять package в проект;
- уметь детектировать коллизии в Unreal Engine 4.

3.2 Теоретические сведения

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 вашем Event Graph и наберите в поиске Add Custom Event. Дайте название вашему событию (рисунок 40).

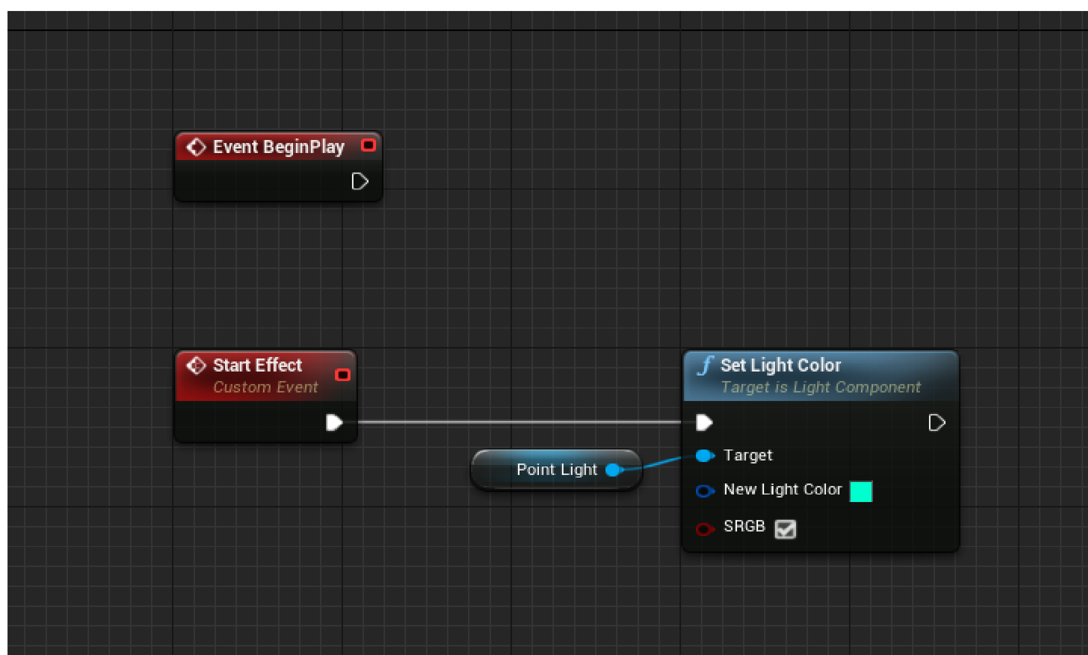

Рисунок 40. Пользовательское событие Start Effect и подключенный нод Set Light Color.

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 вашего события - в описанном примере **Start Effect** (рисунок 41).

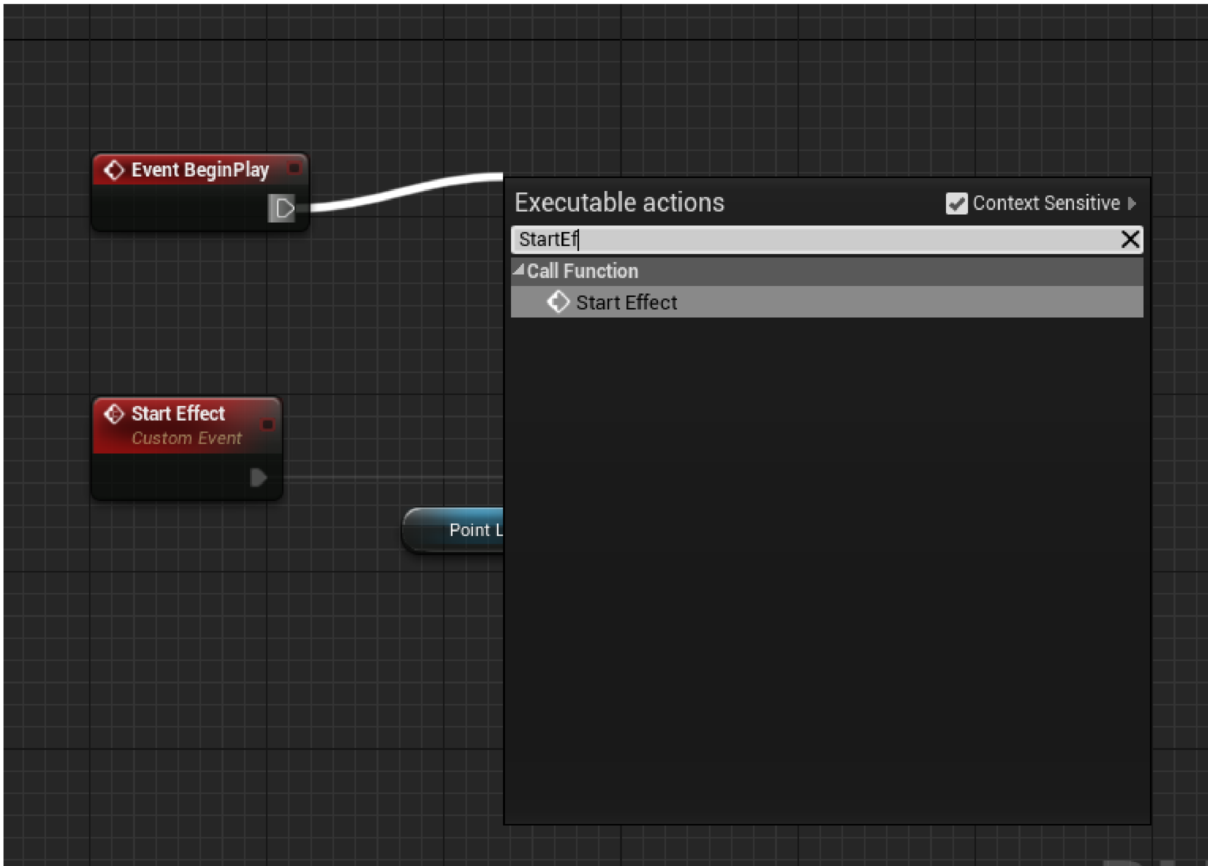

Рисунок 41. Вызов пользовательского события Start Effect в событии BeginPlay.

Вы можете добавлять входные аргументы для ваших событий. Щелкните на нод вашего события и в панели Details добавьте аргументы в поле Inputs. Вы можете дать названия вашим аргументам, а также выбрать тип данных (рисунок 42).

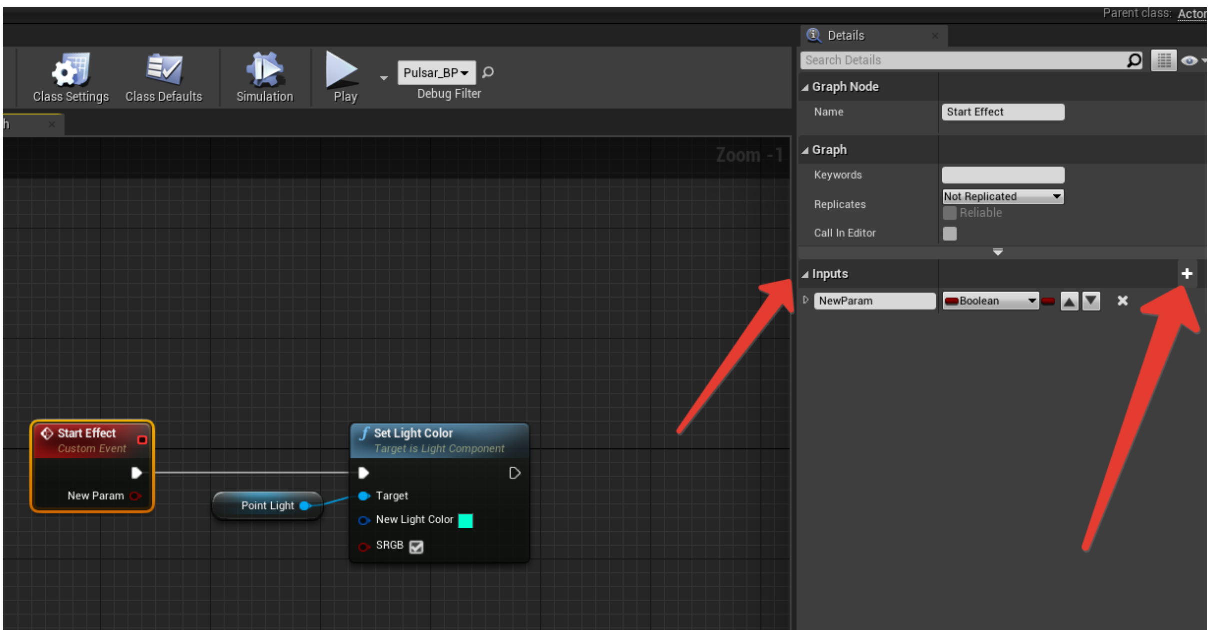

Рисунок 42. Добавление входных аргументов для пользовательского события.

Создайте следующий аргумент вашего ивента: имя **New Color**, тип данных **Linear Color**. Свяжите появившийся пин в ноде ивента с входящим пином нода **Set Light Color** (рисунок 43). Обратите внимание, что вы можете изменять входной аргумент в ноде вызова вашего ивента.

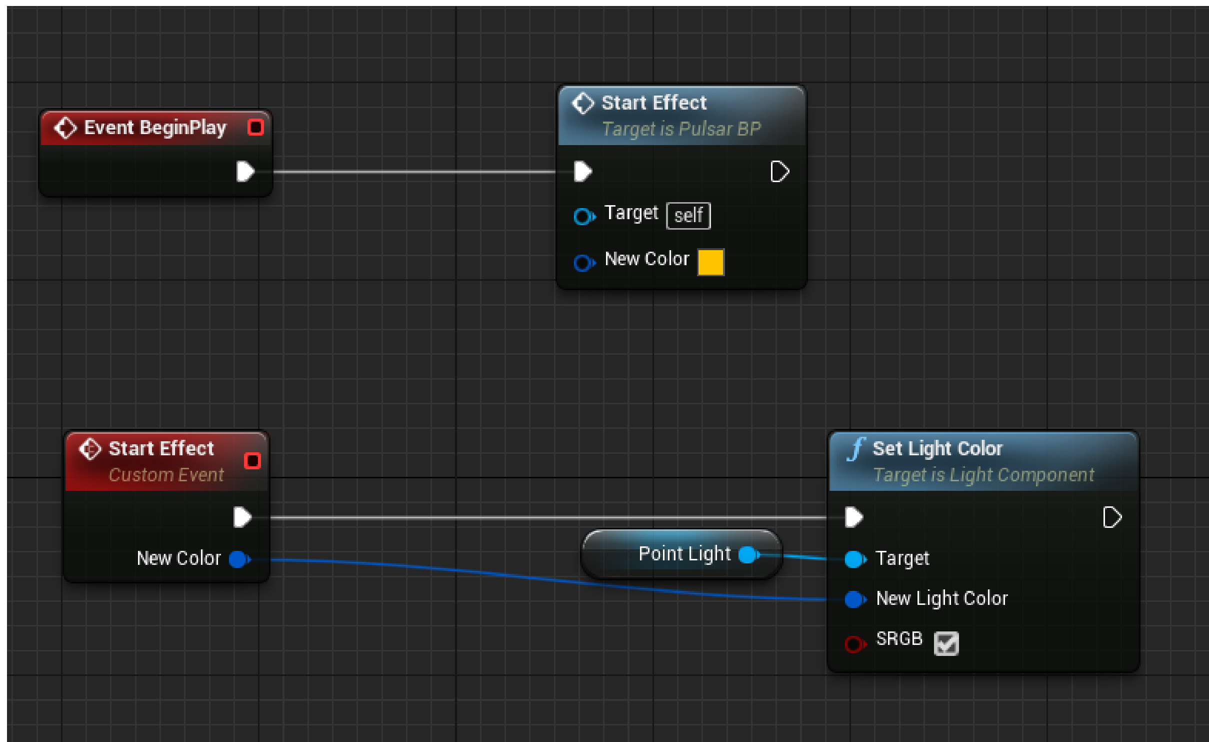

Рисунок 43. Обратите внимание, что аргумент события появился в ноде вызова вашего события.

Подробнее про пользовательские события вы можете прочитать в официальной документации.

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 44).

В момент создания новой функции у вас появится элемент с именем вашей функции. Все функции описываются в своих 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 нод вашей функции (рисунок 45).

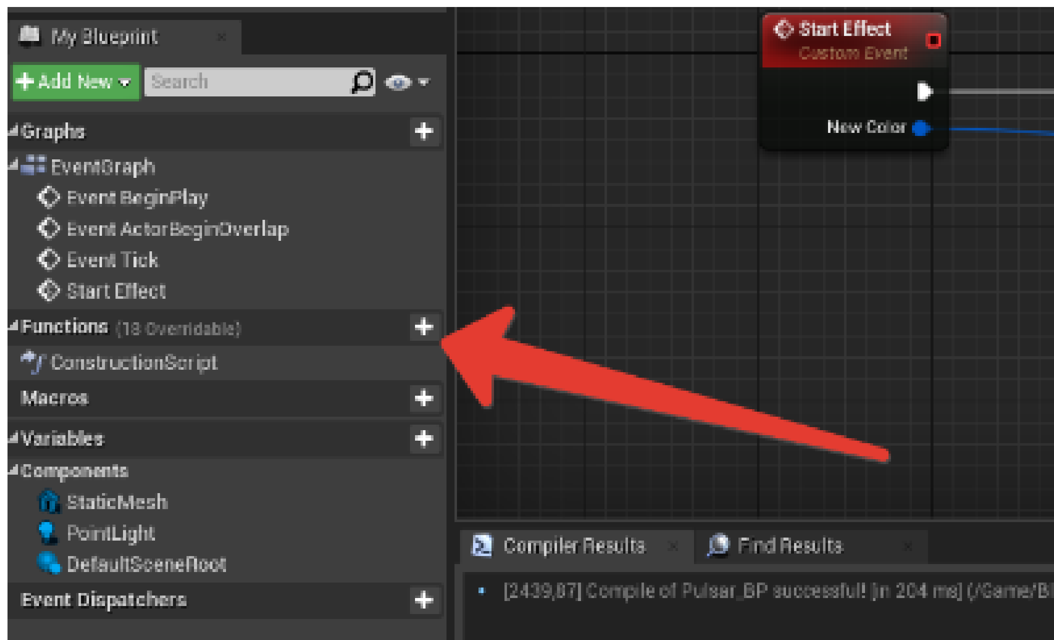

Рисунок 44. Поле Functions содержит список всех пользовательских функций, которые созданы в вашем классе.

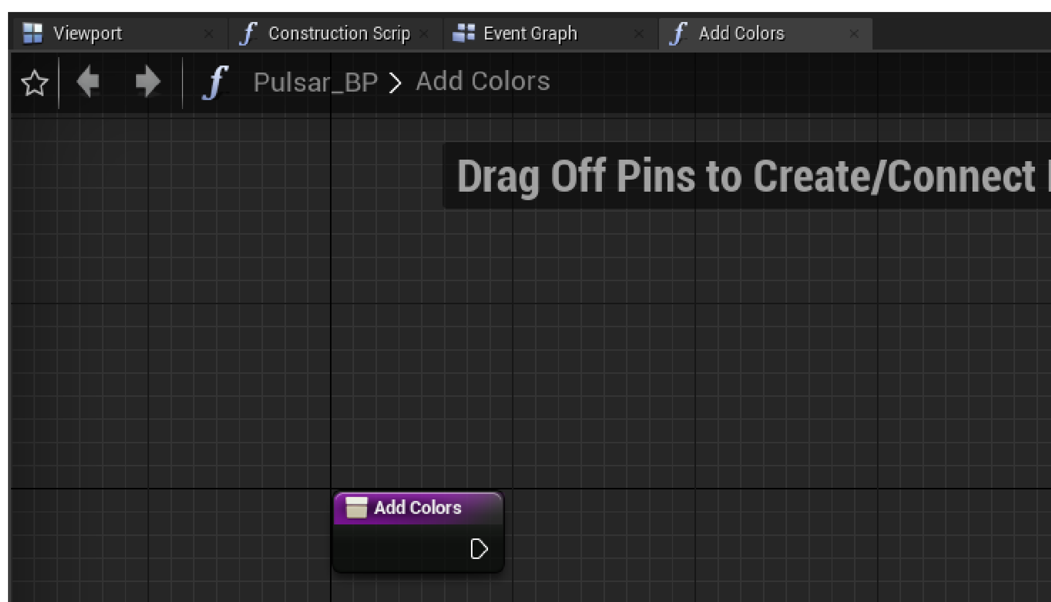

Рисунок 45. Для каждой функции создается отдельная вкладка со своим окном, в котором пишется логика.

Давайте напишем функцию, которая будет складывать два цвета (**Linear Color**) и возвращать результирующий цвет (рисунок 46). Так же, создайте два входных аргумента с именем Color1 и Color2 типа данных **Linear Color** и выходной аргумент с именем ResultColor типа **Linear Color**.

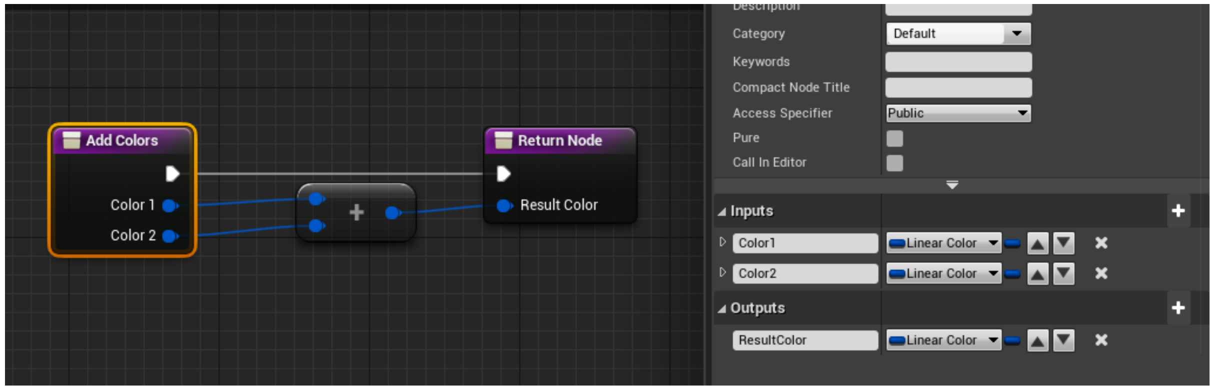

Рисунок 46. Обратите внимание на аргументы и их типы данных.

Вернёмся в Event Graph, вызов функции выполняется нодом с именем функции - так же, как и в ситуации с кастомными ивентами и в ситуации классического программирования (рисунок 47).

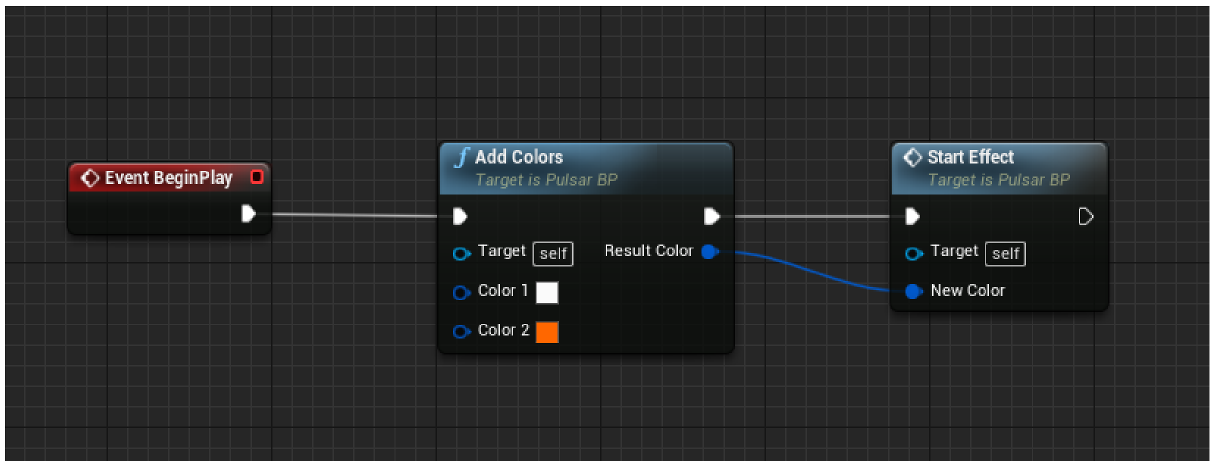

Рисунок 47. Вызов функции осуществляется написанием её имени.

Иногда не является обязательным наличие у функции execution соединения, для этого вы можете включить флаг **Pure** в панели **Details** вашей функции. В таком случае вид вызова вашей функции преобразится (рисунок 48).

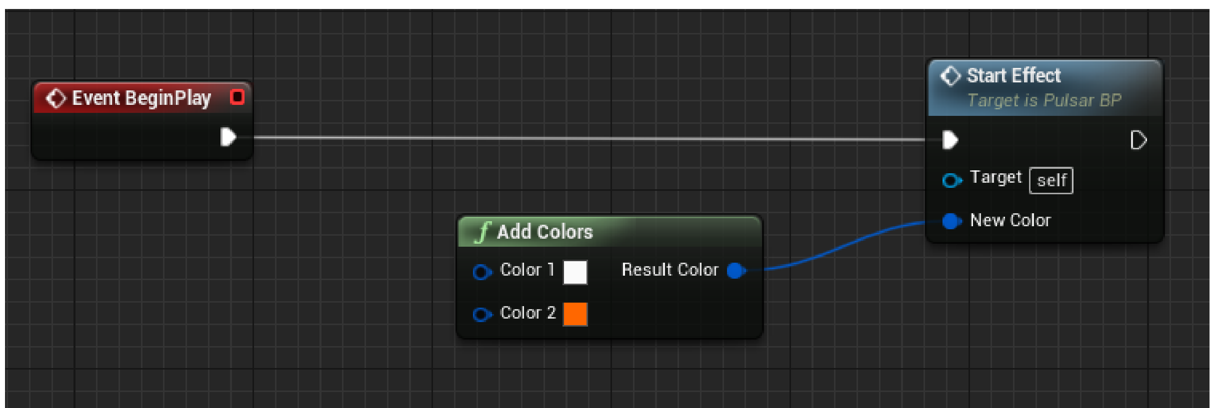

Рисунок 48. Pure функция не имеет execution соединения.

В панели Details функции вы можете указать **Access Specifier** (уровень доступа) в виде классических **Public, Private и Protected.**

Подробнее про функции читайте в официальной документации Unreal Engine.

Конечно же, вы можете создавать свои собственные переменные и также давать им уровень доступа (рисунок 49).

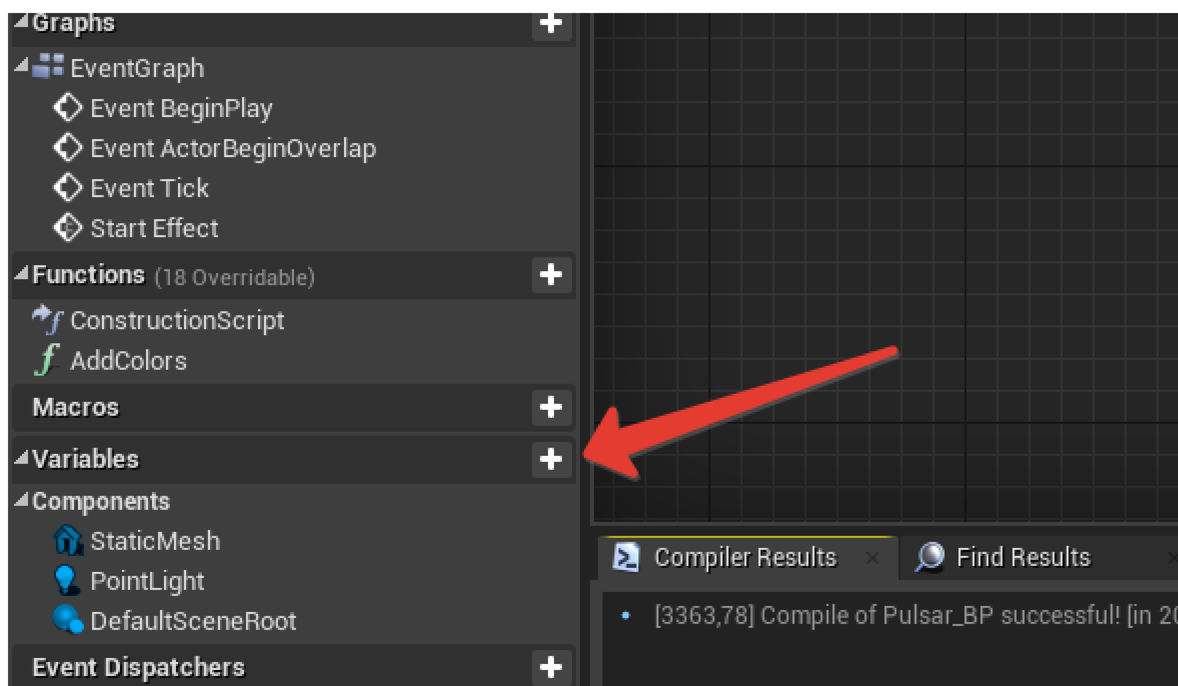

Рисунок 49. Во вкладке Variables перечислены все пользовательские переменные.

