Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет геосистем и технологий»
(СГУГиТ)

Н. С. Головачев, П. Ю. Бугаков

ТЕХНОЛОГИИ ТРЕХМЕРНОГО МОДЕЛИРОВАНИЯ И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

РАЗРАБОТКА ИНТЕРАКТИВНЫХ ТРЕХМЕРНЫХ СЦЕН

Утверждено редакционно-издательским советом университета
в качестве практикума для обучающихся по направлению подготовки
09.03.02 Информационные системы и технологии
(уровень бакалавриата)

Новосибирск
СГУГиТ
2026

УДК 004.946(075)
Г60

Рецензенты: кандидат технических наук, доцент СГУПС *А. А. Уланов*
кандидат технических наук, доцент, СГУГиТ *А. А. Колесников*

Головачев, Н. С.

Г60 Технологии трехмерного моделирования и виртуальной реальности. Разработка интерактивных трехмерных сцен : практикум / Н. С. Головачев, П. Ю. Бугаков. – Новосибирск : СГУГиТ, 2026. – 103 с. – Текст : непосредственный.

ISBN 978-5-907998-70-4

Практикум подготовлен ассистентом Н. С. Головачевым и кандидатом технических наук, доцентом П. Ю. Бугаковым на кафедре прикладной информатики и информационных систем СГУГиТ.

Издание включает лабораторные работы, охватывающие ключевые разделы дисциплины «Технологии трехмерного моделирования и виртуальной реальности»: основы разработки интерактивных 3D-приложений в Unreal Engine; создание и настройку пользовательского персонажа; реализацию управления и взаимодействия с объектами; разработку интерактивных элементов с использованием Blueprint, коллизий и виджетов; создание виртуального персонажа с базовым искусственным интеллектом и организацию его перемещения; а также разработку, настройку и тестирование VR-приложений.

Практикум по дисциплине «Технологии трехмерного моделирования и виртуальной реальности» предназначен для обучающихся по направлению подготовки 09.03.02 Информационные системы и технологии (уровень бакалавриата).

Рекомендован к изданию кафедрой прикладной информатики и информационных систем, Ученым советом Института геодезии и менеджмента СГУГиТ.

Печатается по решению редакционно-издательского совета СГУГиТ

Ответственный редактор: кандидат технических наук, доцент, СГУГиТ
С. Ю. Кацко

УДК 004.946(075)

ISBN 978-5-907998-70-4 © СГУГиТ, 2026

СОДЕРЖАНИЕ

Введение ... 4
Лабораторная работа № 1. Настройка системы интерактивного взаимодействия с пользователем в среде Unreal Engine ... 6
Лабораторная работа № 2. Создание интерактивных областей взаимодействия с объектами ... 29
Лабораторная работа № 3. Создание виртуального персонажа с базовым искусственным интеллектом в среде Unreal Engine ... 66
Лабораторная работа № 4. Создание базового проекта для виртуальной реальности (VR), его запаковка и тестирование ... 92
Библиографический список рекомендуемой литературы ... 102

ВВЕДЕНИЕ

Данное учебное издание предназначено для обучающихся уровня бакалавриата по направлению подготовки «Информационные системы и технологии» в качестве практикума по дисциплине «Технологии трехмерного моделирования и виртуальной реальности». Настоящее учебное издание представляет собой вторую часть практикума «Разработка интерактивных трехмерных сцен» и охватывает этапы создания интерактивных систем, организации пользовательского взаимодействия, а также разработки виртуальных персонажей и *VR*-приложений. Оно дополняет изучаемый теоретический материал и позволяет закрепить его на практике.

Практикум направлен на формирование у обучающихся устойчивых навыков разработки интерактивных приложений в *Unreal Engine*, создания систем взаимодействия пользователя с виртуальной средой и реализации базовой логики поведения объектов. Задания лабораторных работ охватывают полный цикл разработки интерактивного приложения: от настройки пользовательского управления и создания интерактивных объектов до реализации виртуальных персонажей с элементами искусственного интеллекта, а также разработки *VR*-приложений с последующим тестированием и упаковкой.

Каждая лабораторная работа включает теоретическую часть, описание среды выполнения, пошаговые инструкции и практические задания, направленные на развитие прикладных навыков. Такой подход позволяет обучающимся не только освоить инструменты, но и понять методику создания трехмерных объектов и сцен, актуальных для применения в сфере информационных технологий, цифрового производства, архитектуры, дизайна и обучающих симуляций.

Предлагаемая система взаимосвязанных лабораторных работ по дисциплине «Технологии трехмерного моделирования и виртуальной реальности» позволяет обучающимся сформировать следующие компетенции:

1) использует готовые макеты графического дизайна интерфейсов информационных систем и визуализации данных, применяет принципы моделирования и способы представления трехмерных моделей для разработки ИС и *web*-дизайна, использует технологии компьютерной анимации в профессиональной деятельности.

Знает: тенденции в графическом дизайне, макеты графического дизайна интерфейсов информационных систем и визуализации данных.

Умеет: использовать готовые макеты графического дизайна интерфейсов информационных систем и визуализации данных.

Владеет: технологиями компьютерной анимации в профессиональной деятельности («ПК-15.1»);

2) разрабатывает трехмерные модели.

Знает: принципы обработки изображений для решения практических задач в области информационных систем и технологий.

Умеет: разрабатывать трехмерные модели, графический дизайн интерфейсов информационных систем.

Владеет: растровыми и векторными пакетами программ («ПК-15.2»);

3) использует программные средства компьютерной анимации для реализации проектов в профессиональной деятельности.

Знает: средства компьютерной анимации.

Умеет: выполнять элементы графического дизайна интерфейсов информационных систем и визуализации данных.

Владеет: программными и аппаратными средствами компьютерной анимации для реализации проектов в профессиональной деятельности («ПК-15.3»).