Unreal Engine 4 поддерживает все основные типы данных (целые числа, дробные числа, bool значения, строковые значения и другие) и множество комплексных типов данных, такие как как векторы, списки, цвета и прочие.

Давайте привнесем интерактивность в проект и добавим управляемого персонажа. После создания Blank проекта вы можете 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** (рисунок 49,50).

После добавления контента вы увидите, что появились новые папки с файлами. В папке Third Person BP -> Blueprints вы можете найти блюпринт уже настроенного персонажа. Вы можете попробовать перетащить 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** лежит нужный нам **Game Mode** для персонажа от третьего лица (рисунок 52).

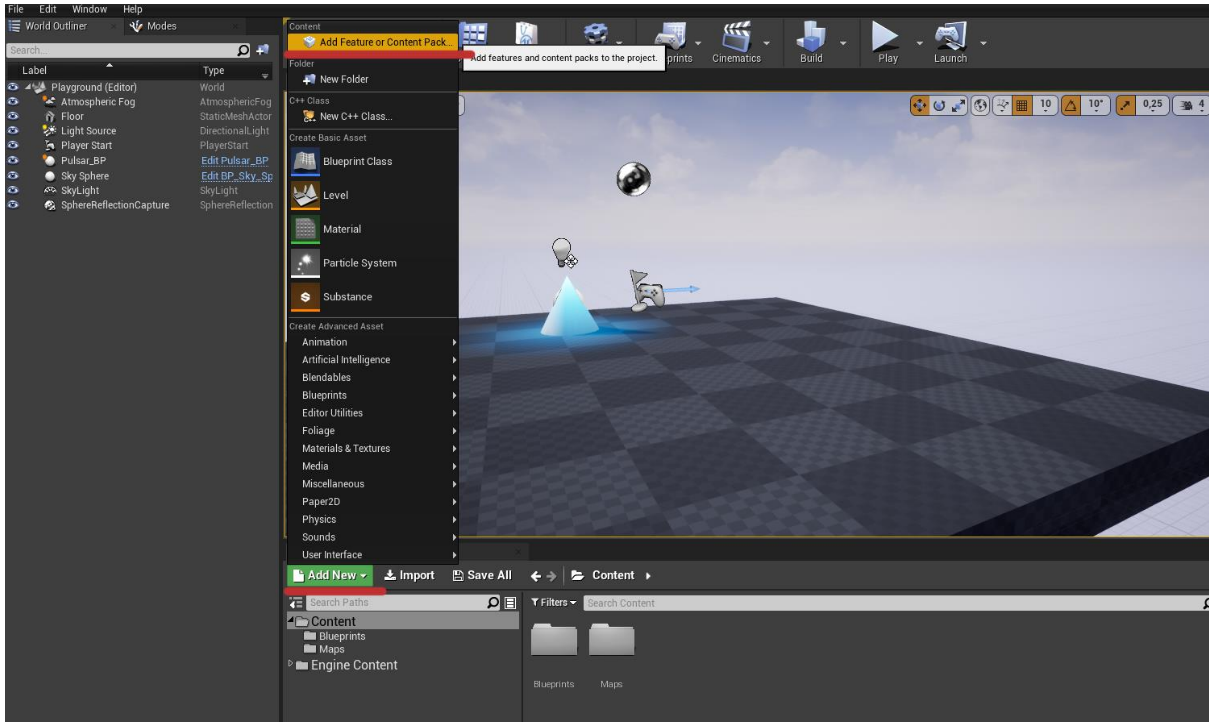

Рисунок 50. Для добавления дополнительных стандартных ассетов используйте опцию Add Feature or Content Pack.

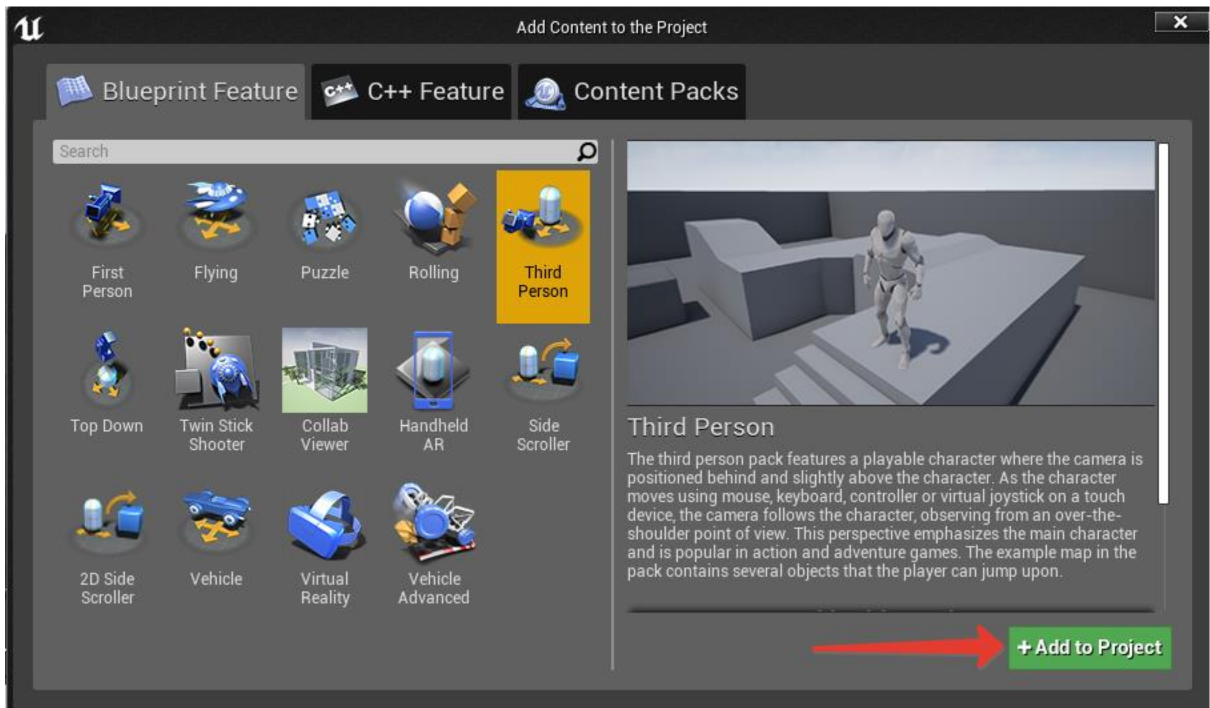

Рисунок 51. Большинство стандартных ассетов Unreal Engine 4 посвящены распространенным игровым жанрам и типам приложений.

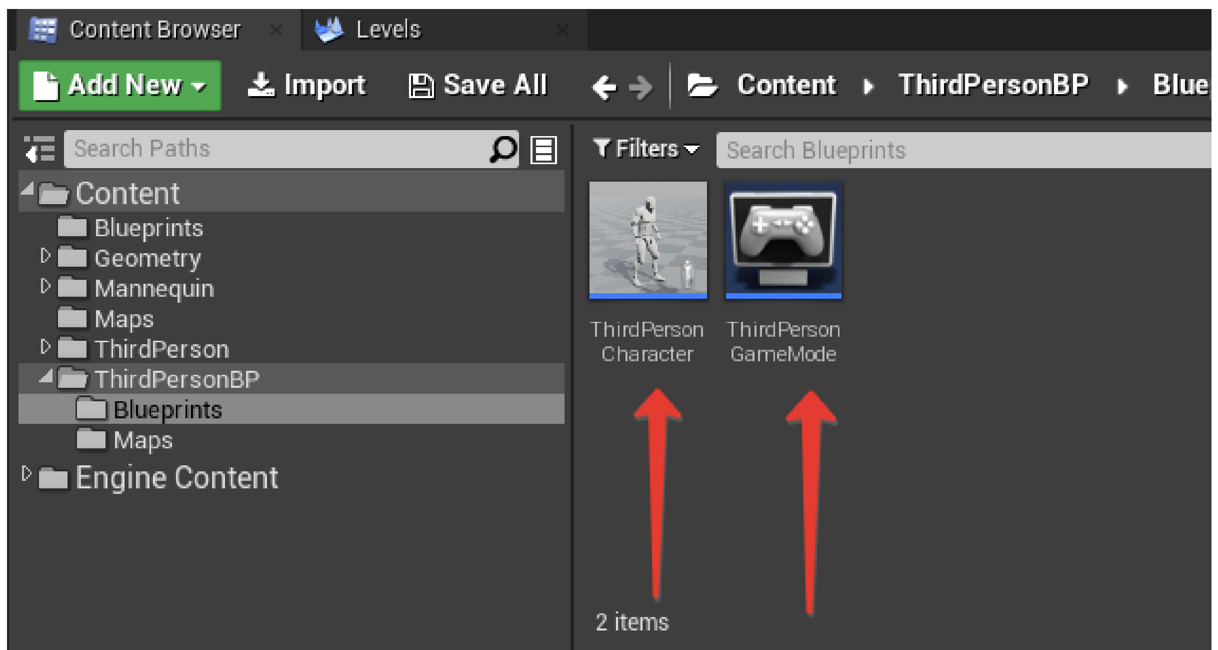

Рисунок 52. Найдите блюпринт-класс стандартного управляемого персонажа в папке Blueprints импортированного контента.

Чтобы установить этот **GameMode,** нажмите на кнопку **Blueprints** верхней панели - **Project Settings: Game Mode-> Select GameModeBase Class-> ThirdPersonGameMode** (рисунок 53).

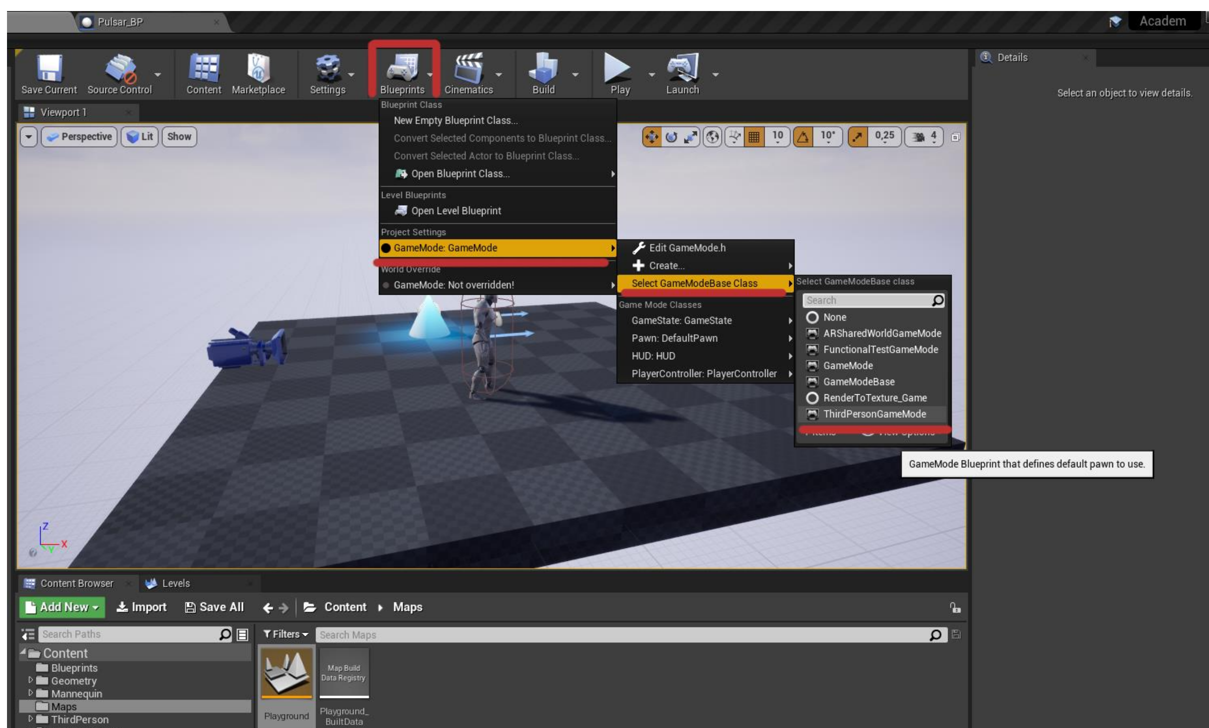

Рисунок 53. Доступ к базовым настройкам логики проекта можно найти, нажав на кнопку Blueprints.

Теперь при запуске проекта вы сможете управлять персонажем, однако их

будет двое (при условии, что вы не удалили после тогор как поставили на уровень первый раз). Это связано с тем, что **ThirdPersonGameMode** по умолчанию создает блюпринт персонажа в позиции актора **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 54).

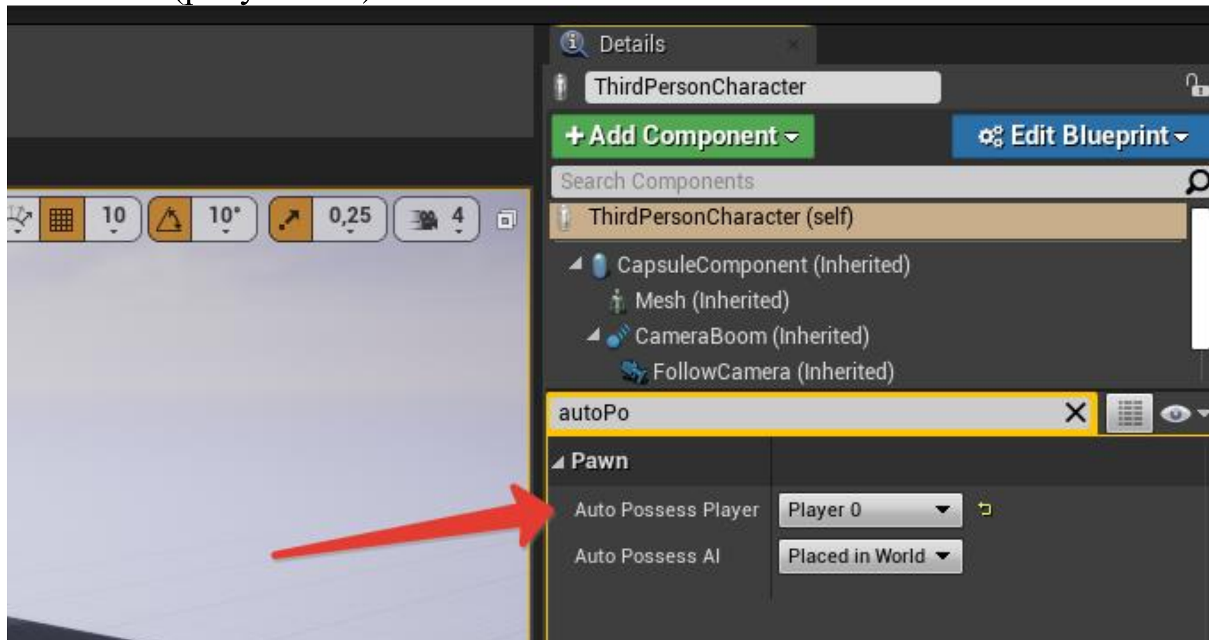

Рисунок 54. Используйте поисковую строку, чтобы найти нужный параметр (Auto Possess Player) на объекте ThirdPersonCharacter.

Давайте сделаем так, чтобы при приближении нашего персонажа конус менял цвет на рандомный. В первую очередь давайте добавим **Sphere** коллайдер в **Pulsar_BP** (рисунок 55).

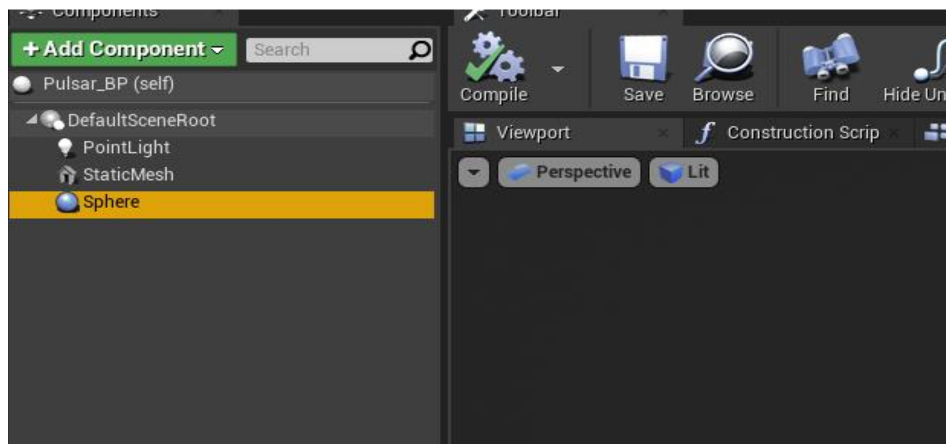

Рисунок 55. Добавьте компонент Sphere к классу Pulsar_BP.

Убедитесь в том, что пресет (настройки) данного Sphere коллайдера установлен в состояние **Overlap All Dynamics** (рисунок 56). Подробнее про то, как

работает коллизия в Unreal Engine 4, вы можете узнать в официальной документации.

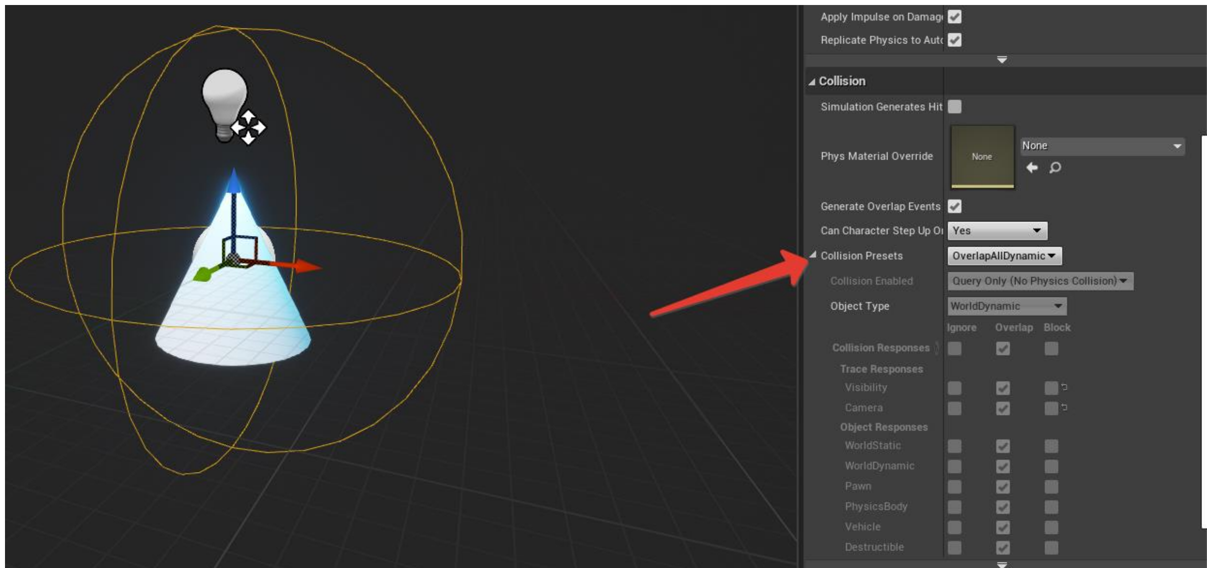

Рисунок 56. Настройки коллизии можно найти в области Collision в панели Details (если в данный момент выбран компонент Sphere)

Откройте **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 своего 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 57).

Подробнее о всех событиях, обрабатывающих коллизию, в официальной документации.

Выберите **Event OnComponentBeginOverlap**. Будет создан новый нод. Аргумент OtherActor будет записывать в себя ссылку на актора, коллайдер которого наш персонаж пересечёт. Нод **Cast To** является нодом приводящим актора к определенному классу. Первый выходной пин этого нода будет выполняться, если приведение типа прошло успешно. Приведите (совершите Cast) OtherActor к Pulsar_BP (рисунок 58) и выведите имя нашего объекта класса.

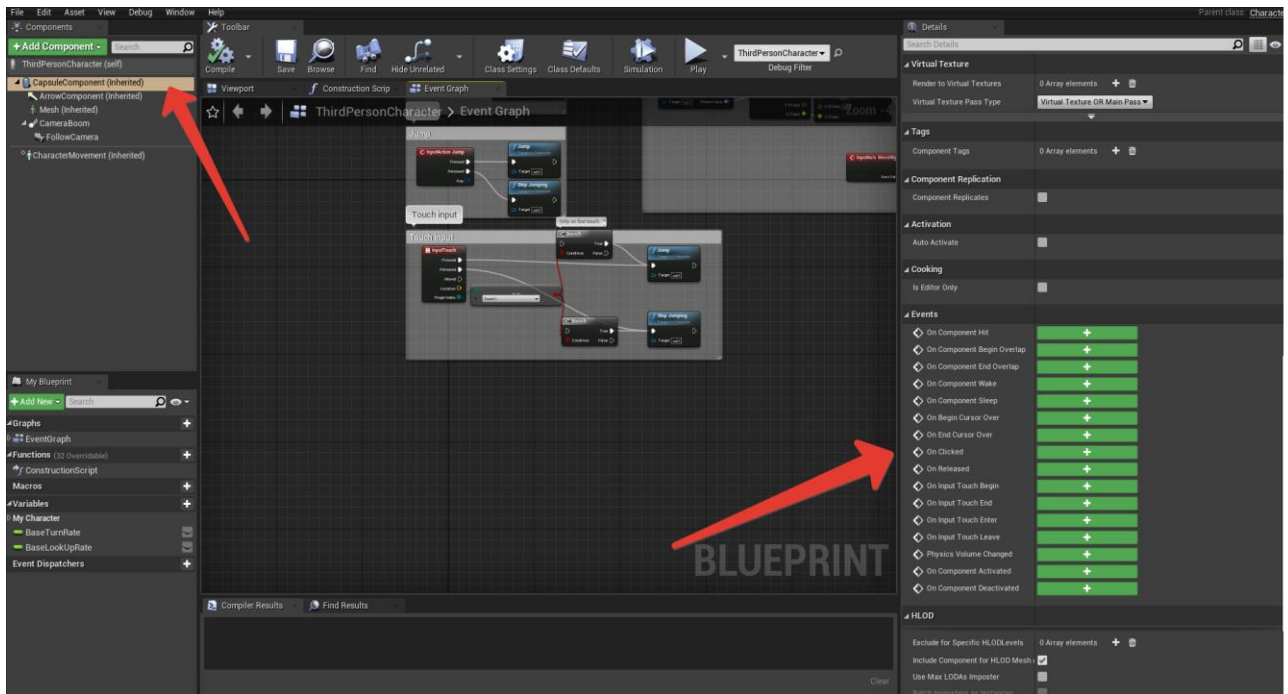

Рисунок 57. Список доступных событий коллизии можно найти, нажав на компонент Capsule Component.

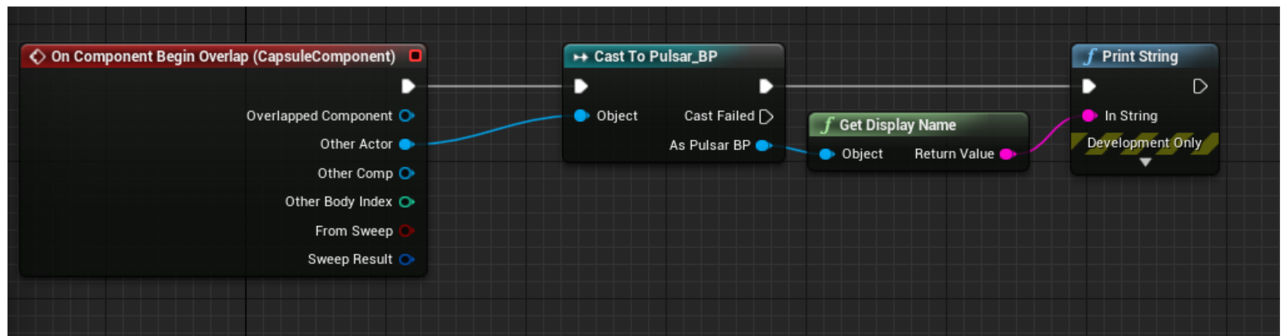

Рисунок 58.

Запустите проект. Теперь, при приближении персонажа к объекту класса **Pulsar_BP**, в консоль должно вывестись имя объекта (рисунок 59).

Рисунок 59. Вывод имени класса Pulsar_BP в консоль.

Реализуйте функцию **SetRandomColor** в **Puslar_BP** и вызовите её при приближении персонажа (рисунок 60, 61).

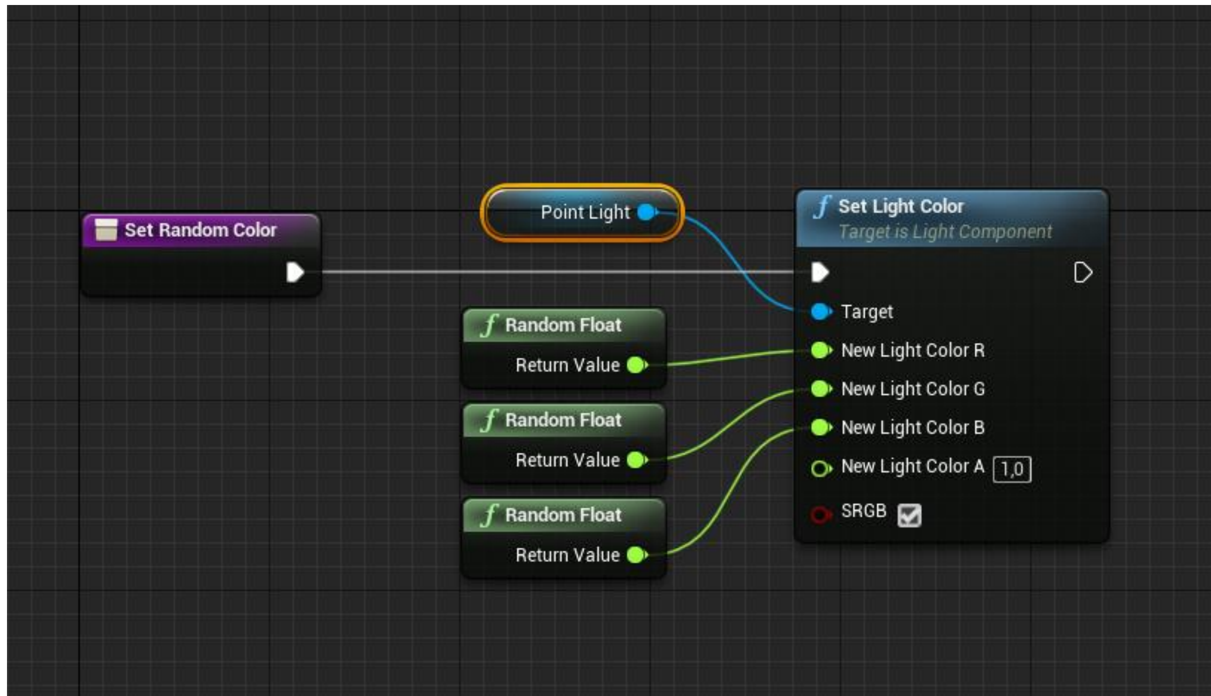

Рисунок 60. *Функция Random Color в Pulsar_BP, Random Float возвращает псевдослучайное значение от 0 до 1.*

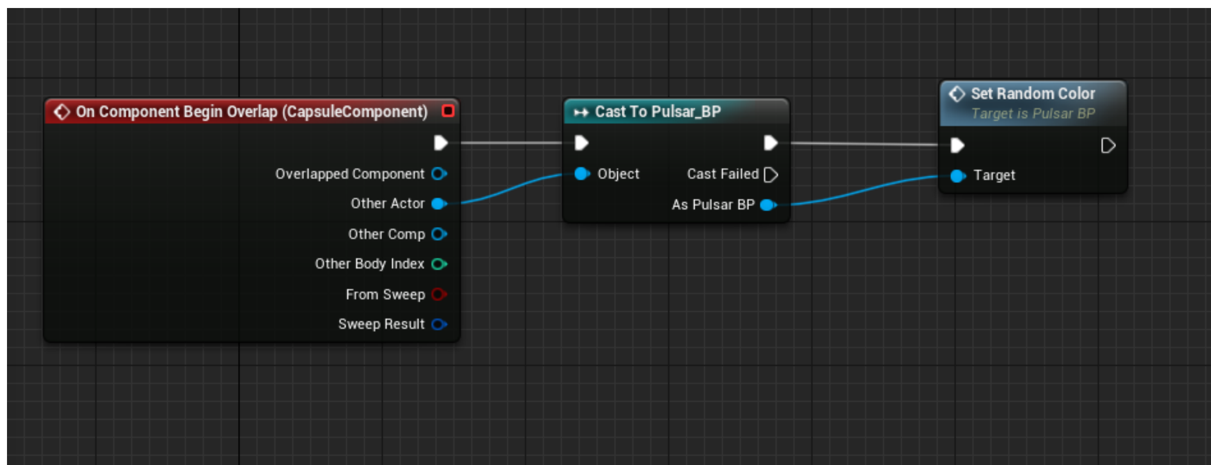

Рисунок 61 *Фрагмент кода из Third Character Blueprint - нод Cast To при успешном приведение типа возвращает ссылку на объект класса, к которому совершается приведение, через которую мы можем вызвать все публичные функции и ивенты данного класса.*

P.S. Зачастую "дешевле" (имеется в виду оптимизация) использовать теги и интерфейсы, чтобы определить объект, с которым происходит коллизия. Интерфейсы будут рассмотрены в следующем занятии.

Занятие #4 Работа с материалами. Material Editor

4.1 Результаты занятия

- уметь создавать собственные материалы;
- уметь писать собственные шейдеры для материалов;
- уметь создавать экземпляры материалов;
- уметь применять материалы на меши;

4.2 Теоретические сведения

Material Editor — это нодовый редактор, позволяющий писать 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 62).

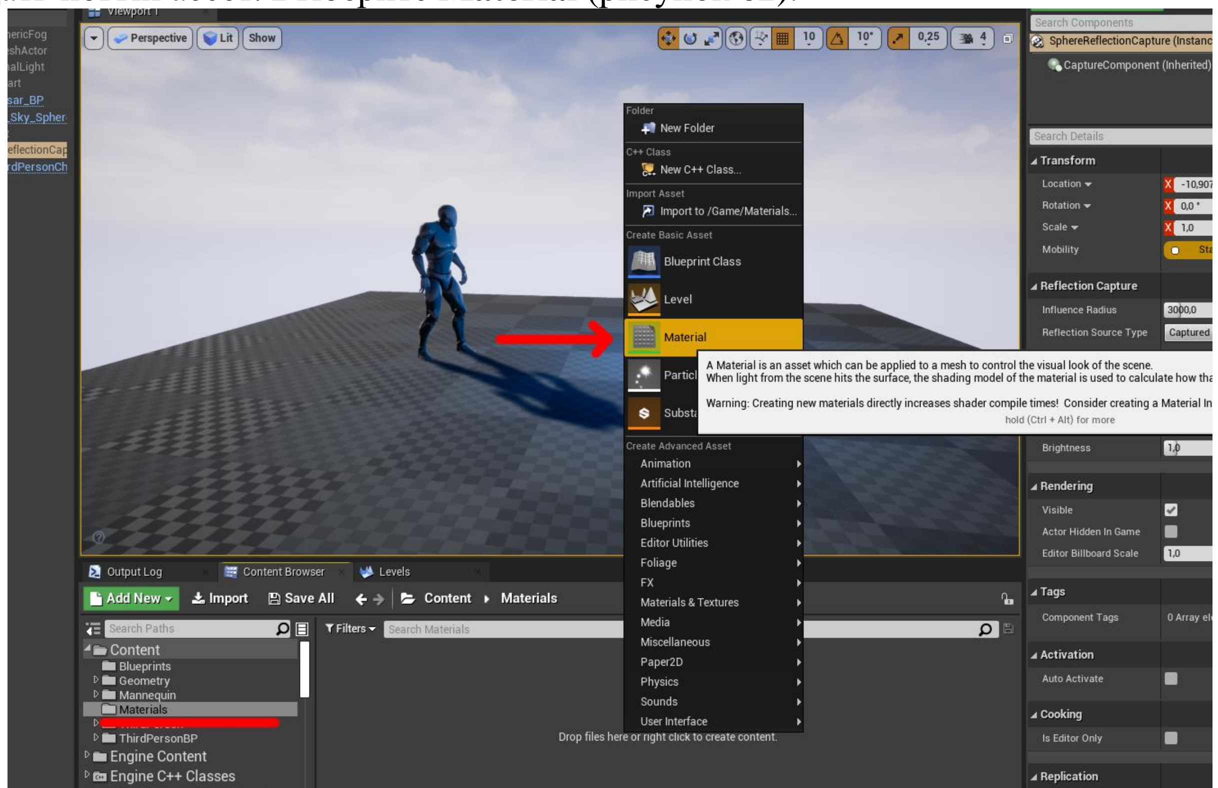

Рисунок 62. Создание ассета типа Material.

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 один главный выходной нод (рисунок 63).

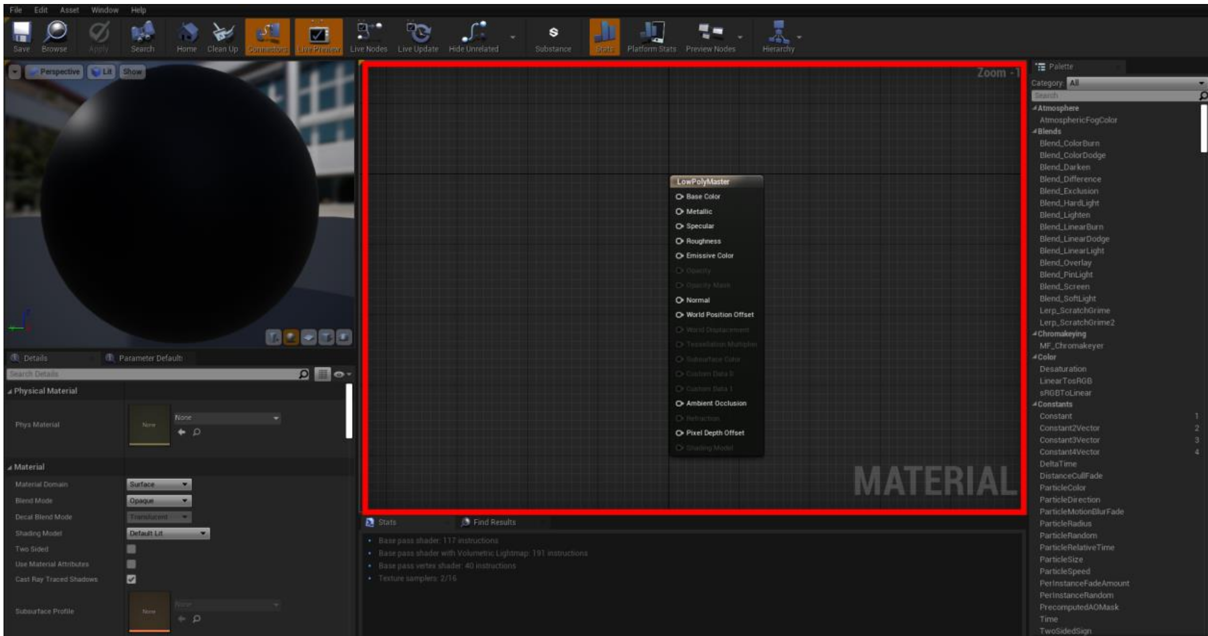

Рисунок 63. Главное окно Material. В нём вы видите главный выходной нод.

Панель Viewport отображает то, как выглядит материал на тестовых мешах. Вы можете изменить превью-меш, нажав их иконки в правом нижнем углу Viewport (рисунок 64).

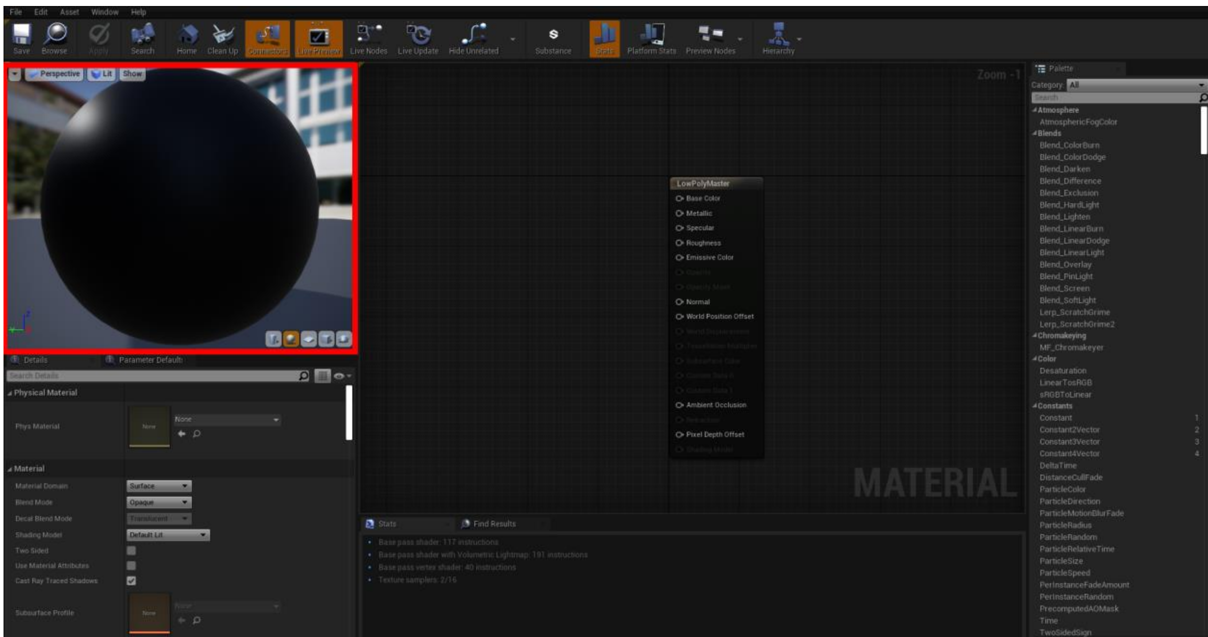

Рисунок 64. Панель Viewport.

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 прочие (рисунок 65).

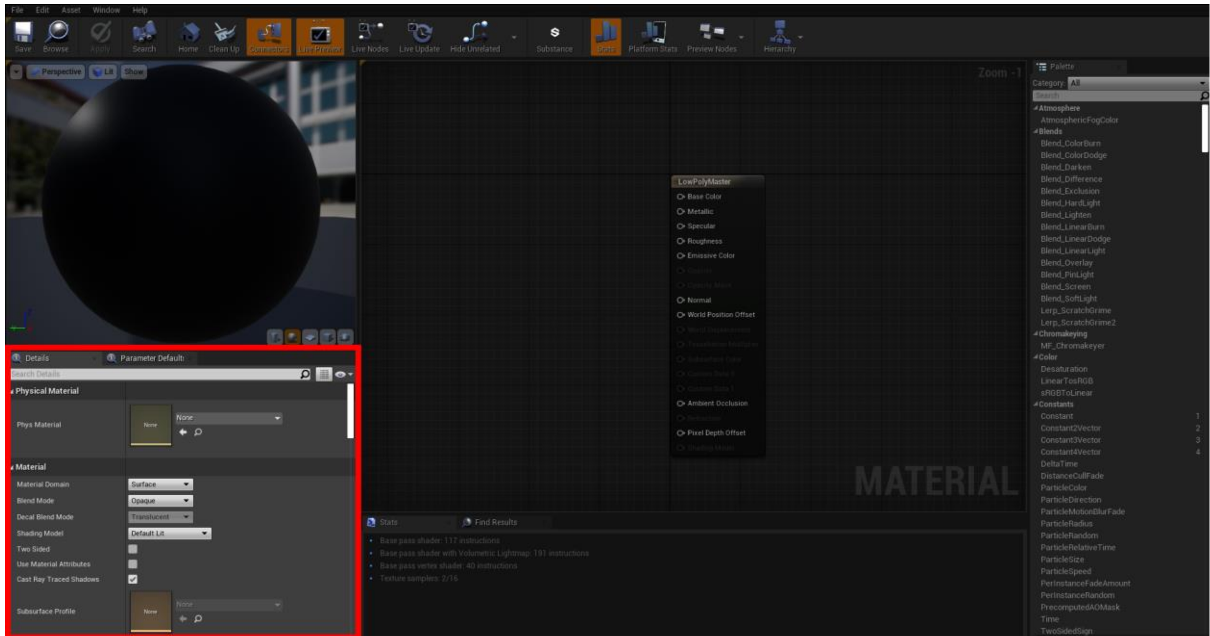

Рисунок 65. Панель Details.

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 66).

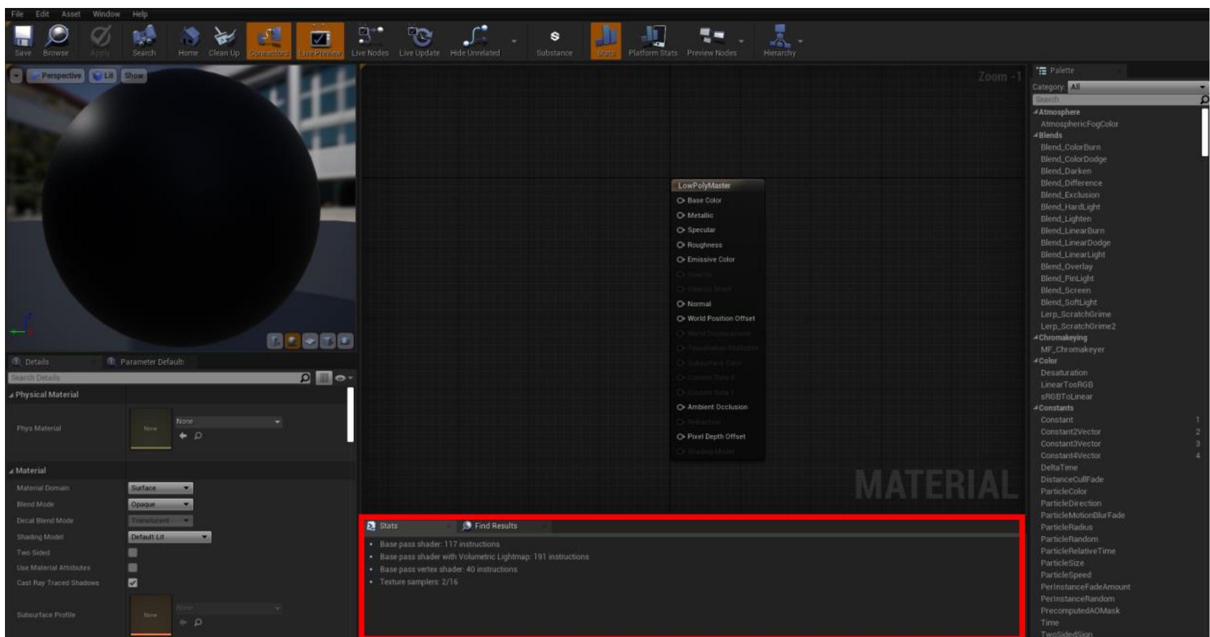

Рисунок 66. Панель Stats.

Панель Palette содержит список всех доступных нодов 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 67).

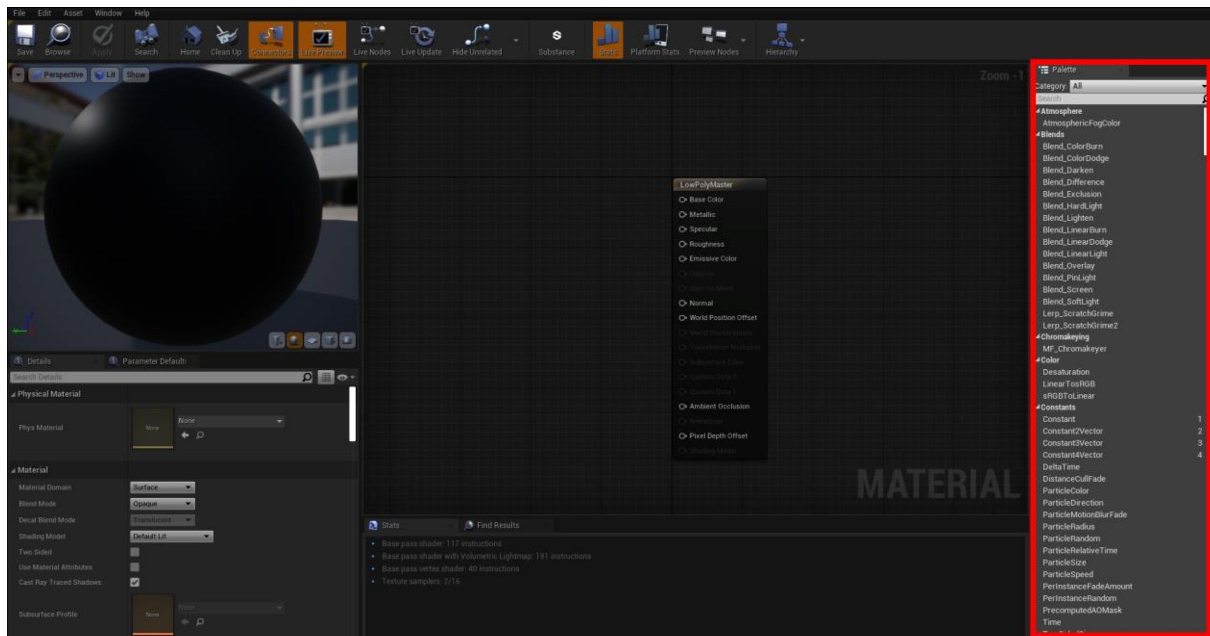

Рисунок 67. Панель Palette. Смотрите официальную документацию, чтобы выяснить назначения множества нодов.

Давайте сделаем простой шейдер, на основе которого будет собран материал с несколькими простыми параметрами - основным цветом (**Base Color или Albedo**), металличностью(**Metallic**) и шероховатостью(**Roughness**). В дальнейшем в данном пособии будут использоваться именно англоязычные термины для описания параметров материала, так как именно эти термины широко используются при коммуникации в индустрии видеоигры.

Большой нод с именем вашего материала и списком пинов (**Base Color, Metallic, Specula, Roughness** и т.д.). 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 68).

Обратите внимание, что вы можете изменять значения отдельных нодов по умолчанию в панели **Details** после нажатия ЛКМ на нужный нод (рисунок 69).

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

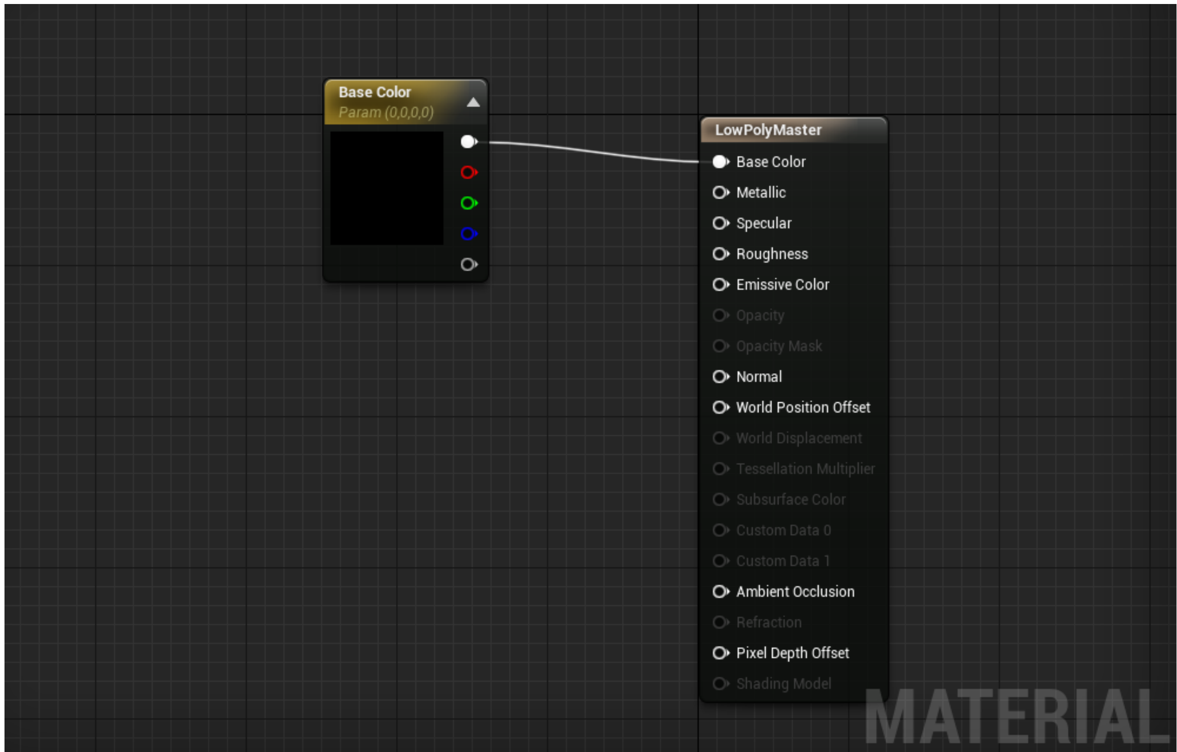

Рисунок 68. Нод BaseColor имеет тип Vector4.