По результатам выполнения каждой лабораторной работы обучающийся должен подготовить отчет. Он оформляется в соответствии с установленными требованиями и должен содержать структурированную информацию о выполненных этапах работы.

Содержание отчета включает следующие элементы.

1. Титульный лист.

2. Цель работы.

3. Формулировка задания.

4. Описание выполненных действий и полученных результатов (включая текстовые объяснения и иллюстрации, такие как скриншоты).

5. Ответы на контрольные вопросы.

6. Вывод о проделанной работе.

Выполнить лабораторные работы наиболее качественно обучающимся поможет библиографический список рекомендуемой литературы [1–9].

Лабораторная работа № 1
НАСТРОЙКА СИСТЕМЫ ИНТЕРАКТИВНОГО ВЗАИМОДЕЙСТВИЯ С ПОЛЬЗОВАТЕЛЕМ В СРЕДЕ UNREAL ENGINE

Время выполнения – 6 часов (аудиторная работа – 2 часа, самостоятельная работа – 4 часа).

Цель работы: овладеть навыками разработки системы базового взаимодействия пользователя с объектами виртуальной среды в *Unreal Engine*.

Задачи работы

1. Овладеть навыками настройки пользовательского персонажа на базе класса *Character* в *Unreal Engine*.
2. Изучить способы привязки клавиш управления движением, обзором и действиями.
3. Реализовать базовую систему взаимодействия с виртуальной средой с использованием визуального программирования *Blueprint*.

Перечень обеспечивающих средств

1. Персональный компьютер с доступом в интернет.
2. Среда разработки *Unreal Engine* (версия не ниже 5.3).

Общие теоретические сведения

Одним из основных элементов взаимодействия пользователя с виртуальной средой является персонаж, которым он управляет. В терминологии *Unreal Engine* такой объект обычно представлен классом *Pawn*. Данный класс используется для обозначения любого объекта, который может находиться под контролем пользователя или искусственного интеллекта и участвовать во взаимодействии с виртуальным миром.

В свою очередь, понятие *Actor* является более общим и обозначает любой объект, размещенный в сцене и способный выполнять определенные действия. Класс *Pawn* является частным случаем *Actor* и выступает в качестве базового

класса для всех объектов, которыми может управлять игрок или система искусственного интеллекта.

Класс *Pawn* определяет объект, которым может управлять пользователь или искусственный интеллект, и отвечает за его поведение в виртуальной среде, включая взаимодействие с физикой, коллизии и обработку ввода.

Для начала работы необходимо подготовить структуру проекта. В интерфейсе *Unreal Engine* для хранения ресурсов используется *Content Browser* (*Content Drawer*), расположенный в нижней части окна.

Создайте новую папку для хранения объектов персонажа. Для этого щелкните правой кнопкой мыши по корневой папке *Content* и выберите пункт *New Folder*, затем задайте имя папки – *Character*.

Перейдите в созданную папку и создайте новый *Blueprint Class*. Для этого откройте контекстное меню внутри папки *Character* и выберите соответствующий пункт. Окно создания *Blueprint Class* представлено на рис. 1.1.

В открывшемся окне выберите в качестве родительского класса *Character* и задайте имя создаваемого объекта – *BP_Player* (рис. 1.2). Класс *Character* является специализированным типом *Pawn* и содержит встроенные средства для перемещения в пространстве.

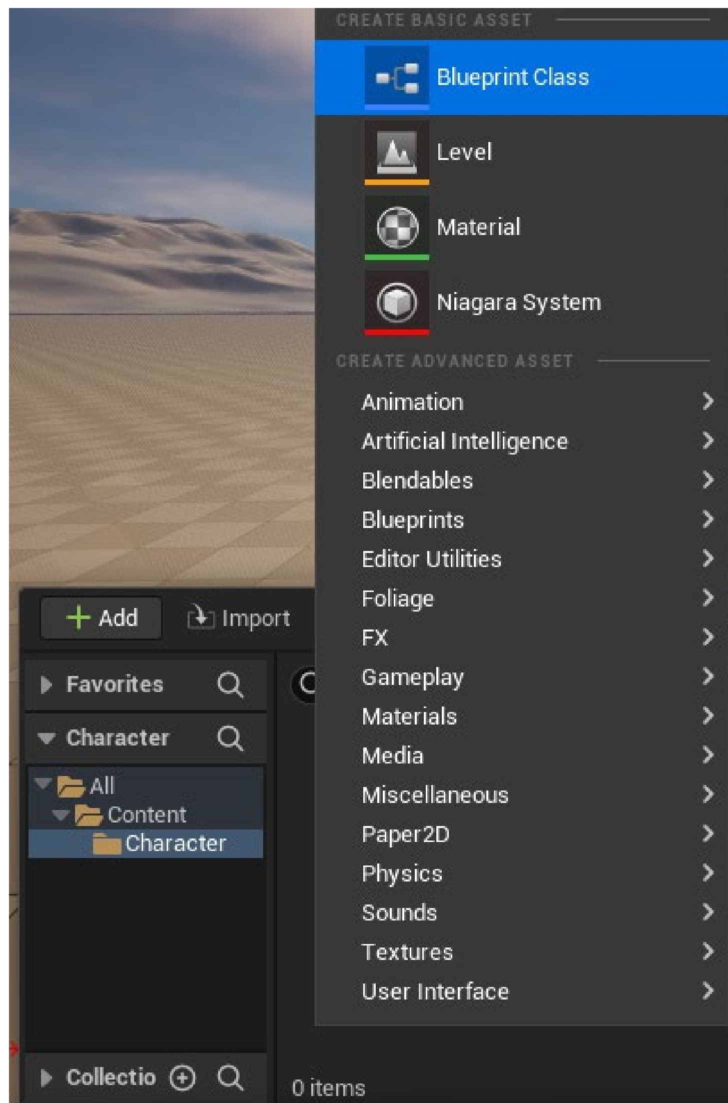

Рис. 1.1. Всплывающее меню *Content Browser*

В отличие от базового *Pawn*, класс *Character* уже включает компонент *CharacterMovement*, который отвечает за реализацию стандартных механик передвижения, таких как ходьба, прыжки и управление направлением взгляда. Благодаря этому достаточно использовать готовые функции и события, а обработка перемещения и взаимодействия с физической средой выполняется автоматически.