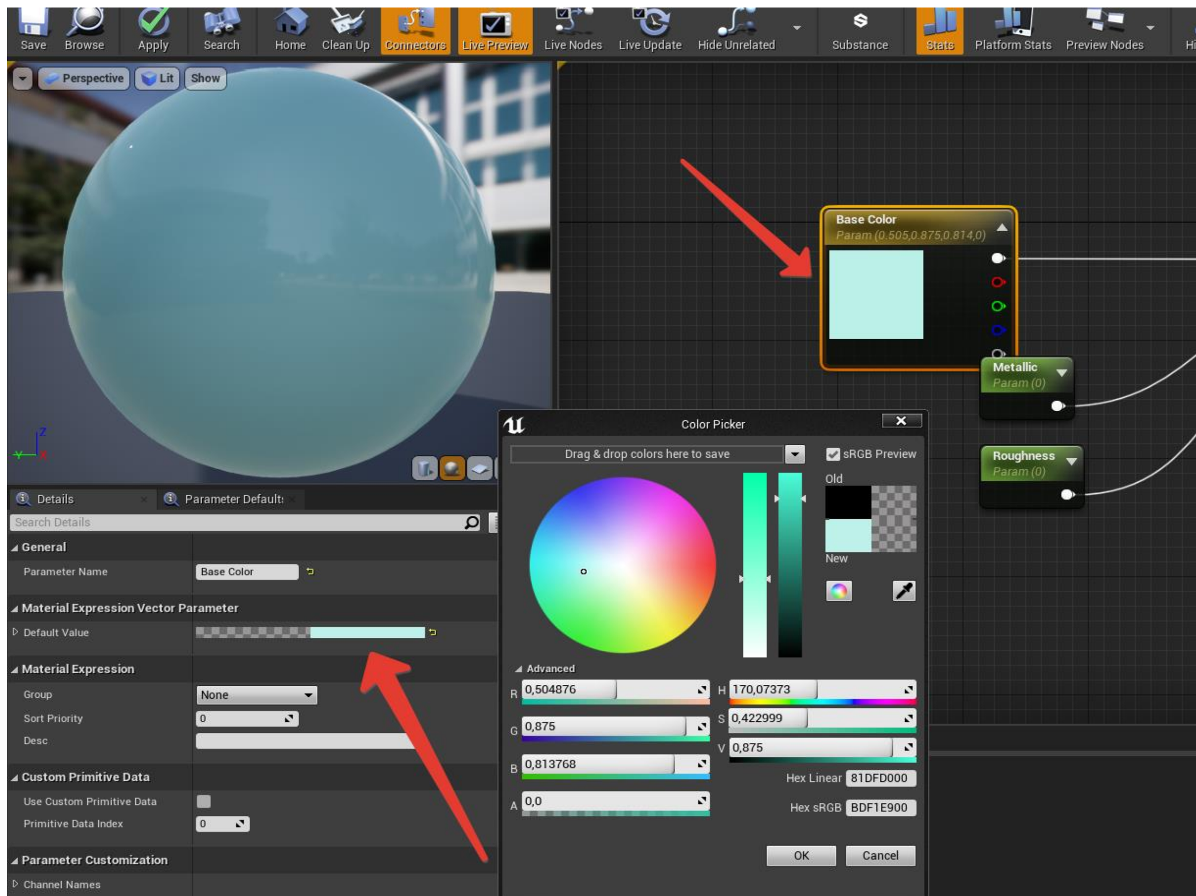

Рисунок 69. Вы можете менять значения переменных нодов в панели Details.

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 (рисунок 70).

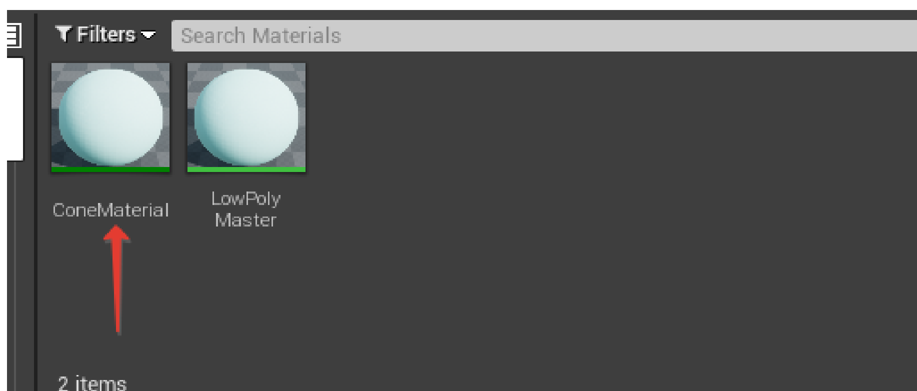

Рисунок 70. Созданный экземпляр материала будет автоматически создан в той же папке, в которой находился мастер материал.

Открыв экземпляр материала, вы можете 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.

Чтобы применить материал на меш, вы можете 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 71).

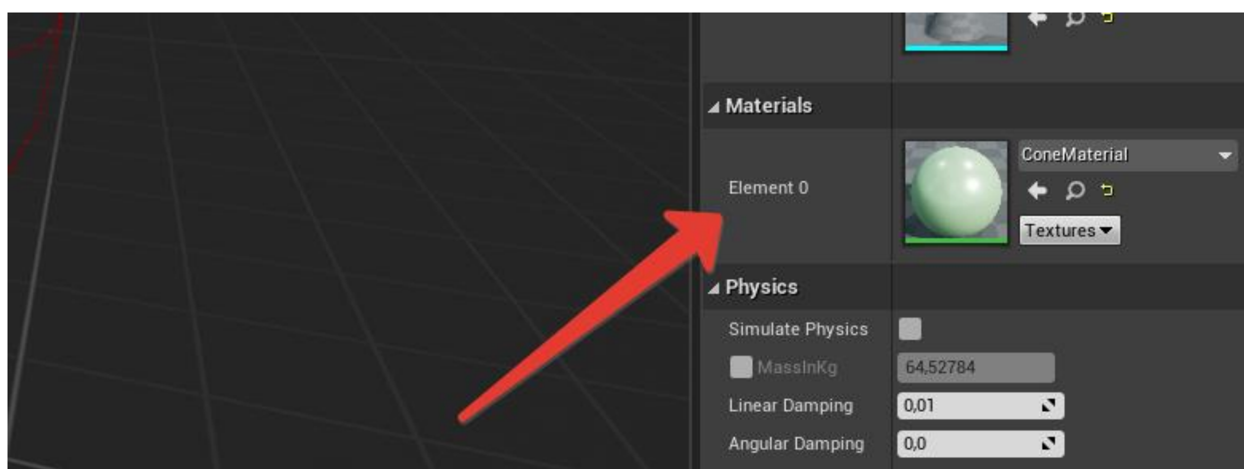

Рисунок 71. Вы можете установить ваш материал через поле Materials выбранного меша. Вы также можете получить доступ к полю Materials программно.

P.S. Созданные “вручную” ноды типов данных в **Material Graph** (например, Vector или **Texture Sample)** не являются переменными по умолчанию. Для этого вам нужно кликнуть на нужный нод и выбрать **Convert To Parameter** (рисунок 72).

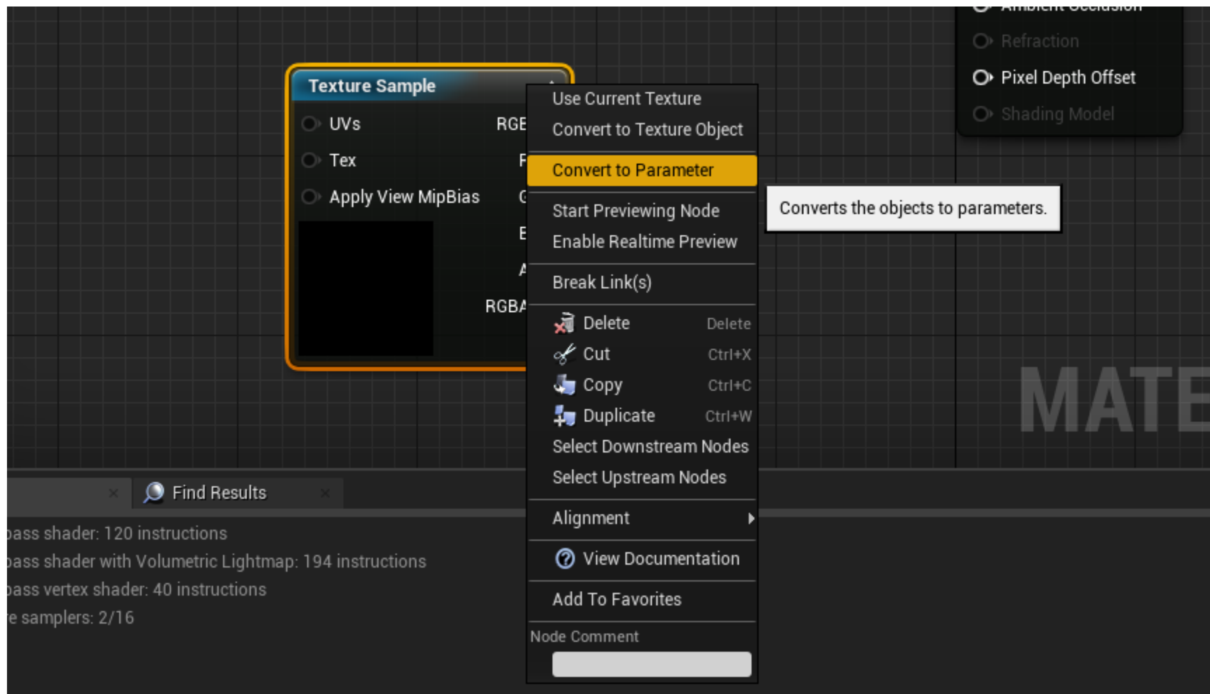

Рисунок 72. Конвертация нода в переменную происходит через выбор опции Convert To Parametr.

Подробнее про **Material Editor** вы можете узнать в официальной документации.

4.3 Варианты самостоятельного задания:

1. Сделать полноценный PBR материал с текстурными картами Albedo, Normal Map, Displacement Map.
2. Сделать материал, цвет Albedo которого плавно меняется из черного в белый в реальном времени.

Занятие #5 Работа с контентом в UE 4 II. Animation Blueprint

5.1 Результаты занятия

- уметь импортировать контент (собственный или из Marketplace);

- уметь создавать блюпринт-класс из нескольких акторов;

5.2 Теоретические сведения

Много важной информации о работе с контентом в Unreal Engine вы можете узнать в официальной документации UE 4.

Импорт ассетов может происходить несколькими способами, вы можете перетаскивать ваши ассеты в соответствующие папки или использовать кнопку **Import** в окне **Content Browser**. Вы также можете воспользоватся **Unreal Marketplace -** магазином контента для разработчиков UE 4. Там можно найти множество бесплатного контента.

В данном занятии будет использоваться следующий бесплатный контент-пак с 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 акторов. Это логично в случае с сундуком, если мы хотим в дальнейшем 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 73).

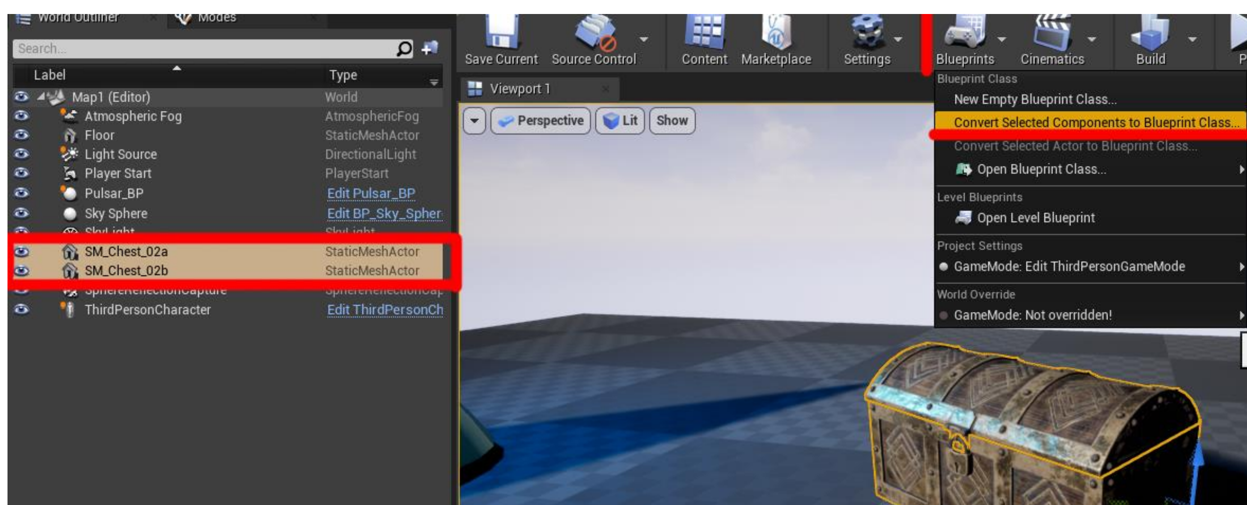

Рисунок 73. Выделив несколько акторов используйте опцию Convert Selected Components to Blueprint Class.

Данный блюпринт сундука будет использоваться в следующем занятии.

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

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

Чтобы создать Animation Blueprint, нажмите ПКМ в свободном месте, выберете вкладку **Animation**, далее **Animation Blueprint**. Будет открыто окно, в котором нужно указать **Skeleton** к которому будет приписан блюпринт. Скелет стандартного манекена - это **UE4_Mannequin_Skeleton** и нажмите **OK** (рисунок 74).

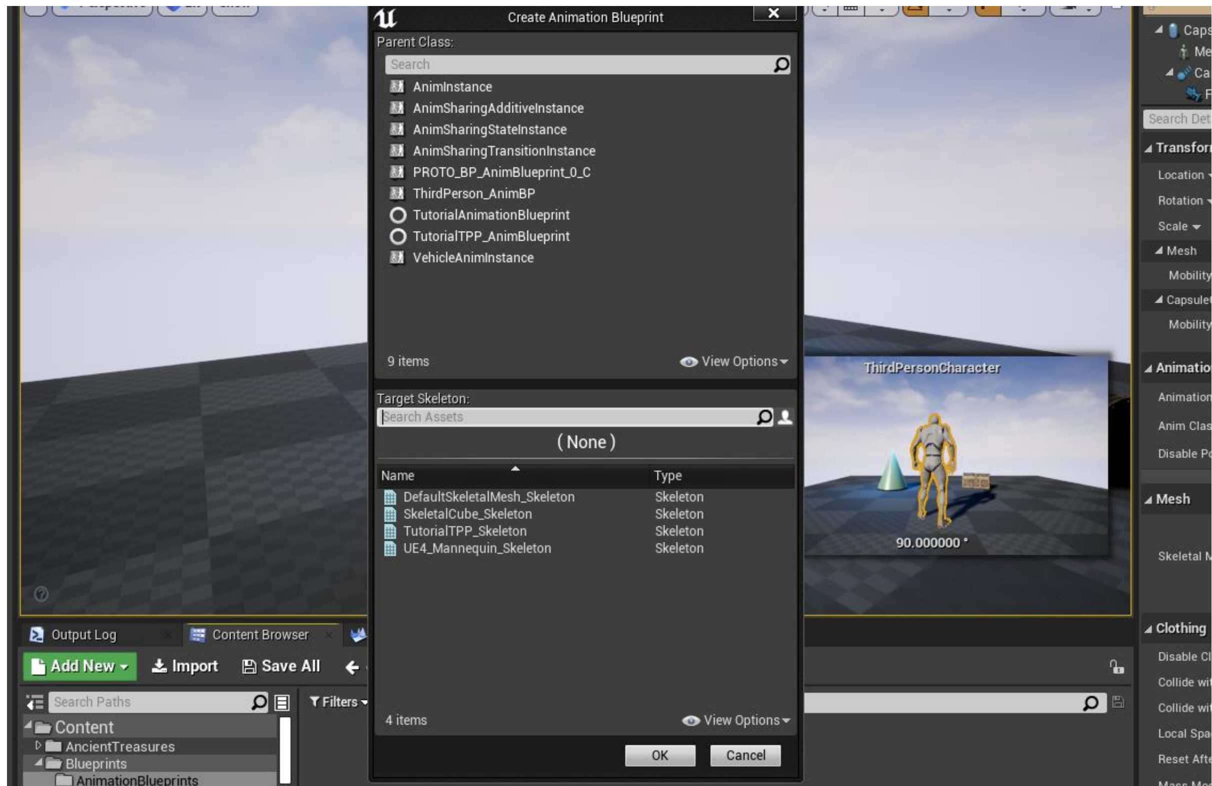

Рисунок 74.Окно выбора скелета меша.

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 (рисунок 75).

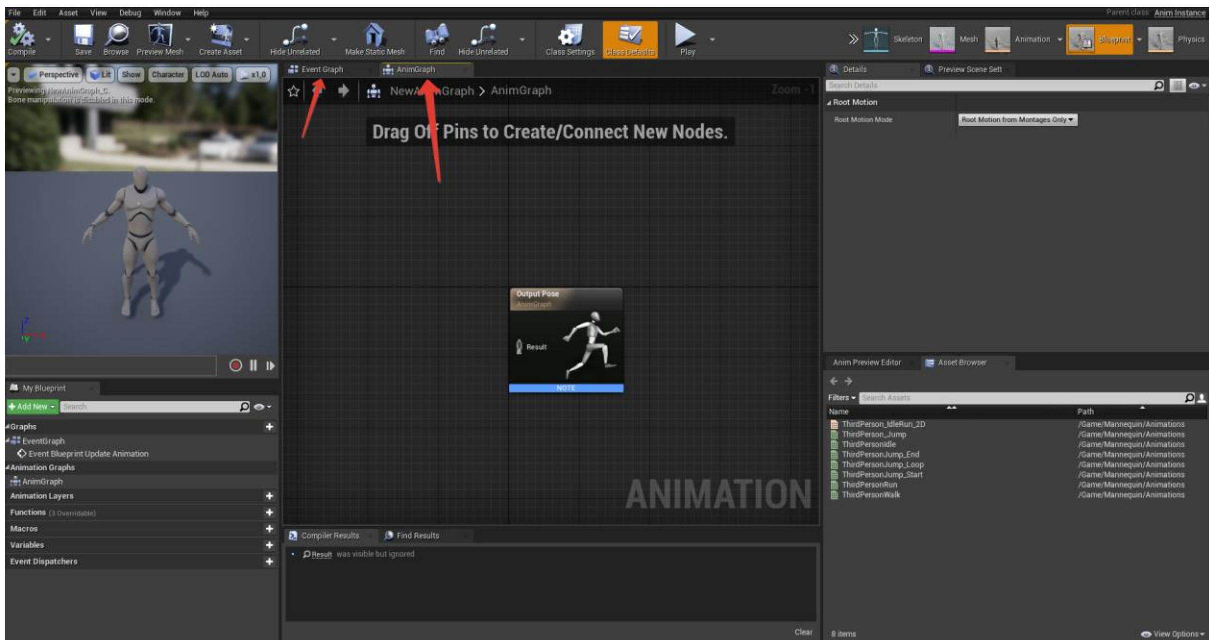

Рисунок 75. Окно Animation Graph, обратите внимание что сверху доступна вкладка Event Graph, в окне которой можно писать специальную логику для анимаций.

Находясь в окне **AnimGraph**, создайте нод **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 выходной пин стейт-машины и входной пин нода **Output Pose**. **Output Pose** - это всегда выходной нод любого Animation Blueprint. Желтоё предупреждение **Warning!** сообщает о том, что созданная стейт-машина пуста (рисунок 76).

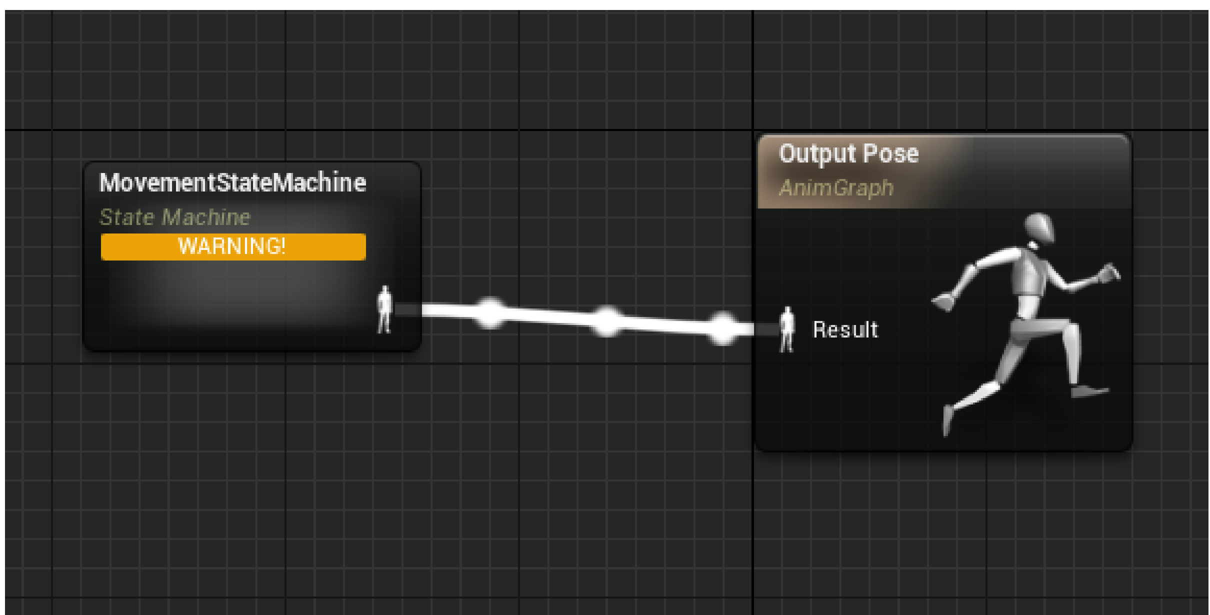

Рисунок 76. Изначально любая новая State Machine будет содержать предупреждение о том, что она пуста.

Чтобы добавить состояния в стейт-машину, нужно щелкнуть два раза на нод стейт-машины. Нод Entry является стандартным, и с него начинается выполнение стейт-машины. Вы можете перетаскивать анимации из панели Asset Browser в окно стейт-машины. Вытягивайте ЛКМ связи из границ нодов анимаций (рисунок 77).

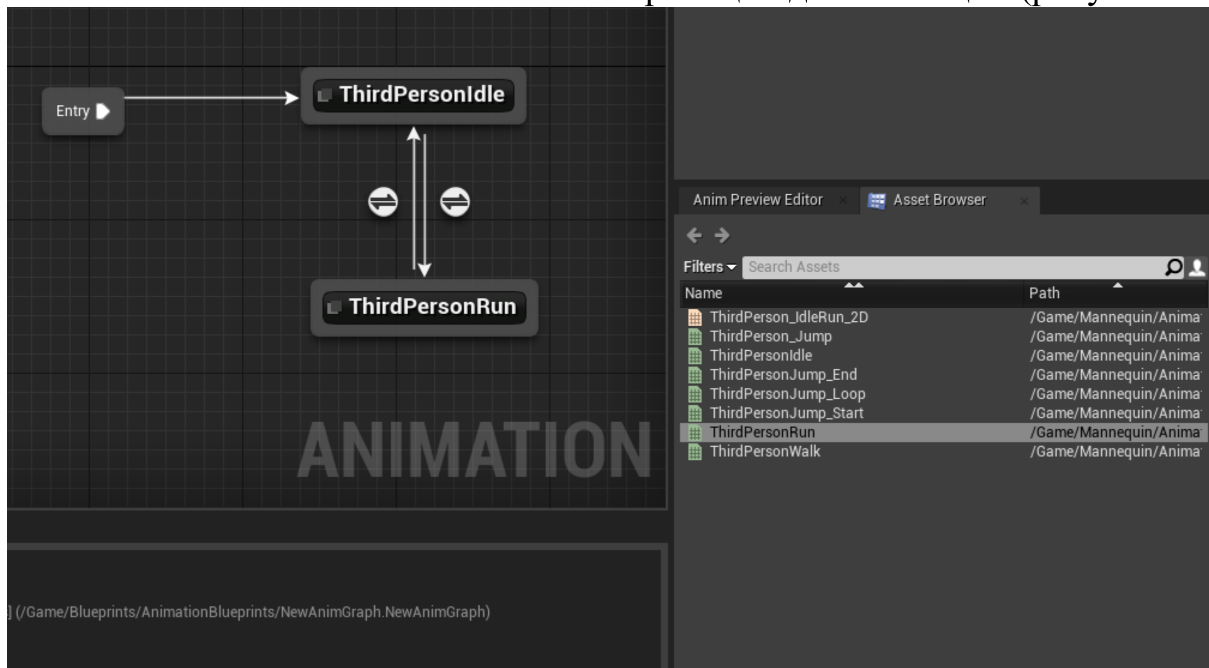

Рисунок 77. Доступные анимации находятся во вкладке Asset Browser.

Проектирование стейт-машины анимаций - это отдельный навык, который нарабатывается с опытом. Нажав двойным ЛКМ по связи, откроется окно для настройки логики связи перехода. Добавьте переменную типа float, дайте ей название (рисунок 78). В данном примере имя переменной Anim_Velocity.

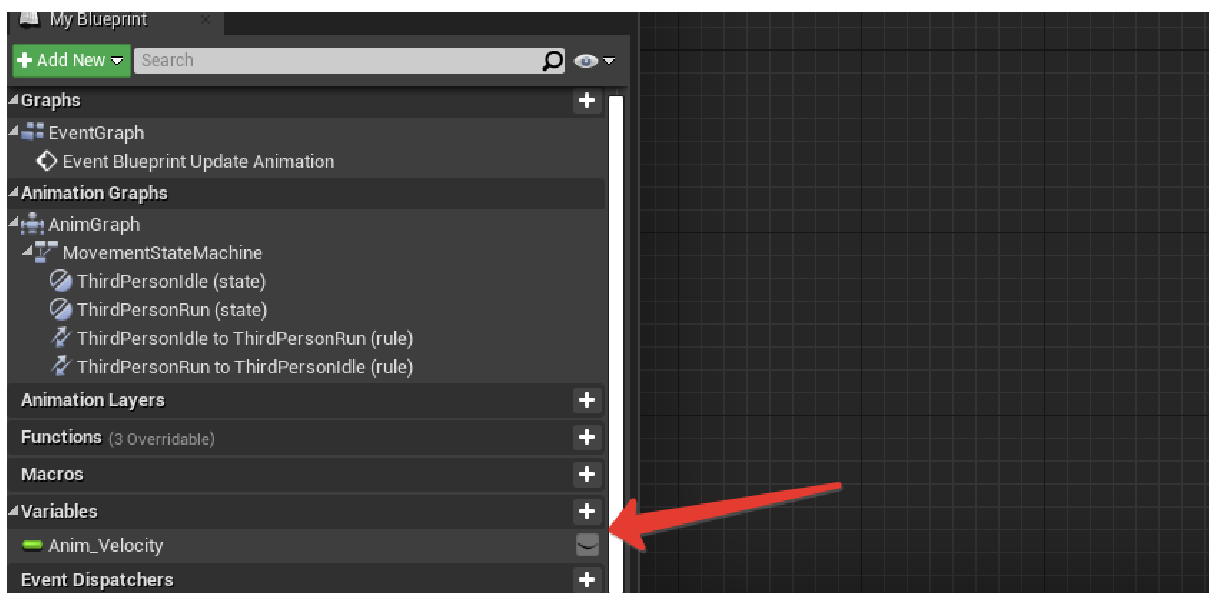

Рисунок 78. В блоке Variables перечислены все пользовательские переменные.

В связи, **выходящей** из анимации **ThirdPersonIdle** добавьте проверку значения *больше или равно 0.2* переменной Anim Velocity (рисунок 79).

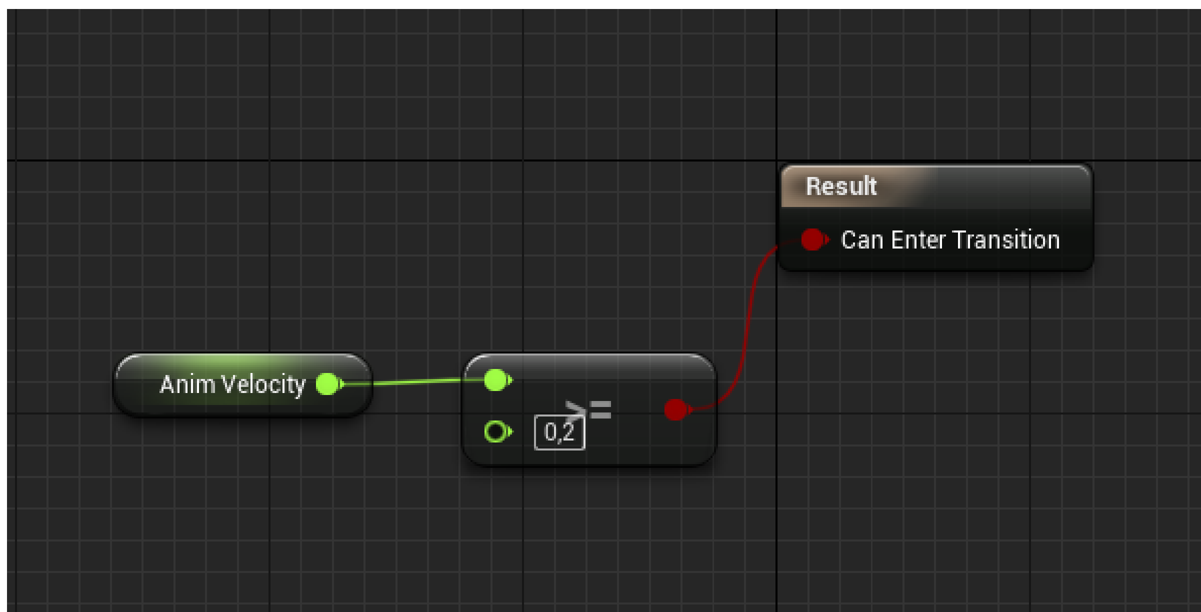

Рисунок 79. Логика перехода анимации.

В связи, **входящей** в анимацию **ThirdPersonIdle,** добавьте проверку значения меньше *0.1* переменной Anim Velocity (рисунок 80).

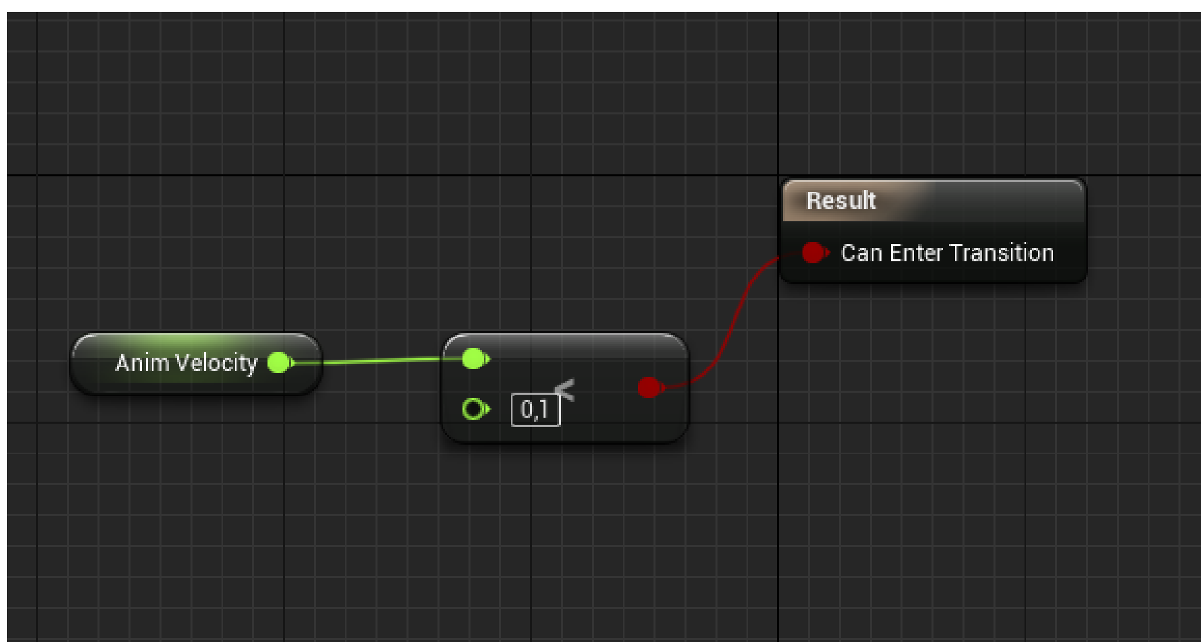

Рисунок 80. Логика перехода анимации.

В переменную Anim Velocity будем записывать значение скорости нашего 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. Нод Event Blueprint Update

Animation является аналогом Tick Update из обычных блюпринтов. Нод TryGetPawn Owner получает ссылку на объект, к которому назначен данный Animation Blueprint.

Добавьте переменную **Owner Ref** типа **Actor (Reference)** - в неё запишите ссылку на pawn через **Try Get Pawn Owner**.

В ноде **Update Animation** получите длину вектора **velocity** павна и присвойте её в переменную **Anim_Velocity** (рисунок 81).

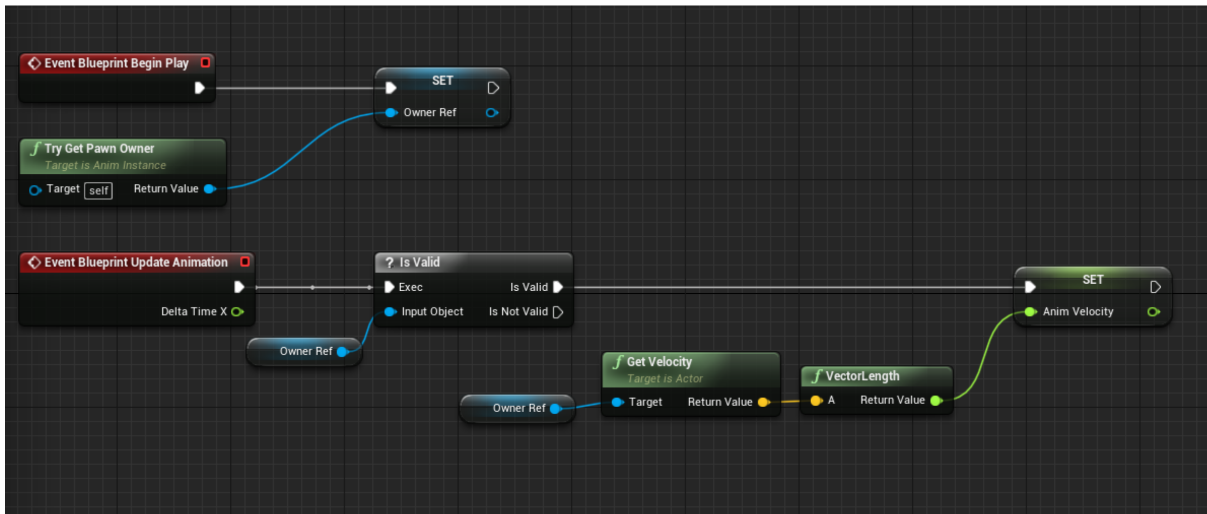

Рисунок 81

Скомпилируйте **Animation Blueprint** и присвойте к **ThirdPersonCharacter** (рисунок 82). Запустите игру и проверьте смену анимаций при движении персонажа.

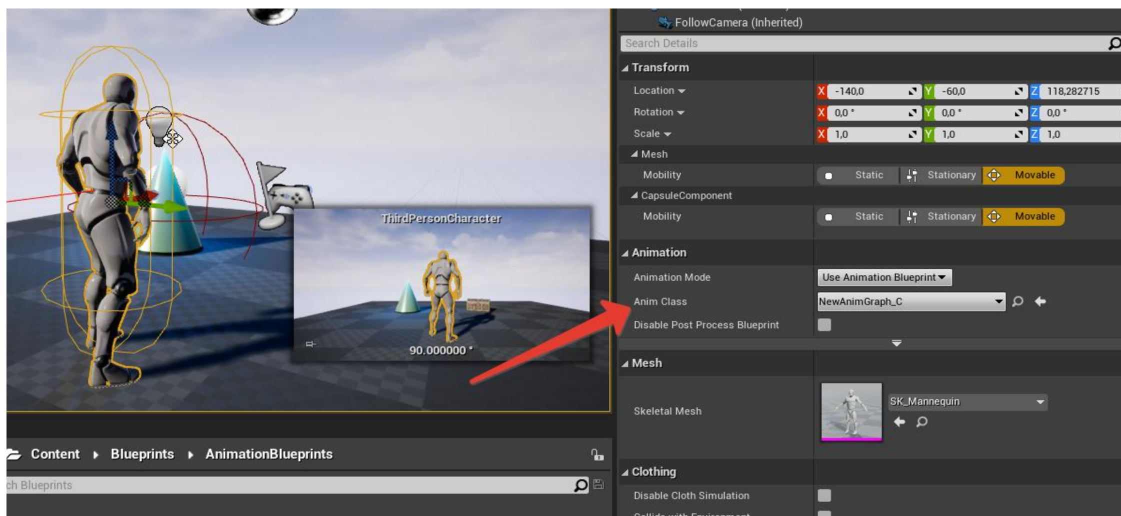

Рисунок 82. Выбрать Animation Blueprint можно в поле Animation, переменная AnimClass.

Занятие #6 Программирование на Blueprints. Часть III

6.1 Результат занятия

– уметь создавать Blueprint Interface и назначать его блюпринтам, вызывать методы Blueprint Interface;
– уметь наследовать блюпринт-классы;
– уметь подписываться на эвенты.

Интерфейс — это удобный способ обозначать методы, которые должны быть имплементированы у разных типов объектов. Для того, чтобы создать блюпринт интерфейс, добавьте новый ассет - выберите вкладку **Blueprints-> Blueprints Interface** (рисунок 83).

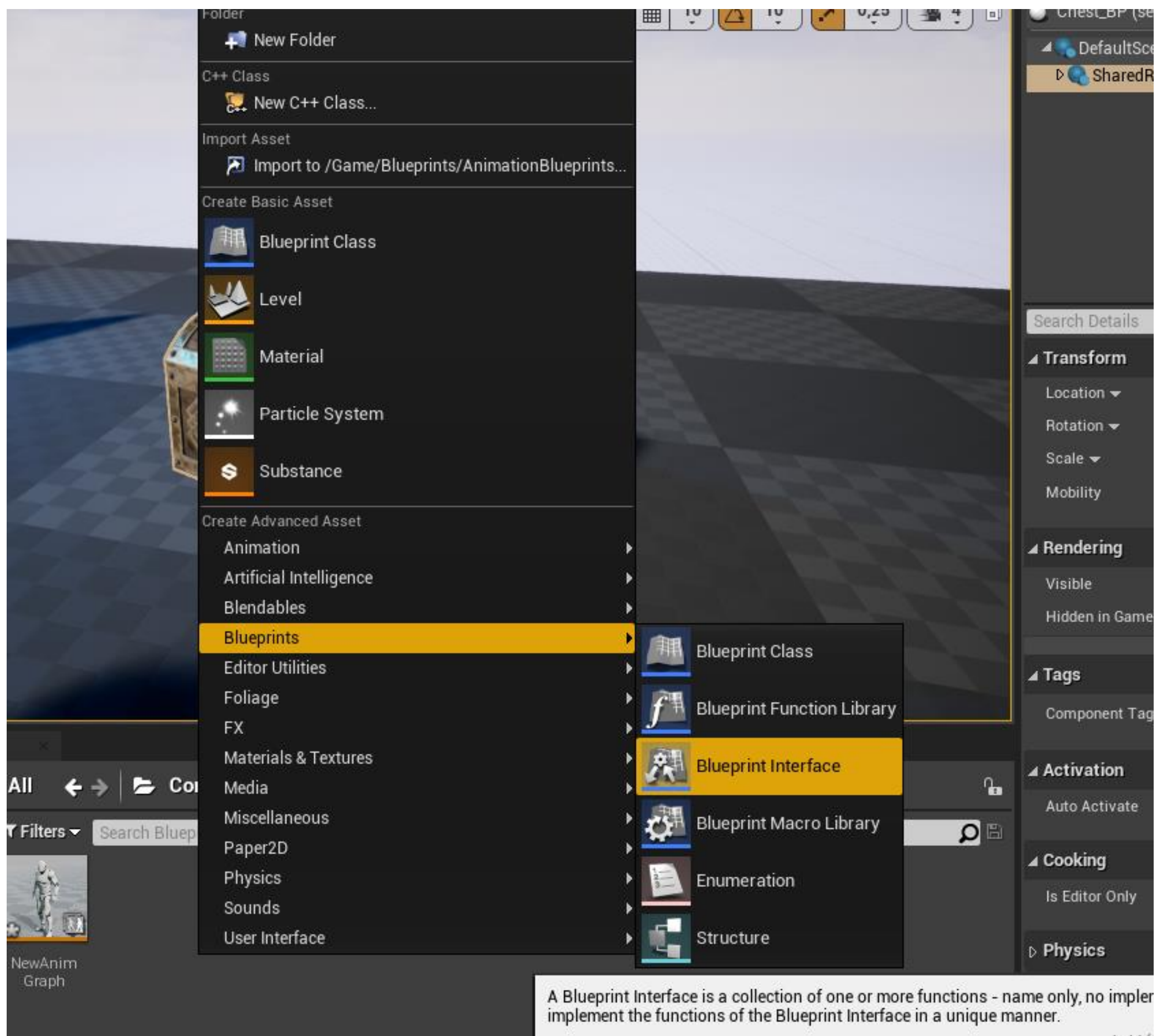

Рисунок 83. Путь создания интерфейса Blueprint.

Открыв **Interface Blueprint,** вы можете перечислить все методы, а также их входные и выходные аргументы, которые можно будет использовать в качестве интерфейса. (рисунок 84).

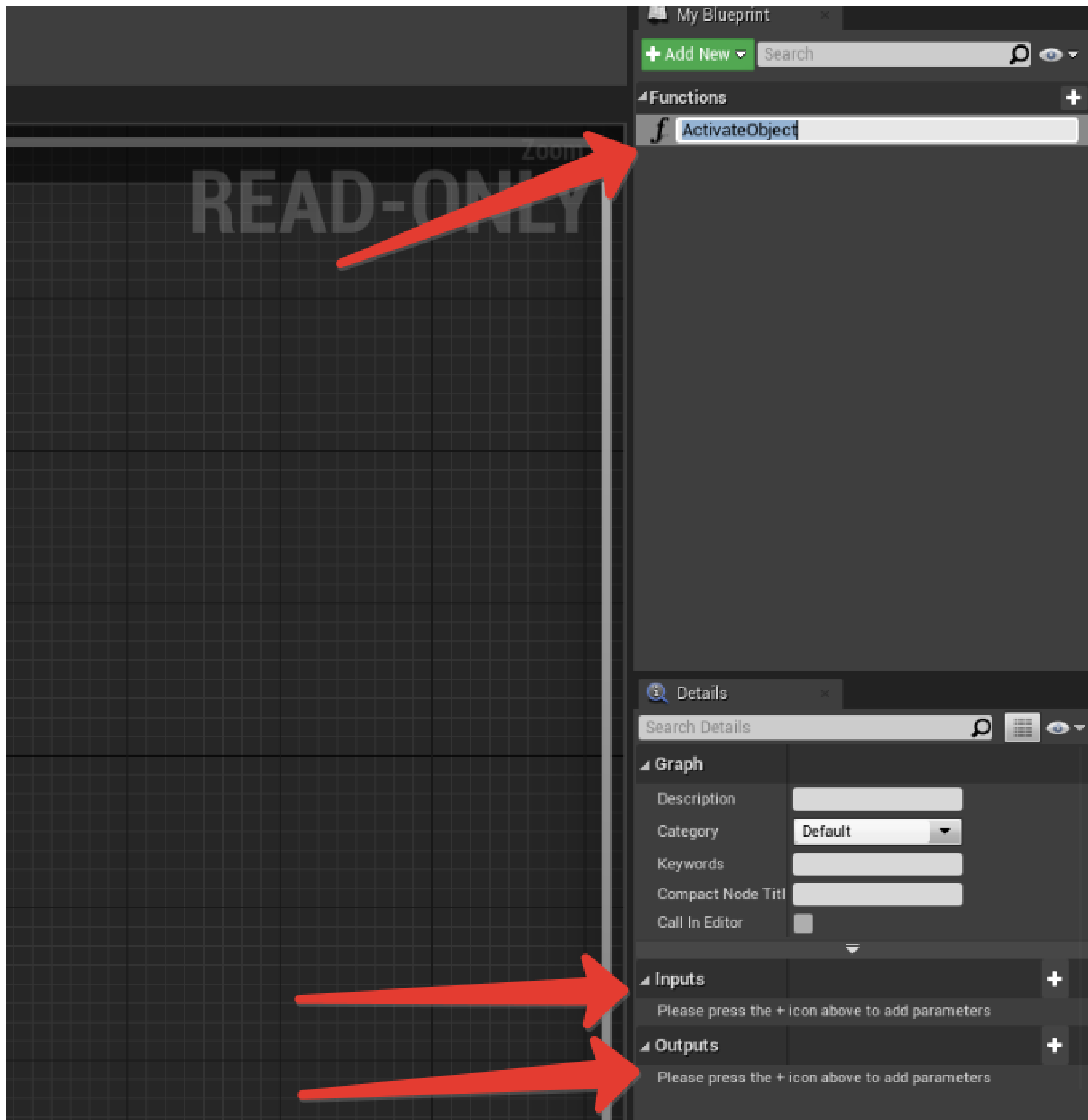

Рисунок 84. В интерфейсе вы можете так же, как и в функциях, объявлять входные и выходные аргументы.

Перейдите в блюпринт сундука, созданный в прошлом занятии. Нажмите на **Class Settings** и добавьте свой Interface Blueprint в список интерфейсов (рисунок 85). Обратите внимание, что ваш класс может содержать имплементацию нескольких интерфейсов.

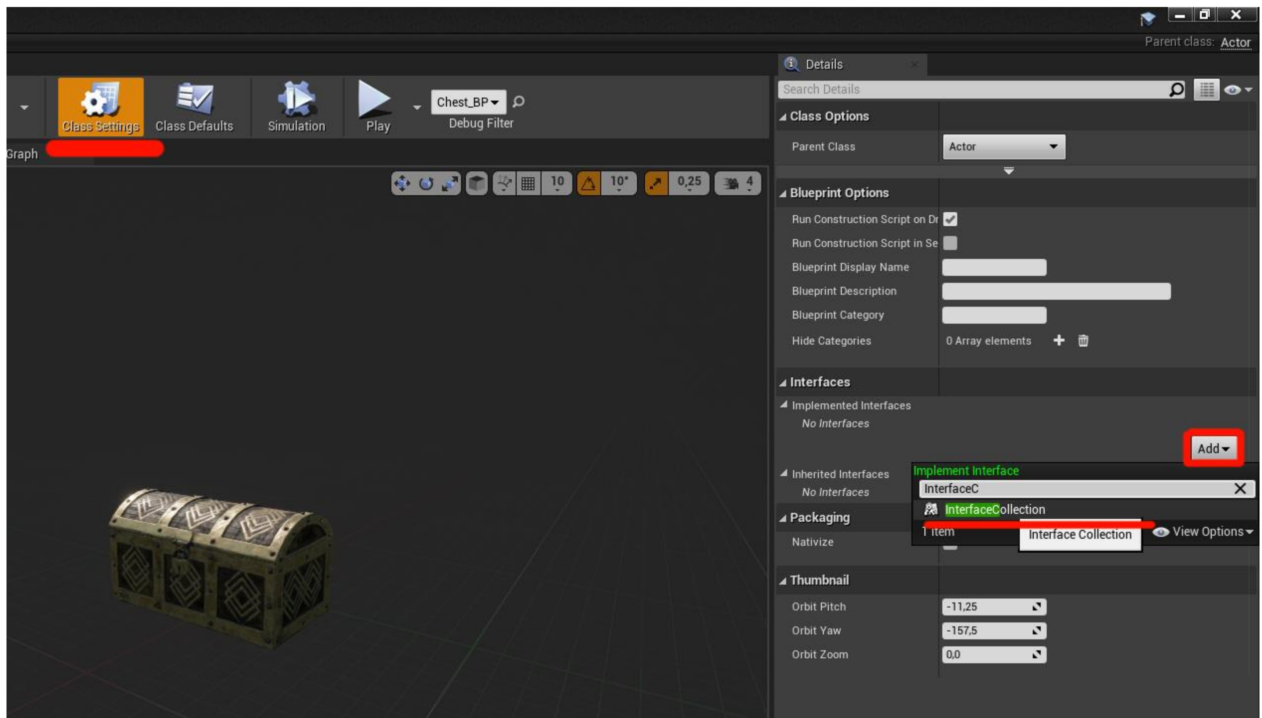

Рисунок 85. Добавление интерфейсов к классу происходит в меню Details при нажатии кнопки Class Settings.

Теперь вы можете реализовать события, описанные в Blueprint Interface, и вызывать их из другого блюпринта, не производя проверку типа блюпринта или не производя Cast (рисунок 86).