Использование класса *Character* позволяет упростить реализацию базового управления персонажем и сосредоточиться на логике взаимодействия с пользователем.

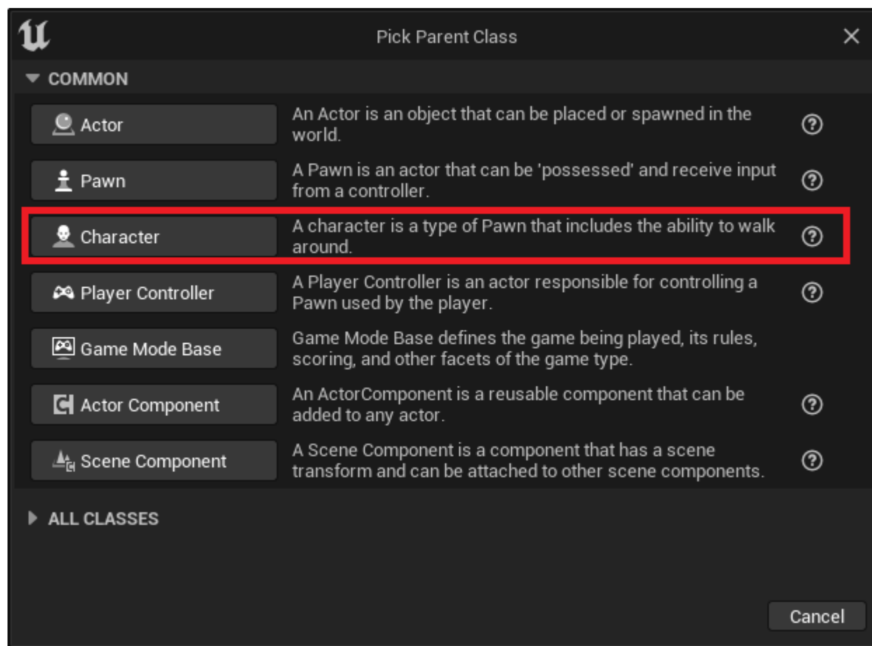

Рис. 1.2. Окно выбора родительского класса

Для реализации перемещения персонажа необходимо обеспечить обработку пользовательского ввода, в частности нажатий клавиш. Для этого параметры движения связываются с соответствующими клавишами управления.

В качестве стандартной схемы управления обычно используются клавиши *W*, *A*, *S*, *D* либо стрелки (↑, ←, ↓, →).

Настройка привязки клавиш выполняется в параметрах проекта. В *Unreal Engine* для этого откройте раздел *Project Settings*. Чтобы перейти к нему, воспользуйтесь меню *Edit*, расположенным в верхней части окна редактора, и выберите пункт *Project Settings* (рис. 1.3).

В окне *Project Settings* перейдите в раздел *Engine* и выберите пункт *Input* (рис. 1.4). Данный раздел содержит параметры, определяющие обработку пользовательского ввода.

Здесь настраивается соответствие между действиями пользователя и событиями, используемыми в логике приложения, что позволяет управлять персонажем и другими объектами виртуальной среды.