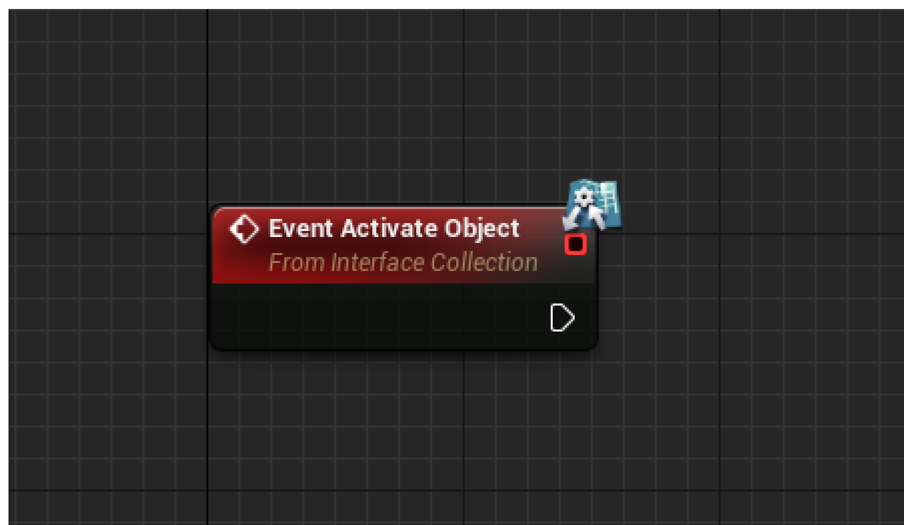

Рисунок 86. Так выглядит событие, созданное на основе интерфейса. Обратите внимание на отличительную иконку в углу нода.

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 на столкновение с сундуком из блюпринта персонажа. Используйте нод **Does Implement Interface! (рис)**
- в реализации интерфейса Activate Object запрограммируйте открытие крышки через нод Timeline (рисунок 87,88).

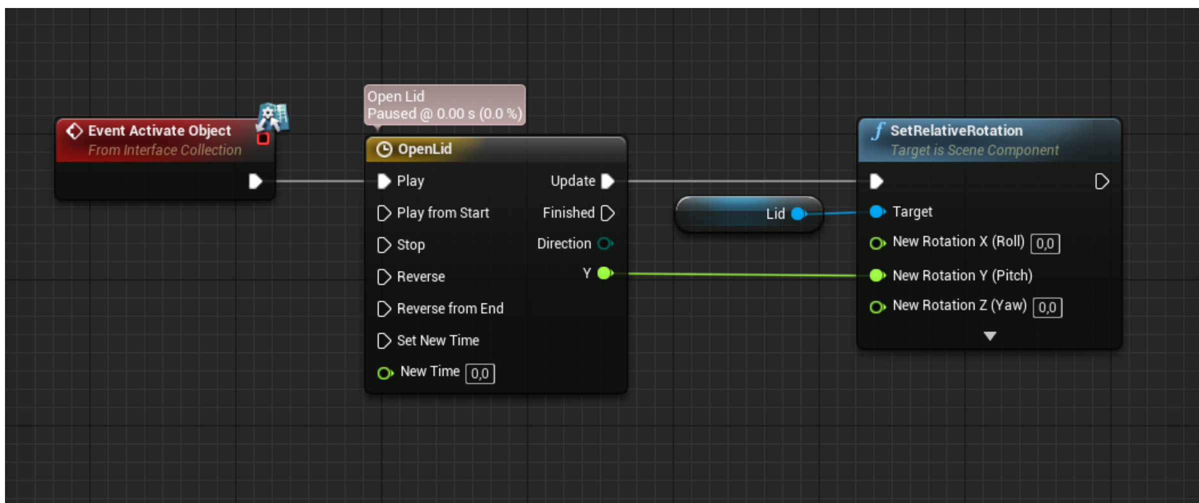

Рисунок 87. Таймлайн управляет углом поворота крышки сундука.

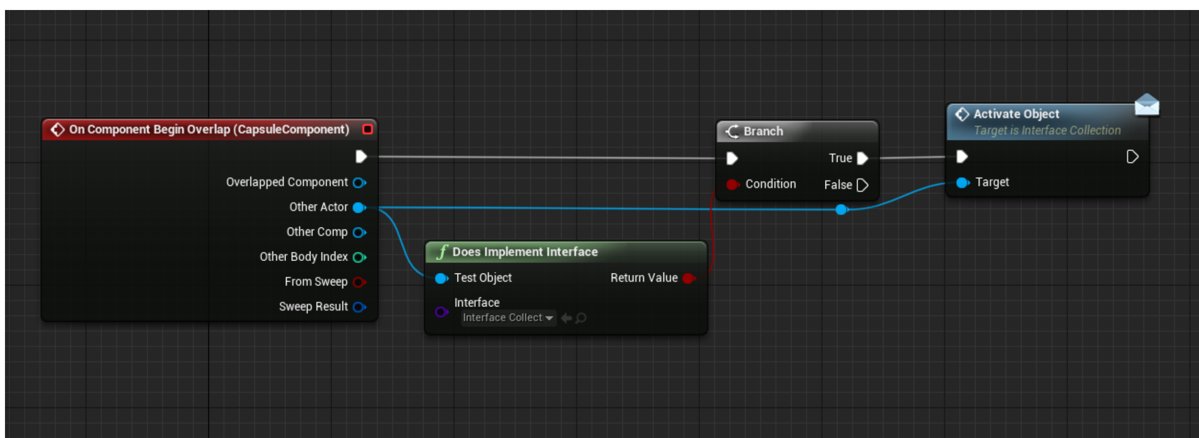

Рисунок 88. Нод Does Implement Interface проверяет объект на содержание определенного интерфейса. Заметьте, что с интерфейсом нет необходимости вызывать нод Cast To.

Обратите внимание, что вызов интерфейса не требует каста блюпринта в определенный тип. Таким образом, можно сильно упростить архитектуру кода вашей 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. Для того, чтобы создать дочерний блюпринт-класс, нажмите на желаемый блюпринт ПКМ и выберите **Create Child Blueprint Class** (рисунок 89).

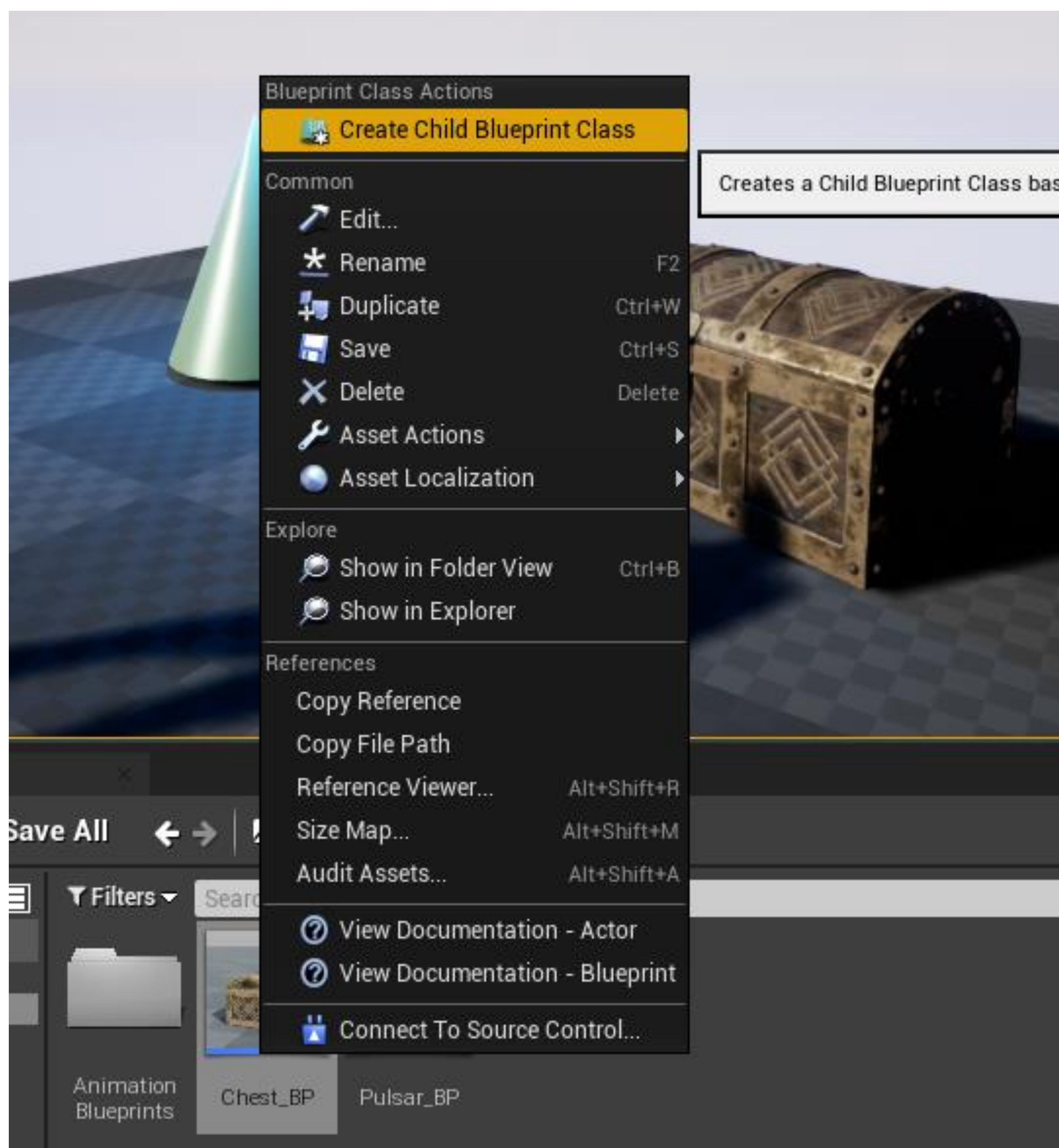

Рисунок 89. Создание дочернего класса блюпринта через опцию ***Create Child Blueprint Class.***

Чтобы переопределить функции родительского класса в дочернем блюпринте, используйте кнопку **override** в блоке Functions. (рисунок 90). Чтобы выполнить родительскую реализацию функции или события, щелкните ПКМ по событию в дочернем классе и выберете **Add Call to parent function.**

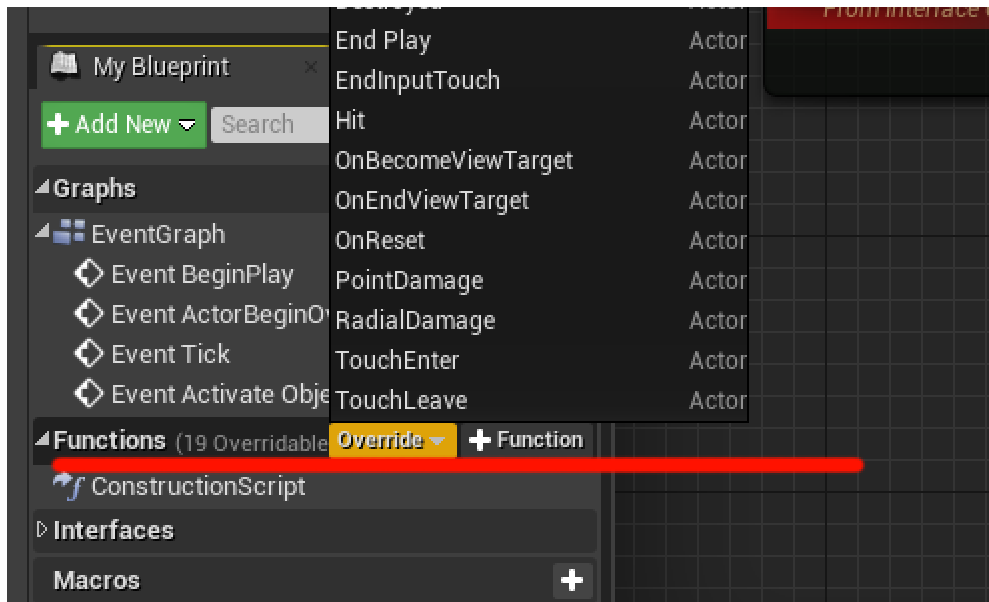

Рисунок 90. Чтобы просмотреть список событий и функций, которые можно переопределить в дочернем классе, нажмите на кнопку Override.

Функцию делегатов в системе Blueprints выполняют **Event Dispatchers** – подписав одно или более событие на **Event Dispatcher**, вы можете вызвать все эти события одновременно, как только ваш **Event Dispatcher** будет вызван. Добавьте **Event Dispatcher**, нажав на + в разделе **Event Dispatchers** (рисунок 91).

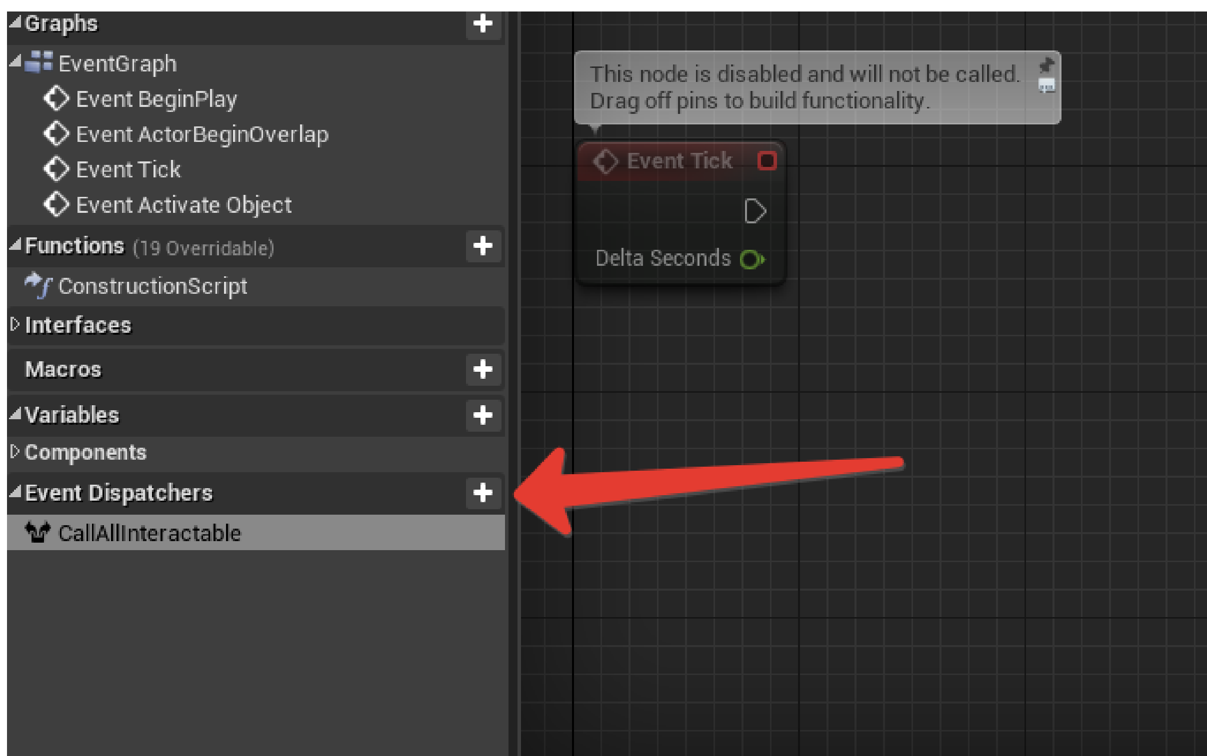

Рисунок 91. Вы можете иметь неограниченное число Event Dispatchers.

Добавьте нод вызова диспетчера по нажатию на клавишу E (рисунок 92). Затем подпишитесь на данный диспетчер из блюпринта Chest_BP (рисунок 93).

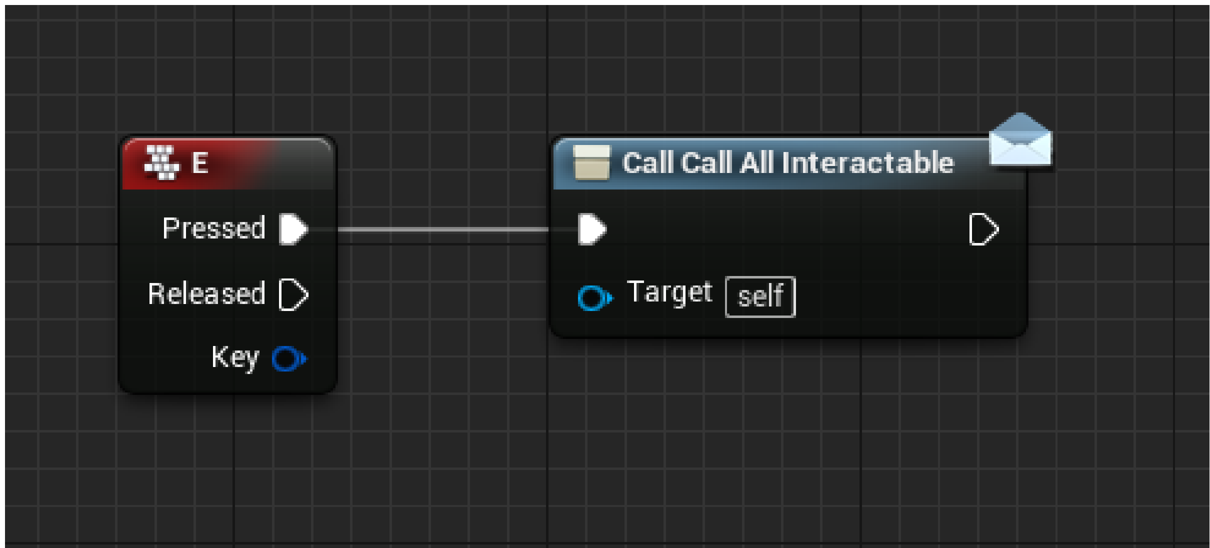

Рисунок 92. Нод E вызывается при нажатии на клавишу клавиатуры латинской E, происходит вызов диспетчера.

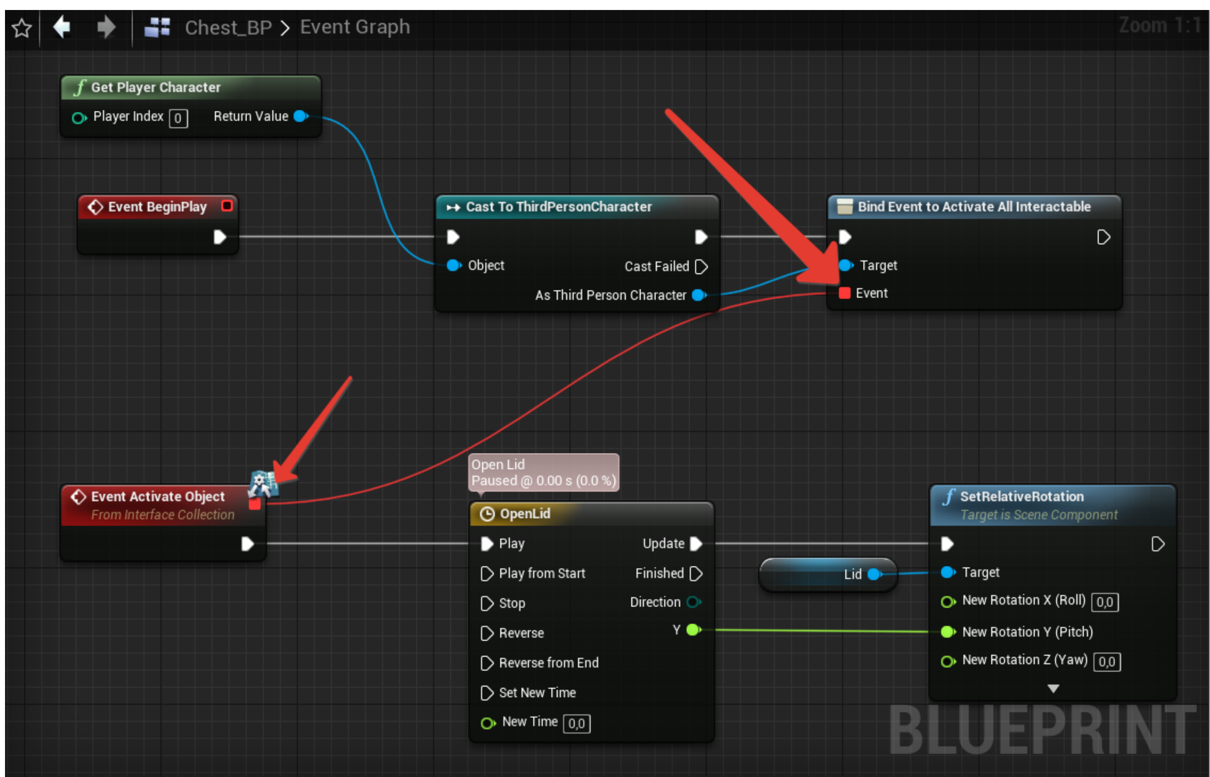

*Рисунок 93. Вы можете подписывать ваши события соединяя их красным узлом с нодом Bind Event to *имя диспетчера*.*

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

7.1 Результаты занятия

- уметь создавать NavMesh;
- уметь использовать ноды AI;
- уметь создавать Behaviour Tree и Blackboard;
- уметь создавать патрулирующего персонажа, который останавливается на каждой точке патруля.
- уметь создавать преследующего игрока персонажа.

7.2 Теоретические сведения

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

- Как AI обрабатывает задачи
- Как AI получает или воспринимает информацию о мире
- Как AI интерпретирует геометрическое пространство мира
- Как AI пытается повторять поведение, приближенное к человеческому поведению

Главным компонентом любого игрового AI является формальная система принятия решений - по сути мозг AI. Этот “мозг” выполняет определенные действия, оценивая окружающую обстановку.

Основные способы реализации системы принятия решений:

1. Конечные Автоматы (Finite-State machine)
2. Поведенческие деревья (Behavior tree)
3. Кривые выгоды (Utility AI)

Конечные Автоматы (FSM)

Конечный автомат — это некоторая абстрактная модель, содержащая конечное число состояний чего-либо. Программист задает набор состояний и переходов между ними. Самый проверенный временем способ создании ИИ.

Поведенческие деревья (Behaviour Trees)

Дальнейшее развитие идеи Конечных Автоматов. Основная идея такая же - 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 вы можете реализовать довольно просто свою систему FSM для AI, либо использовать стандартный инструментарий для написания Behaviour Tree. В ситуации, когда планируется, что AI в вашей игре будет комплексным, рекомендуется использовать Behaviour Trees (рисунок 94).

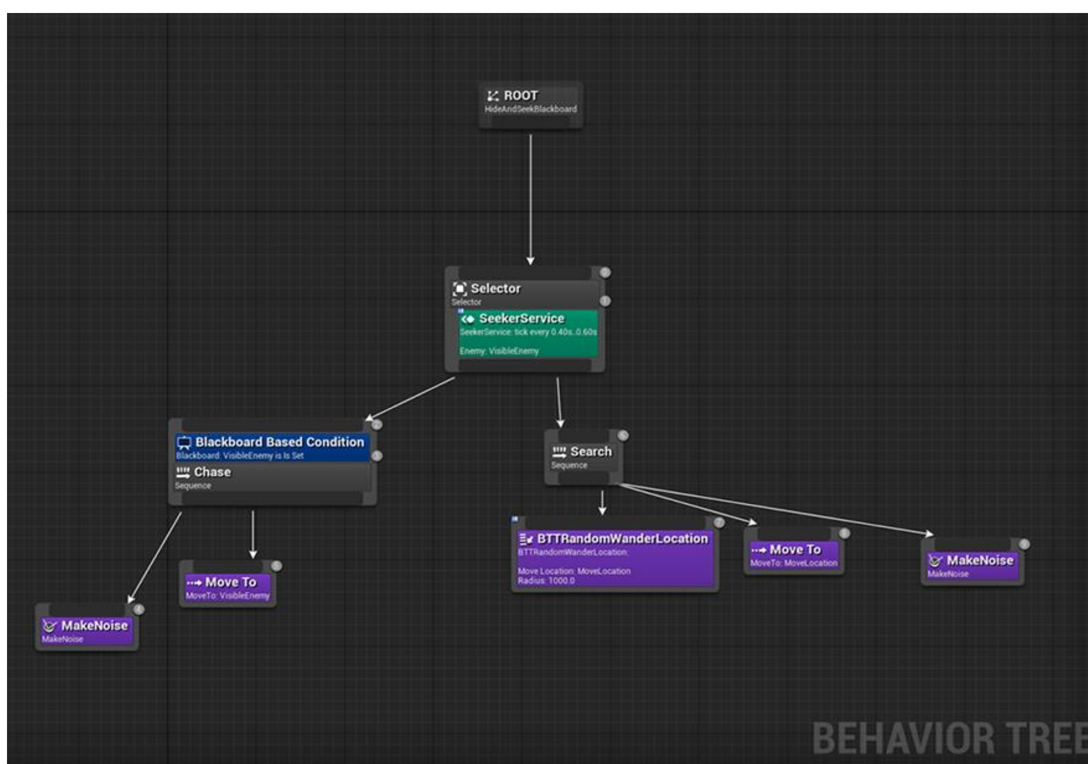

Рисунок 94. Пример дерева поведения в Unreal Engine 4.

Root нод - уникальный нод дерева поведения. Начальная точка. К нему нельзя прикрепить декораторы и сервисы (рисунок 95).

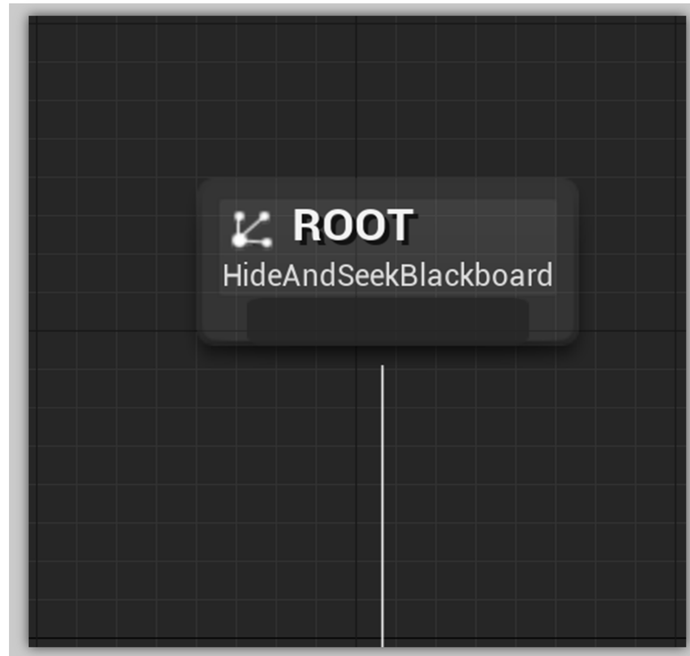

Рисунок 95. Нод Root.

Task нод - листья дерева. Они сообщают, что “делать”, и не имеют выходных соединений (рисунок 96).

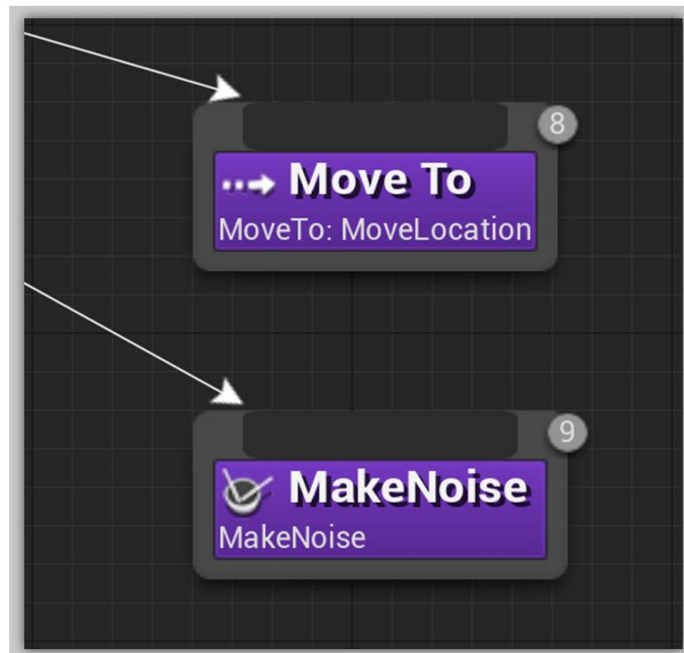

Рисунок 96.Нод Task.

Sequence нод - этот узел выполняет своих “детей” слева-направо и останавливает выполнение цепочки, если один из детей не может выполниться (рисунок 97).

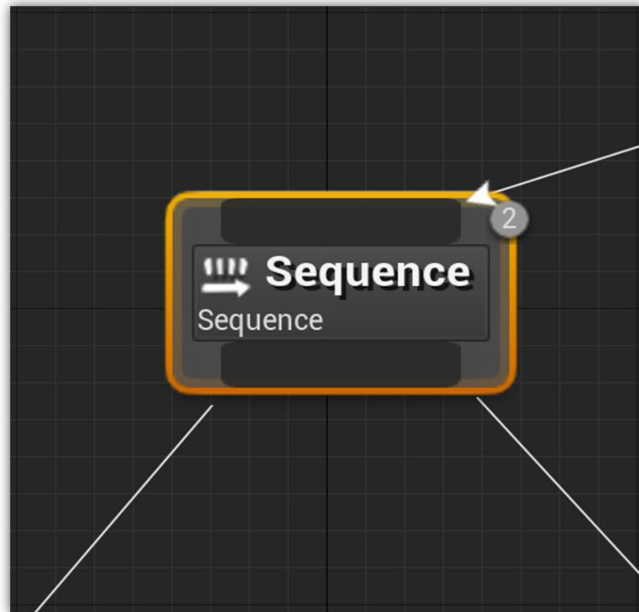

Рисунок 97. Нод Sequence.

Selector нод - 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 98).

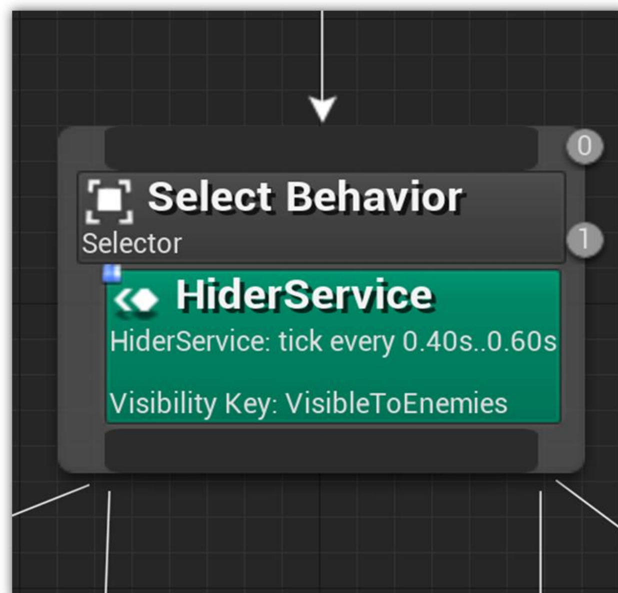

Рисунок 98. Нод Selector.

Decorators нод - то же самое, что и условия. Они присоединяются к другим нодам и решают, будет ли выполняться ветка или лист дерева (рисунок 99).

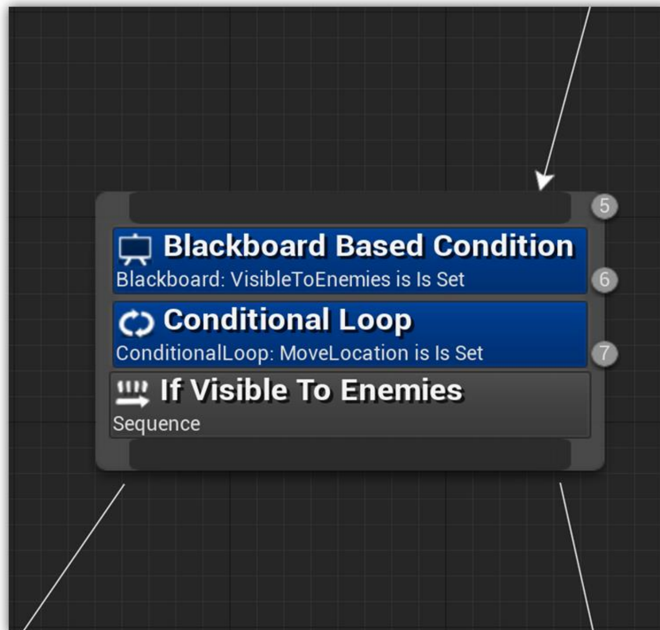

Рисунок 99. Нод Decorator.

Observers - это свойства декоратора, через которые можно связываться со значениями из Blackboard (*блекборда*). Также в *Обзерверах* указываются условия прекращения выполнения узла или ветки дерева (рисунок 100).

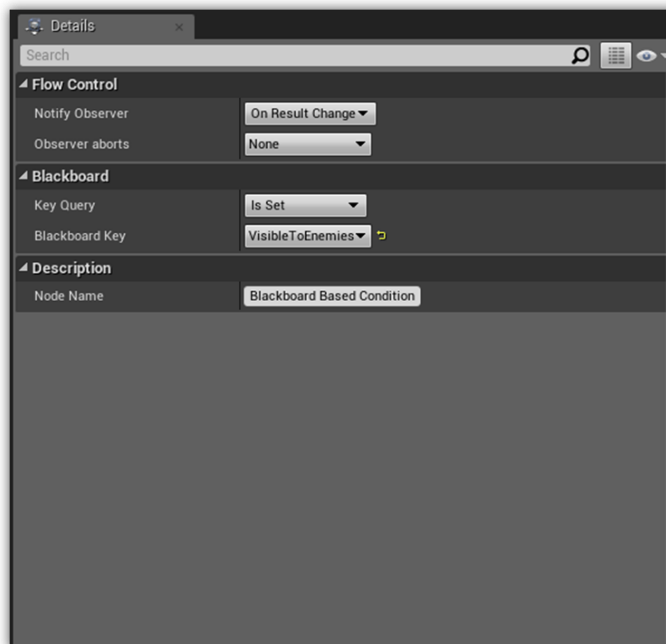

Рисунок 100. Details панель Blackboard с списком Observers.

Blackboards - память AI. Таблица, которая хранит ключи и значения, которые используются деревом поведения (рисунок 101).

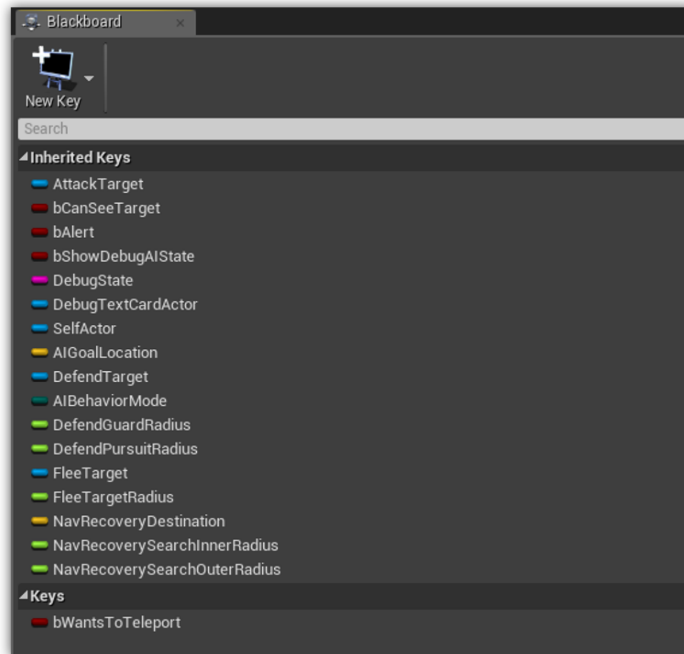

Рисунок 101. Примерный вид Blackboard.

Services - это модуль, присоединяемый к нодам. Он сообщает частоту выполнения какого-либо нода, если этот нод вообще выполняется. Они нужны для обновления проверок значений или обновления значений в Blackboard (рисунок 102).

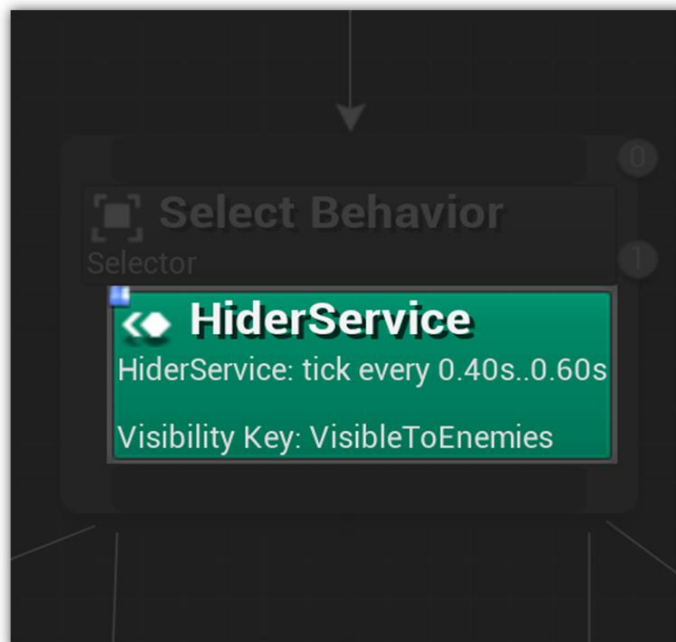

Рисунок 102. Нод Services.

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 актор Nav Mesh Bounds Volume на уровень (рисунок 103). Вы можете изменить масштаб актора, чтобы покрыть нужную площадь уровня.

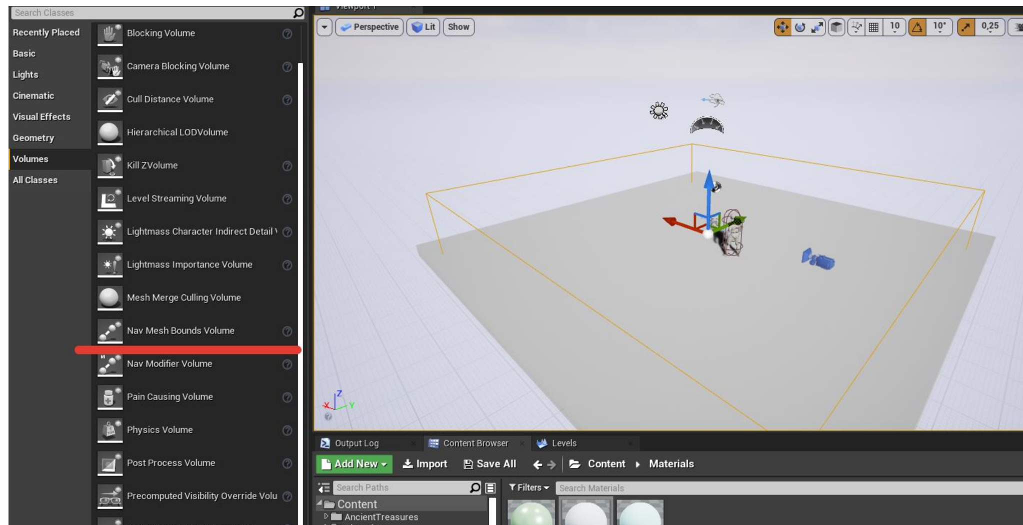

Рисунок 103. Вы можете видеть границы NavMesh Bounds Volume в окне Viewport.

Откройте пункт Build и выберете 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 104).

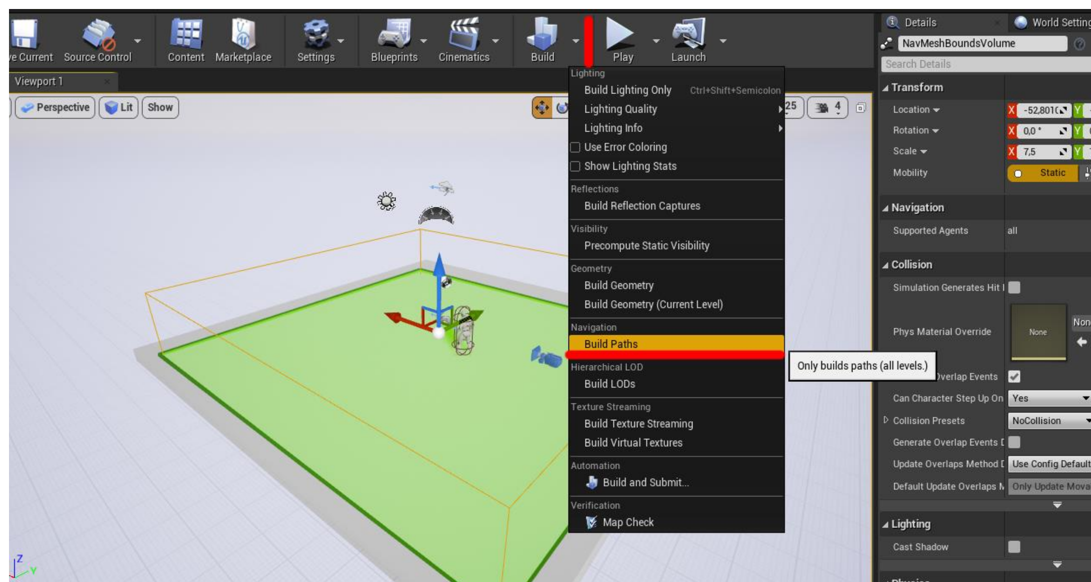

Рисунок 104. В выпадающем списке рядом с кнопкой Build, выберете опцию Build Paths.

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. А если у вас игра про морские бои, где корабли управляются AI, то логичнее использовать Pawn актора.

Добавьте Character Actor, назначьте ему скелетал меш (в данном примере стандартный манекен) и превратите его в Blueprint класс. Откройте созданный блюпринт (рисунок 105).

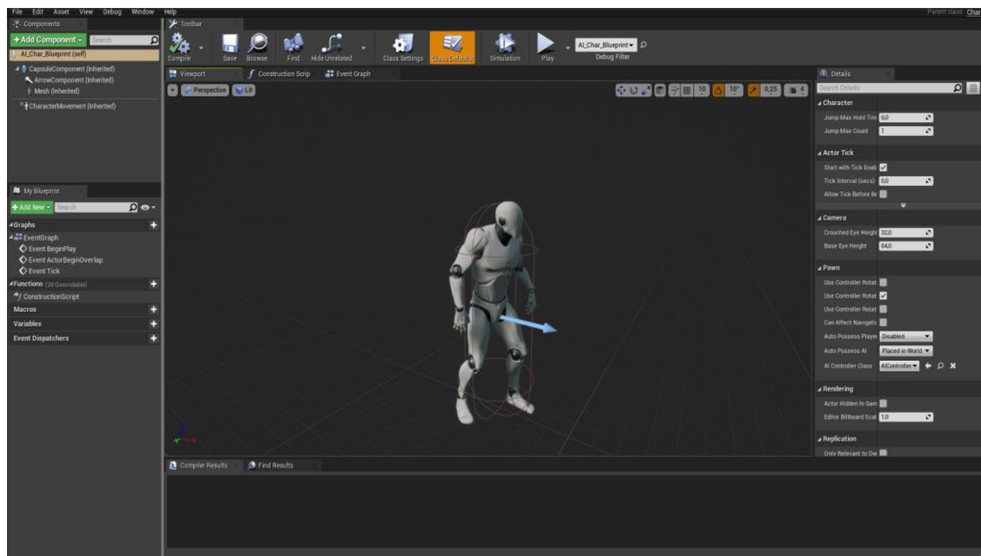

Рисунок 105. Созданный блюпринт класс персонажа.

Используя нод **AI Move To**, создайте код, который заставляет агента преследовать игрока (рисунок 106).

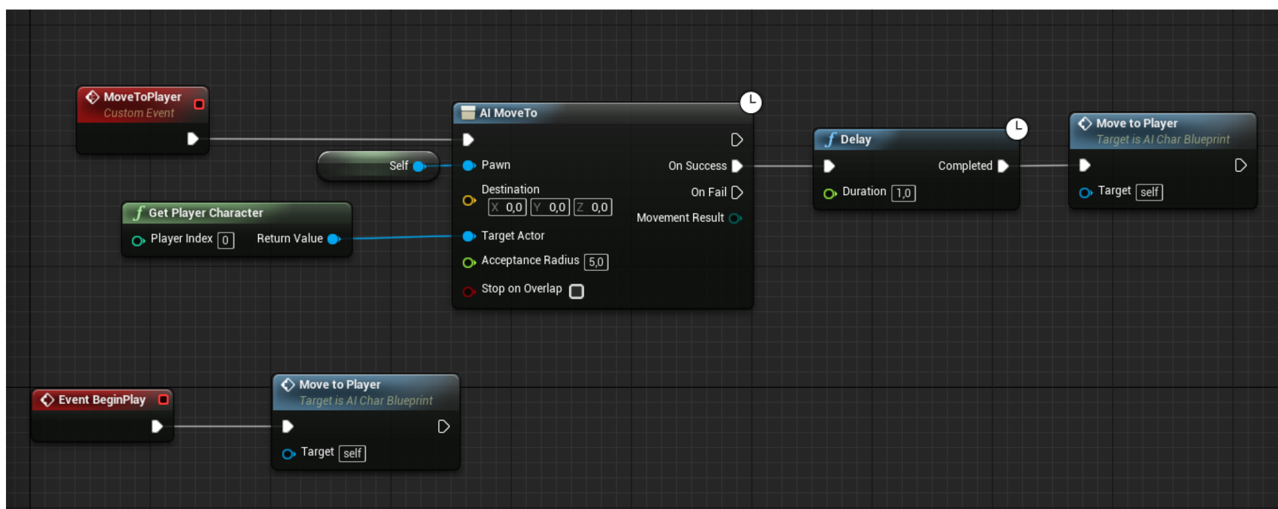

Рисунок 106. Нод AI Move To, заставляет блюпринт перемещаться к нужной позиции, если она достижима на существующем навигационном меше.

P.S. 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** (рисунок 107).

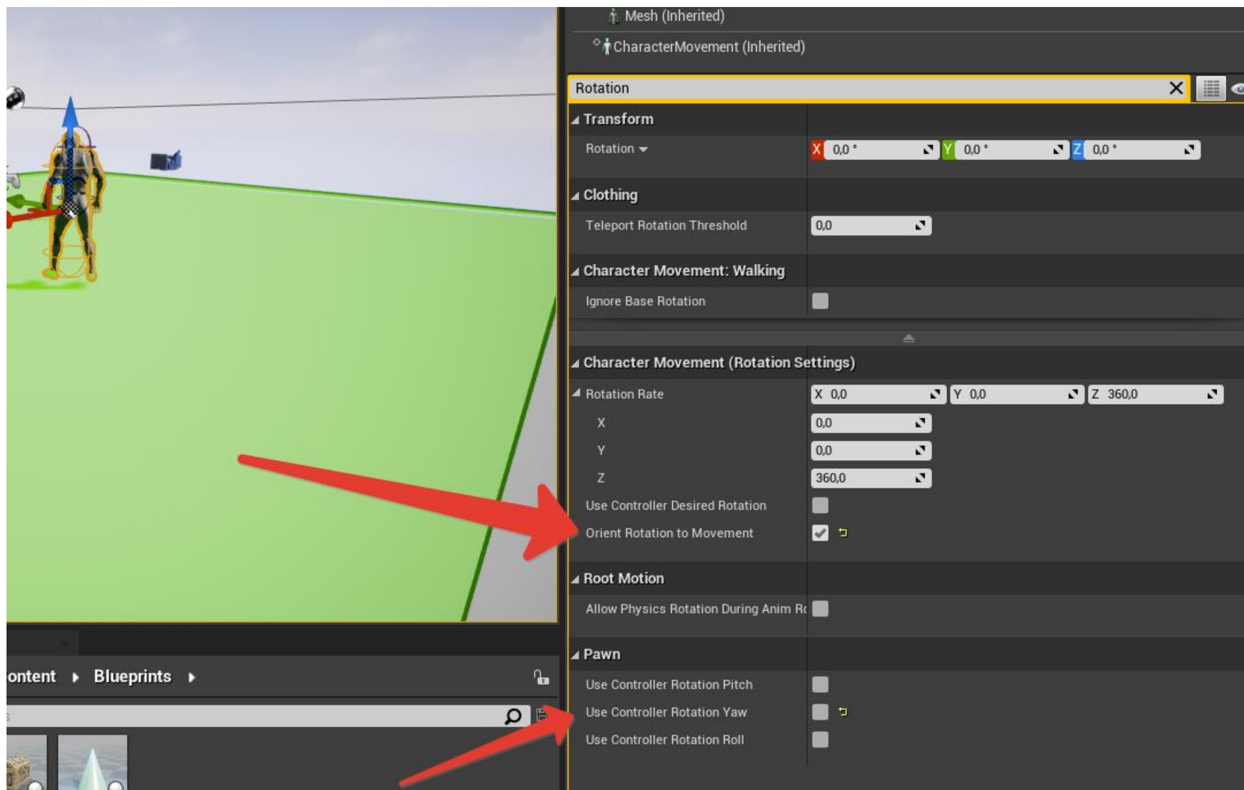

Рисунок 107. Флаги, на которые стоит обратить внимание, чтобы персонаж плавно поворачивался в направлении движения.

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

Напишем 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 108).

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 109). 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.