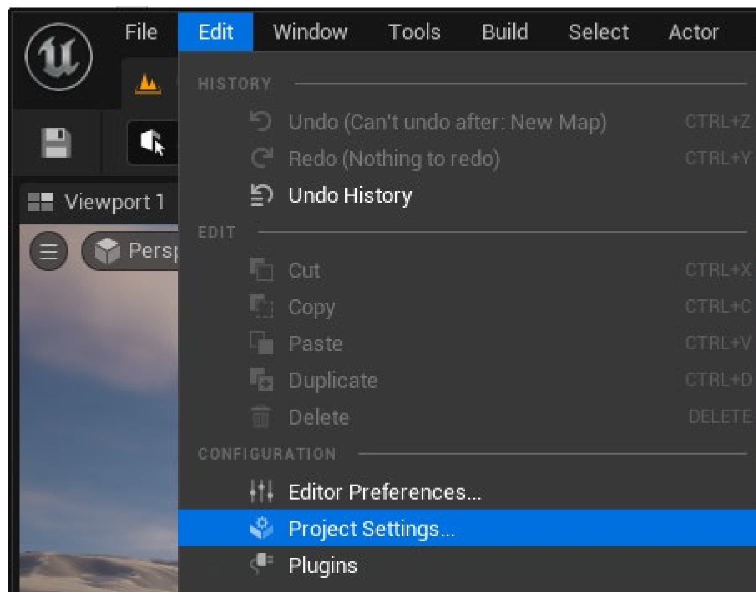

Рис. 1.3. Меню *Edit*

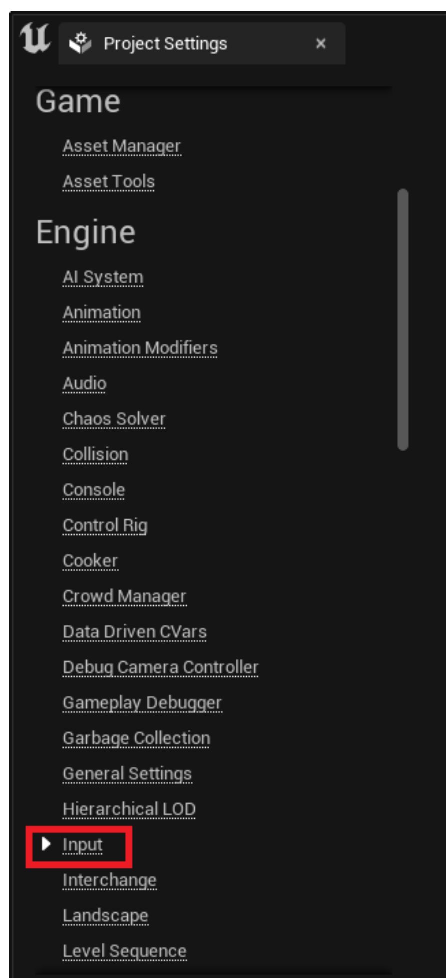

Рис. 1.4. Окно *Project Settings*

В *Unreal Engine* предусмотрена привязка устройств ввода к осям (*Axis Mappings*) и действиям (*Action Mappings*), что позволяет разделять непрерывный и дискретный ввод.

Axis Mappings используются для обработки непрерывного ввода и возвращают числовые значения. По умолчанию значение равно 0. При нажатии клавиш оно может принимать значения 1 или −1. Например, значение 1 соответствует движению вперед, а −1 – движению назад. Данный тип ввода применяется для управления перемещением и поворотом камеры.

Action Mappings предназначены для обработки дискретных событий. Они имеют два состояния – активное и неактивное – и срабатывают при нажатии или отпускании клавиши. Такой тип используется для действий, не требующих промежуточных значений, например прыжка.

Настройка привязок выполняется в разделе *Bindings* окна *Input*.

Для реализации движения создайте ось *MoveForward* в разделе *Axis Mappings*. Для этого нажмите на кнопку добавления (пункт 1 на рис. 1.5), разверните список осей (пункт 2 на рис. 1.5) и задайте имя *MoveForward*. Затем добавьте привязки (пункт 3 на рис. 1.5): для клавиши *W* (или ↑) установите значение *Scale = 1*, для клавиши *S* (или ↓) – *Scale = –1*.

Для назначения клавиш щелкните левой кнопкой мыши по кнопке слева от значения *None*. По умолчанию она отображается в виде значка клавиатуры (внешний вид может отличаться в зависимости от версии *Unreal Engine*). После нажатия выберите нужную клавишу: для движения вперед – *W* (или ↑), для движения назад – *S* (или ↓).

Аналогично настройте перемещение в стороны. Создайте ось *MoveRight* в разделе *Axis Mappings*. Для нее задайте следующие привязки: клавиша *D* (или →) – *Scale = 1*, клавиша *A* (или ←) – *Scale = –1*.

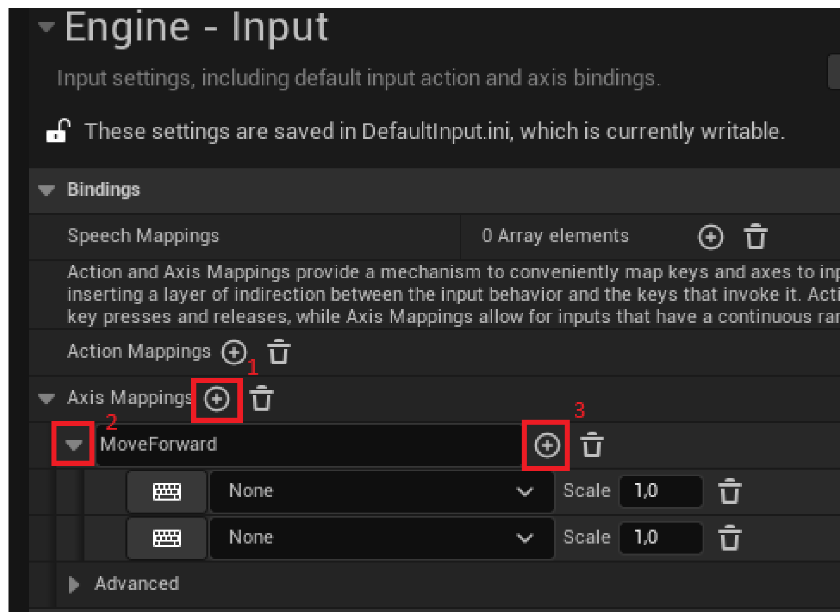

Рис. 1.5. Вкладка Bindings, создание оси MoveForward

После настройки проверьте корректность параметров осей и значений *Scale* (рис. 1.6). В результате должны быть заданы две оси управления: *MoveForward* и *MoveRight*.

Итоговые соответствия:

– клавиша *W* (или ↑) в оси *MoveForward* – *Scale* = *1*;

– клавиша *S* (или ↓) в оси *MoveForward* – *Scale* = *–1*;

– клавиша *A* (или ←) в оси *MoveRight* – *Scale* = *–1*;

– клавиша *D* (или →) в оси *MoveRight* – *Scale* = *1*.

Помимо перемещения персонажа требуется реализовать возможность осмотра окружающей среды, то есть поворота «камеры». В *Unreal Engine* это выполняется с помощью дополнительных осей ввода.

В разделе *Axis Mappings* создайте две новые оси: *LookHorizontal* и *LookVertical*. Первая отвечает за горизонтальное движение камерой, вторая – за вертикальное.

Для настройки выберите соответствующие оси мыши: *Mouse X* для *LookHorizontal* и *Mouse Y* для *LookVertical*. Выбор выполняется аналогично привязке клавиш – через выпадающий список с последующей настройкой параметров (рис. 1.7).

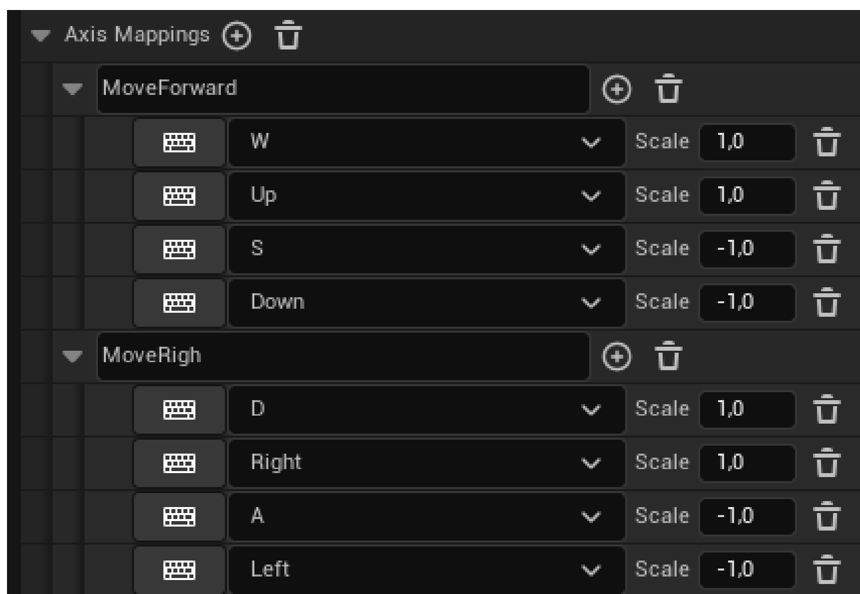

Рис. 1.6. Настройка *Axis Mappings*

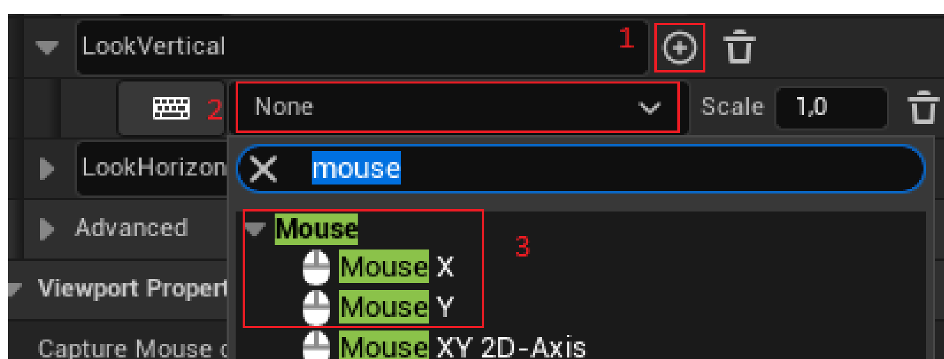

Рис. 1.7. Выбор осей мыши

Для оси *LookHorizontal* выберите значение *Mouse X* и задайте параметр *Scale = 1*. В этом случае движение мыши вправо будет вызывать поворот камеры вправо, а движение влево – поворот влево. При установке отрицательного значения *Scale* управление будет инвертировано.

Для оси *LookVertical* выберите значение *Mouse Y* и установите *Scale = –1*. Это обеспечивает привычное управление: движение мыши на себя приводит к повороту взгляда вверх, а движение от себя – вниз. При значении *Scale = 1* управление по вертикали будет инвертировано.

Итоговая конфигурация осей должна соответствовать примеру, представленному на рис. 1.8.

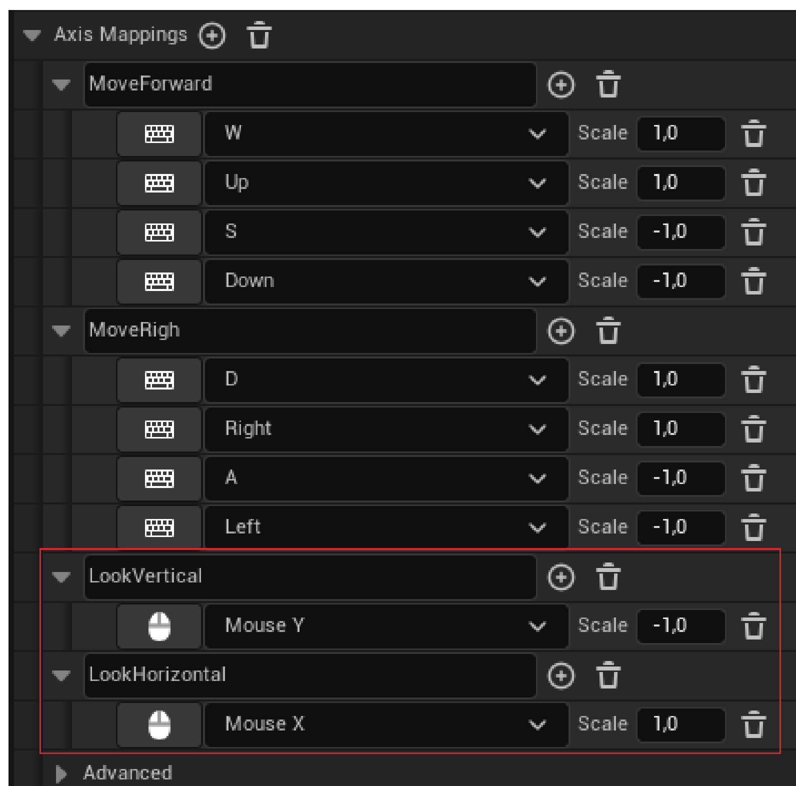

Рис. 1.8. Окно *Project Settings* (итоговые значения осей)

После настройки осей движения и обзора можно перейти к созданию действия «Прыжок» с использованием *Action Mappings*. Данный тип ввода предназначен для обработки дискретных событий, таких как нажатие клавиш.