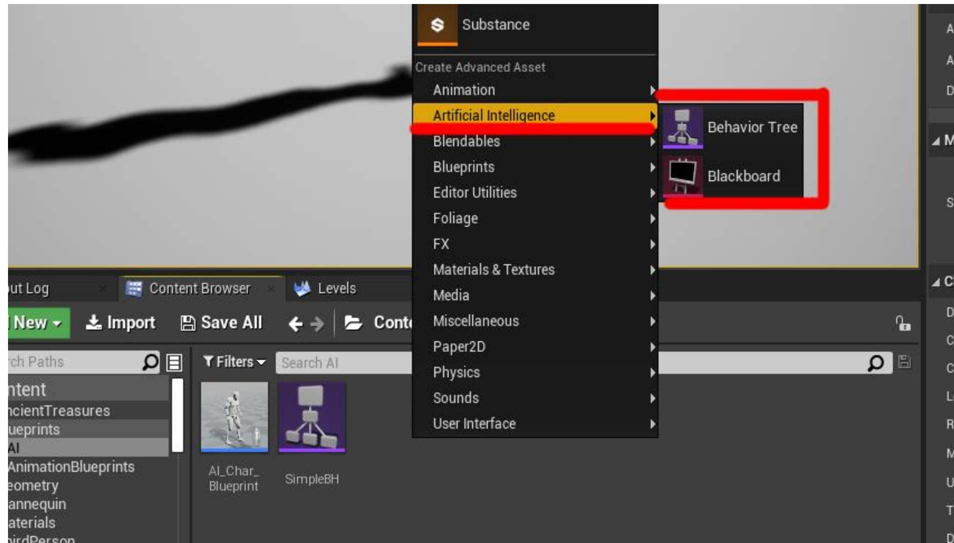

Рисунок 108. Создайте ассем Behaviour Tree и Blackboard.

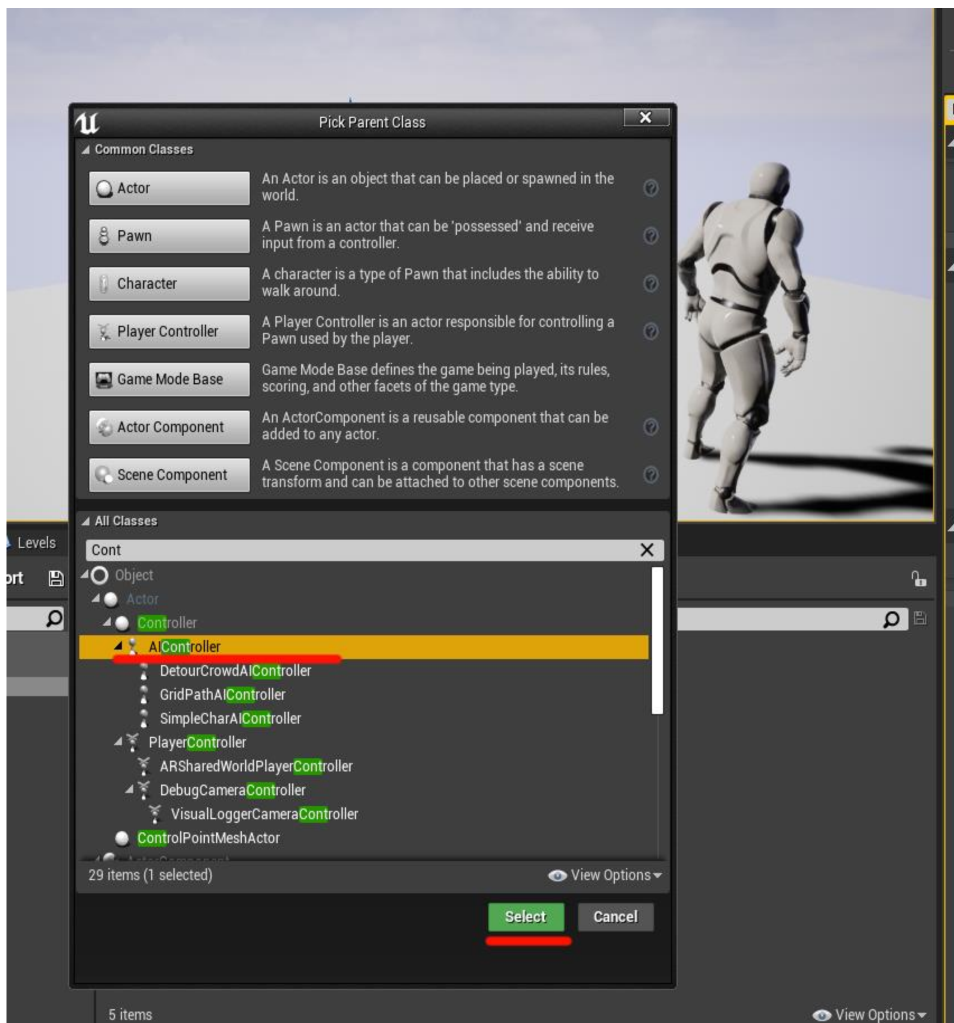

Рисунок 109. Создайте Blueprint класс, наследуемый от AIController.

Добавьте созданный контроллер класс в раздел Pawn вашего агента (рисунок

110).

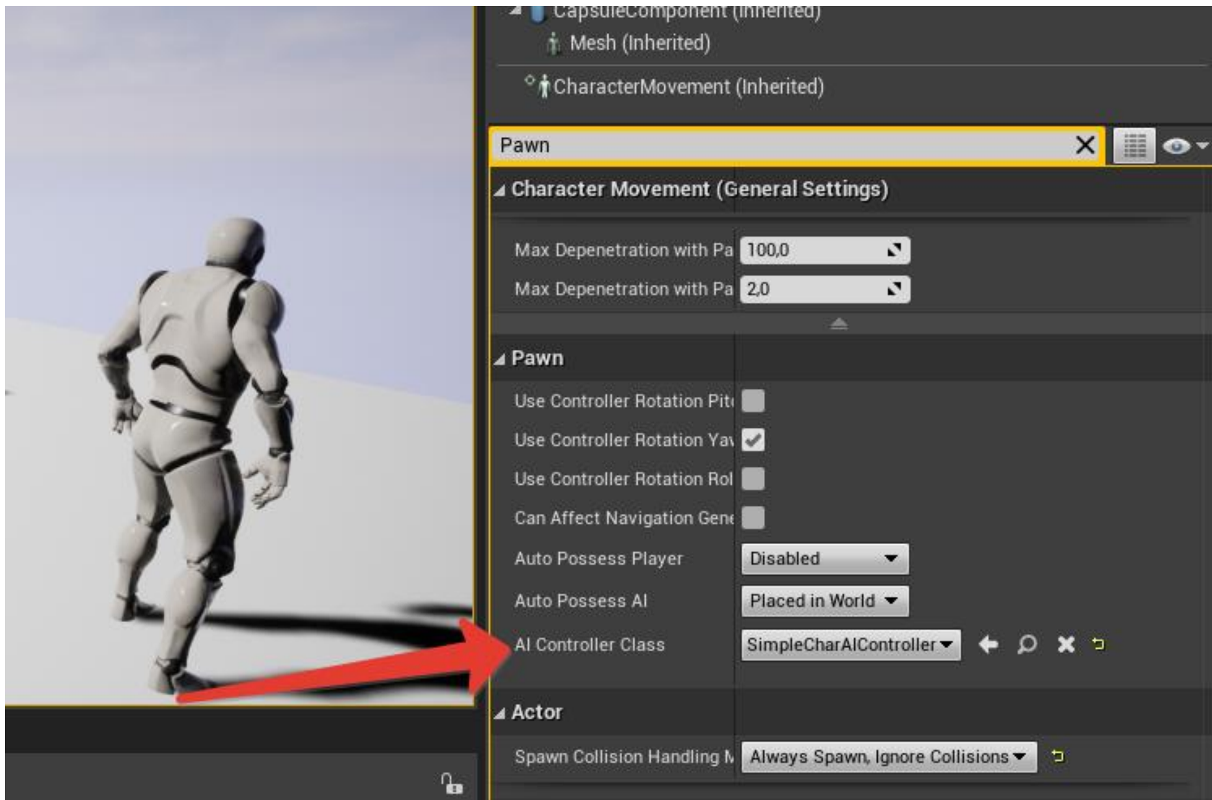

Рисунок 110. Используйте строку поиска, чтобы найти переменную AI Controller Class в панели Details.

Откройте **Event Graph** созданного и присвоенного контроллера. Нод **Run Behavior Tree** запускает выполнение дерева поведения. Нод **Use Blackboard** подключает использование конкретного блэкборда (рисунок 111).

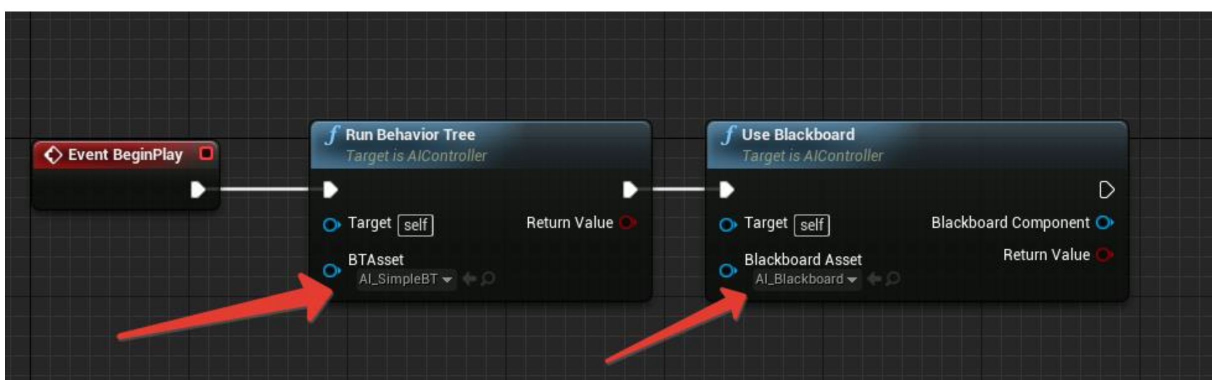

Рисунок 111. Нод Run Behaviour Tree создает экземпляр дерева поведения и запускает его, нод Use Blackboard включает использование Blackboards.

Откройте ваш **Behaviour Tree** и добавьте следующие ноды (рисунок 112): нод **Sequence** и от него ноды **MoveTo** и нод **Wait**. Ноды добавляются ПКМ. Важно помнить, что ноды выполняются **сверху-вниз**, **слева-направо**. У данных нодов

есть стандартные аргументы. У нода **MoveTo** - это переменная из Blackboard к которой нужно двигаться (Vector позиции или ссылка на объект). У нода **Wait** это количество секунд ожидания.

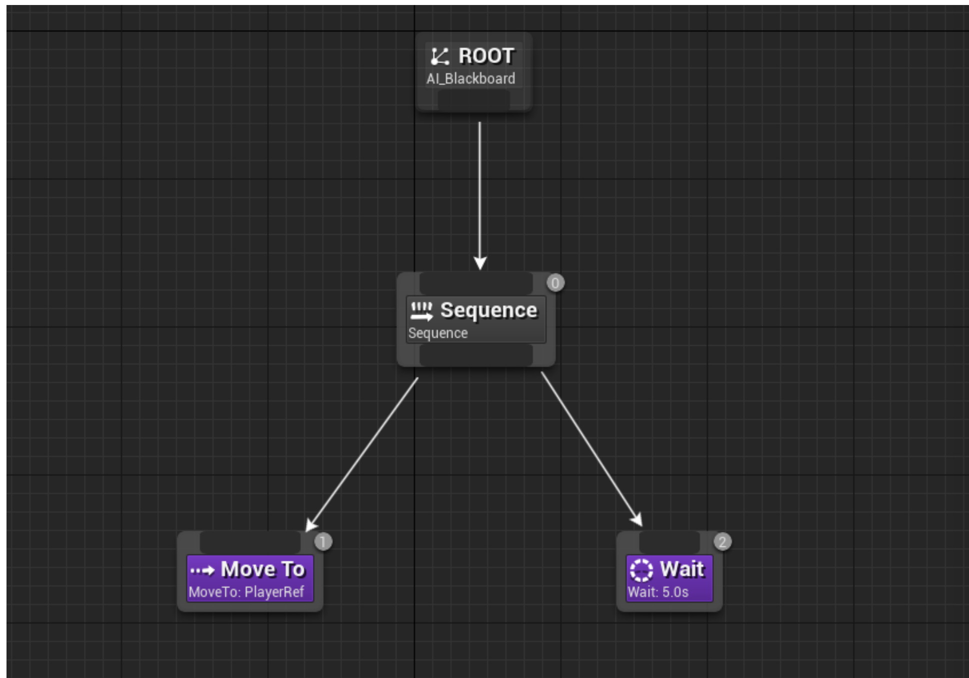

Рисунок 112. Простое дерево поведения.

Откройте созданный Blackboard, добавьте переменную, в которой будет храниться ссылка на игрового персонажа. Тип переменной Object, класс Actor. (рисунок 113, 114). Обратите внимание на флаг Instance Synced. Включенный флаг означает, что значение данной переменной будет одно во всех инстансах данного блэкбоарда. В случае с ссылкой на игрового персонажа логично поставить данный флаг в true.

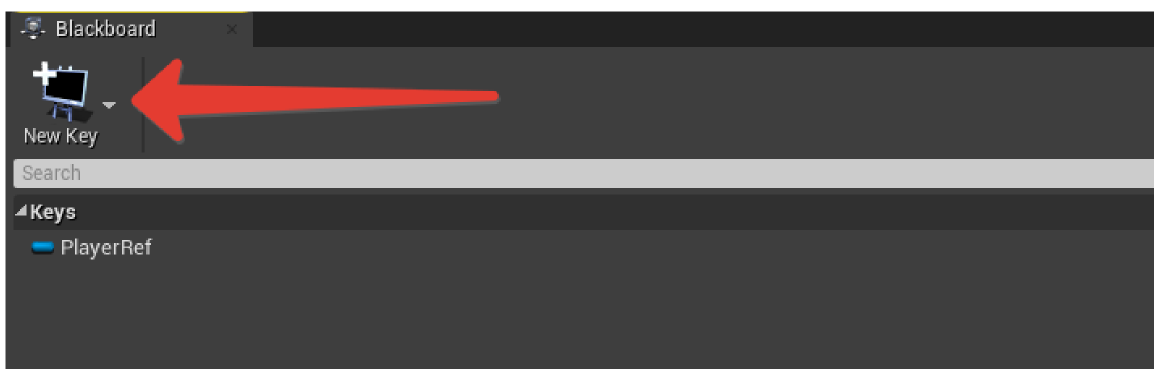

Рисунок 113. Используйте кнопку New Key, чтобы добавлять новые переменные в Blackboard.

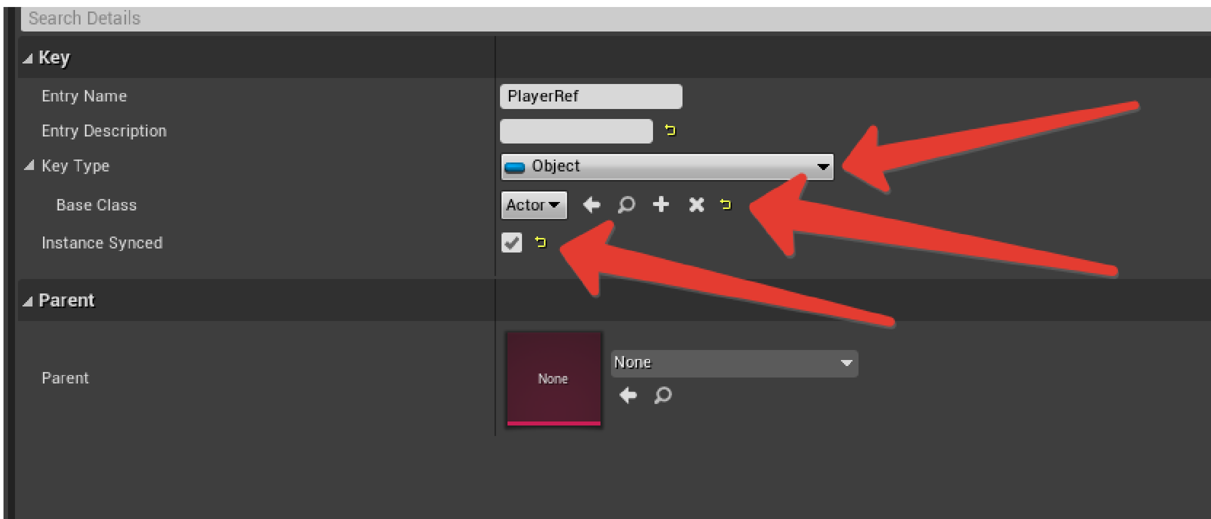

Рисунок 114. Выделив созданную переменную, вы можете изменять её характеристики, например, тип данных. В данном примере тип данных Object с Base Class типа Actor.

Проверьте подставленное значение в **Behaviour Tree** нода **MoveTo** (по-умолчанию первая переменная blackboard ставится автоматически во все пустые аргументы нодов дерева) (рисунок 115).

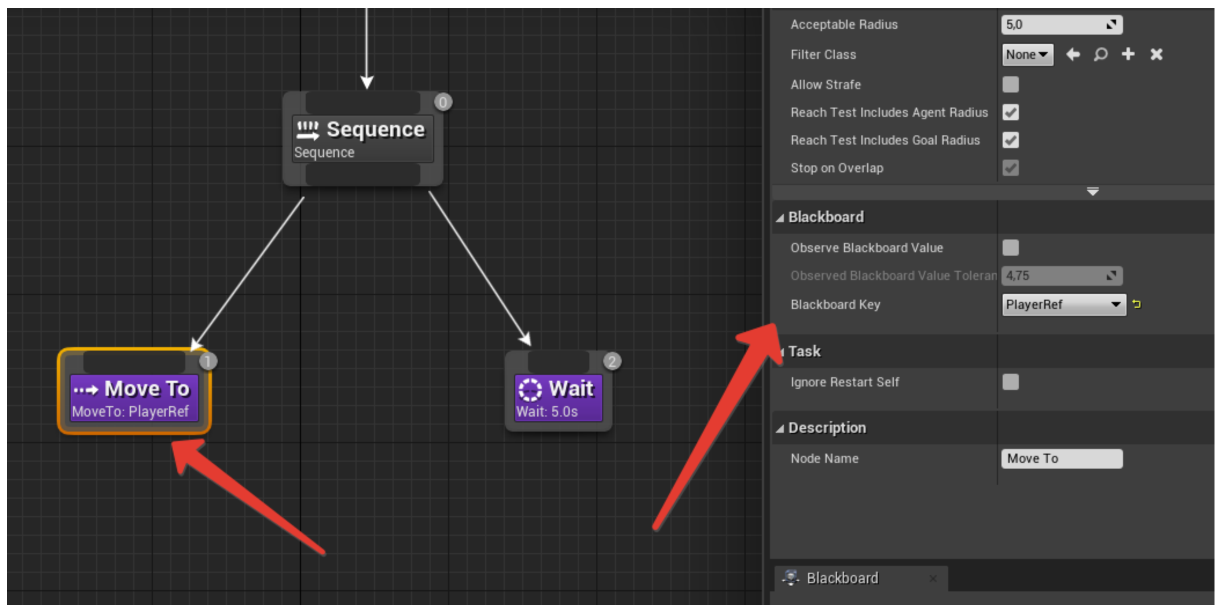

Рисунок 115. Выбрав нод Move To в его параметрах вы должны указать переменную из Blackboard, которая будет использоваться нодом. В данном примере это переменная Player Ref.

Чтобы установить значение в переменную Blackboard, используется нод **Set Value as *тип переменной***. Используйте нод **Make Literal Name**, чтобы указать имя ключа переменной из Blackboard (рисунок 116). Данные ноды можно писать как в обычном блюпринте, так и в Controller блюпринте.

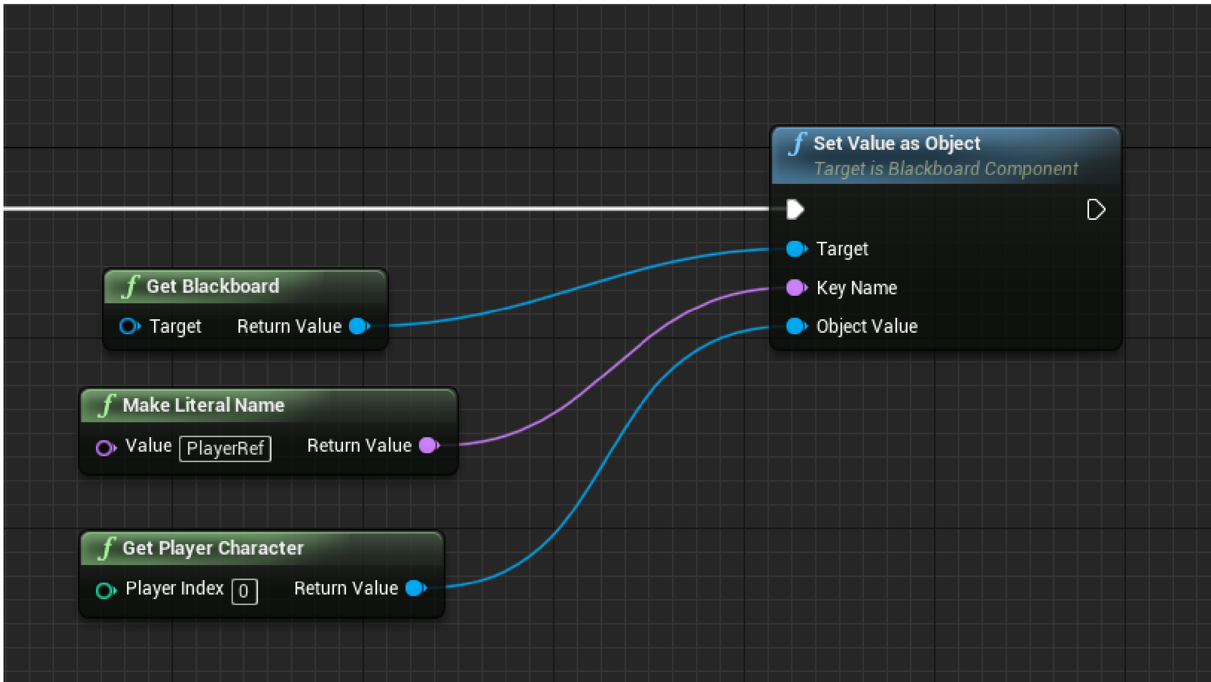

Рисунок 116

При запуске проекта агент будет следовать за игроком и при достижении цели ждать 5 секунд, затем повторять дерево. Вы можете проследить выполнение дерева, открыв его, при запущенной игры (рисунок 117).

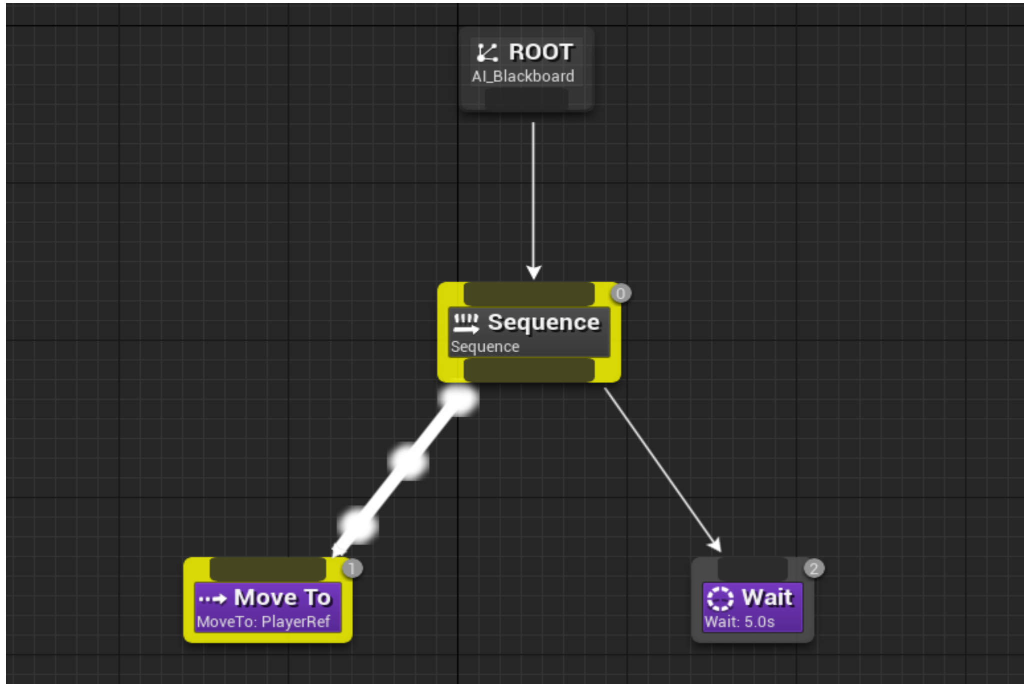

Рисунок 117. Желтой обводкой выделяется нод дерева поведения, который выполняется в настоящий момент времен.

Загарских Александр Сергеевич
Хорошавин Александр Александрович
Александров Эдуард Эмильевич

ВВЕДЕНИЕ В РАЗРАБОТКУ КОМПЬЮТЕРНЫХ ИГР

Учебно-методическое пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49