Для создания действия в разделе *Action Mappings* нажмите на кнопку добавления, затем разверните список записей и введите имя действия – *Jump*.

После этого добавьте привязку для *Jump*, нажав на кнопку добавления рядом с созданным действием. В появившемся поле со значением *None* укажите клавишу *Space* (пробел).

В результате должны быть настроены четыре оси ввода (*MoveForward*, *MoveRight*, *LookHorizontal*, *LookVertical*) и одно действие (*Jump*). Проверьте корректность настроек, сопоставив их с примером на рис. 1.9.

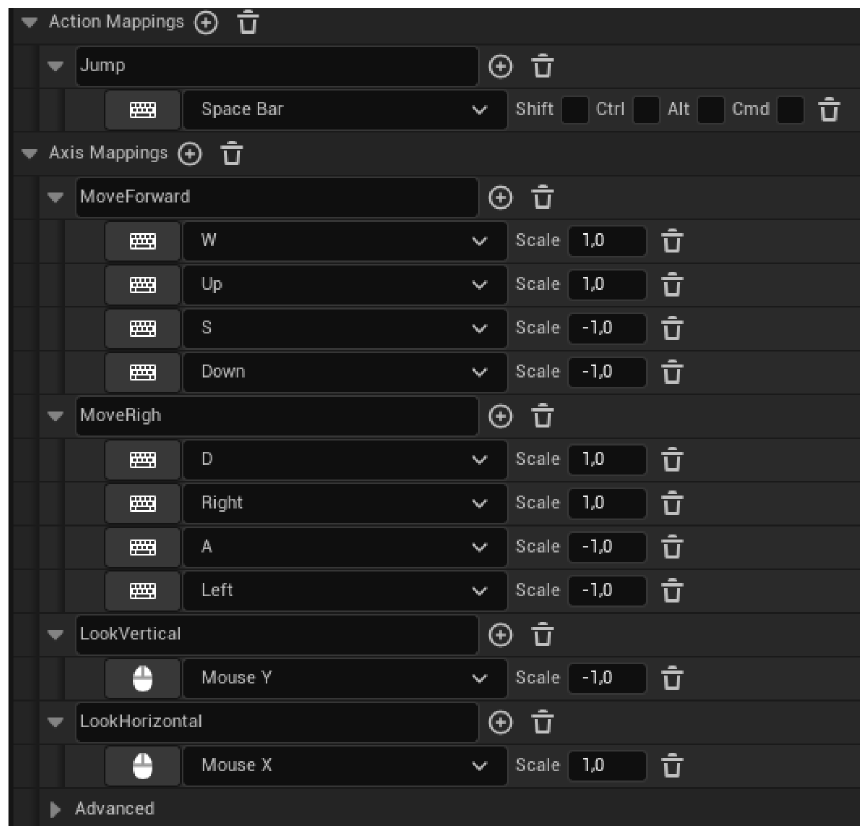

Рис. 1.9. Окно *Project Settings* (итоговый результат)

После завершения настройки и закрытия окна *Project Settings* перейдите в панель *Content Browser*. Ранее в рамках работы была создана папка *Character*, в которой размещен объект типа *Blueprint Class* с именем *BP_Player*.

Созданный объект представляет собой заготовку игрового персонажа и отображается в панели содержимого проекта в виде ресурса (рис. 1.10).

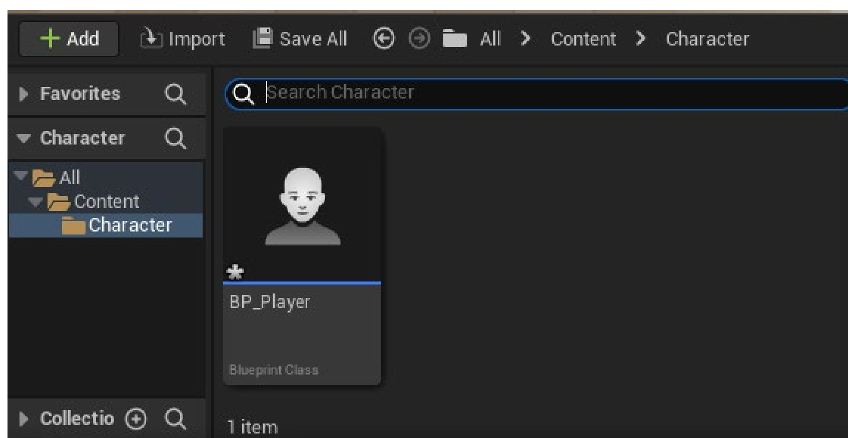

Рис. 1.10. *Blueprint Class* под названием *BP_Player* в папке *Character*

Двойным щелчком левой кнопки мыши по объекту *BP_Player* откройте окно редактирования персонажа (рис. 1.11). В *Unreal Engine Blueprint Editor* представляет собой среду для настройки логики и параметров объекта.

В левой части окна расположена панель *Components*, содержащая список компонентов персонажа. По умолчанию для класса *Character* уже задан компонент *Capsule*, определяющий физические габариты персонажа, а также компонент *CharacterMovement*, отвечающий за перемещение, включая ходьбу и прыжки.

В центральной части находится окно *Viewport*, в котором отображается визуальное представление персонажа и его компонентов. Здесь можно наблюдать изменения параметров и их влияние на внешний вид и поведение объекта.

Справа расположена панель *Details*, предназначенная для редактирования свойств выбранных компонентов. Например, во вкладке *Shape* можно изменить размеры капсулы, что влияет на взаимодействие с окружающей средой.

В верхней части окна находится панель инструментов. Основные команды – кнопки *Compile* (проверка и сборка логики *Blueprint*) и *Save* (сохранение изменений).

Основная логика поведения реализуется в разделе *Event Graph*. Он представляет собой граф визуального программирования, построенный на основе событий (*Events*), которые определяют начало выполнения логики. События запускают процессы при наступлении заданных условий, например при старте игры или нажатии на клавишу.

В одном *Blueprint* может быть определено несколько событий, при этом для каждого типа события обычно используется отдельный экземпляр.

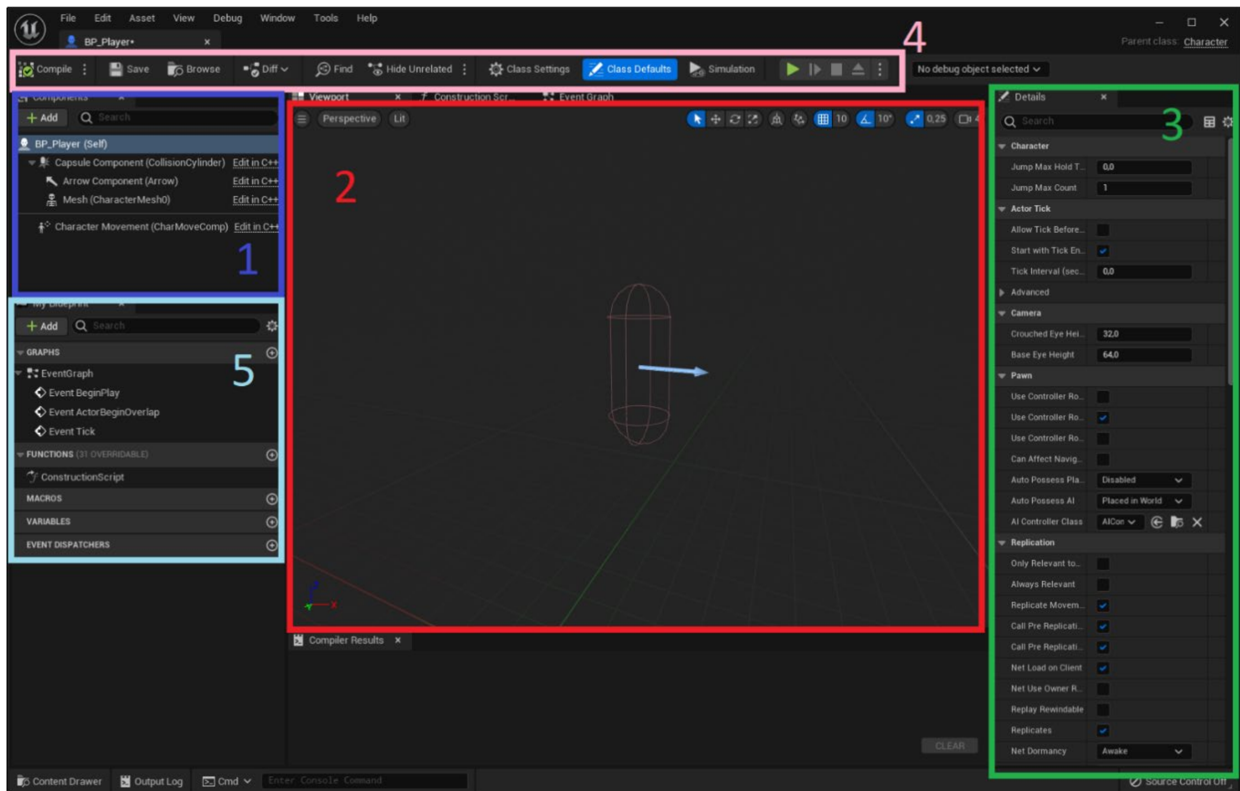

Рис. 1.11. Окно редактирования персонажа

Для реализации управления персонажем, включающего движение, прыжок и обзор, создайте соответствующие события для обработки ввода. Используются два события для движения, два – для управления обзором и одно – для выполнения действия *Jump*.

Работа с логикой выполняется в разделе *Event Graph*. Для добавления элементов щелкните правой кнопкой мыши по пустому месту рабочей области. Откроется контекстное меню с доступными элементами *Blueprint* (рис. 1.12).

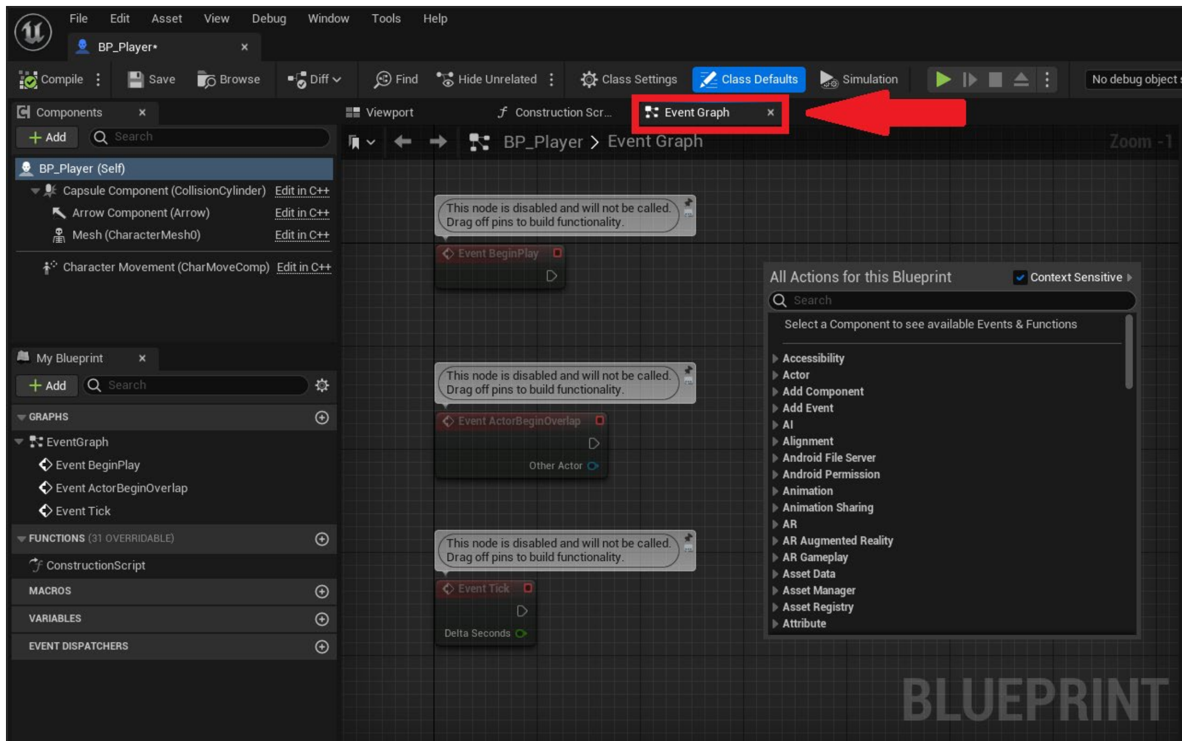

Рис. 1.12. Окно редактирования персонажа, вкладка *Event Graph*

Для добавления ветки события, отвечающей за перемещение персонажа вперед и назад при нажатии соответствующих клавиш, воспользуйтесь поиском в списке элементов *Blueprint*. В поисковой строке введите название события *MoveForward*. Данное событие располагается в разделе *Axis Events* и при выборе подсвечивается, что упрощает его идентификацию.

После того как нужное событие найдено, выберите его щелчком левой кнопки мыши. В результате оно будет добавлено в рабочую область *Event Graph*, где его можно использовать для дальнейшего построения логики перемещения (рис. 1.13).

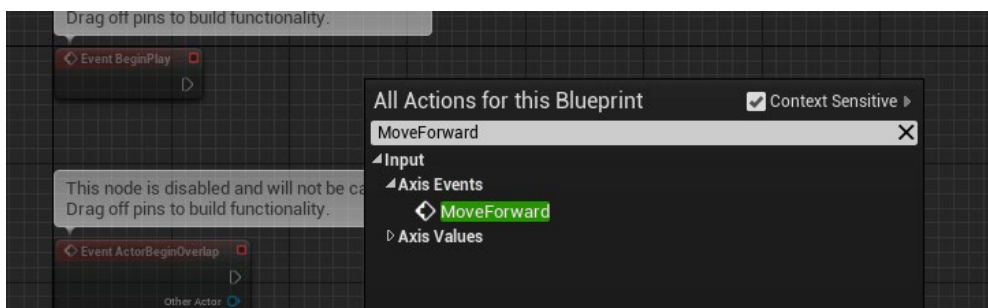

Рис. 1.13. Добавление события *MoveForward*

Далее добавьте узел, отвечающий за обработку входных данных движения. Для этого на пустом поле *Event Graph* щелкните правой кнопкой мыши, чтобы открыть список доступных элементов *Blueprint*. В появившемся меню воспользуйтесь поисковой строкой и найдите узел *Add Movement Input*, расположенный в разделе *Input* (рис. 1.14).

После того как нужный узел будет найден, выберите его щелчком левой кнопки мыши. В результате узел будет добавлен в рабочую область *Event Graph*. Затем соедините выходы события *MoveForward* с входами узла *Add Movement Input*, как показано на рис. 1.15. Такое соединение обеспечивает передачу значения оси в систему движения персонажа.

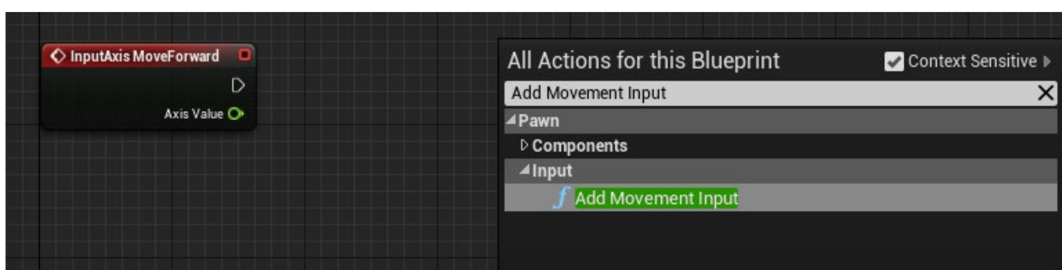

Рис. 1.14. Добавление узла *Add Movement Input*

Рис. 1.15. Соединение события MoveForward и Add Movement Input

Для того чтобы система корректно определяла направление движения персонажа, необходимо учитывать его текущую ориентацию в пространстве, т. е. направление, в котором он «смотрит». Иными словами, требуется получить вектор направления, относительно которого будет выполняться перемещение.

Для этого используется узел *Get Actor Forward Vector*, который возвращает вектор, соответствующий направлению «вперед» для текущего объекта. Чтобы добавить данный узел, щелкните правой кнопкой мыши на пустом поле *Event Graph* и откройте список доступных элементов *Blueprint*. Далее воспользуйтесь поисковой строкой и найдите узел *Get Actor Forward Vector*, расположенный в разделе *Transformation*.

После выбора узел будет добавлен в рабочую область *Event Graph*. Затем соедините выход узла с соответствующим входом узла *Add Movement Input*, как показано на рис. 1.16. Такое соединение позволяет системе корректно интерпретировать направление движения персонажа относительно его текущего положения в пространстве.

Для реализации движения персонажа в стороны добавьте отдельную ветку событий, отвечающую за обработку соответствующих клавиш. В списке доступных элементов *Blueprint* найдите событие *MoveRight*, которое располагается в разделе *Axis Events*. Это событие будет обрабатывать ввод для перемещения персонажа вправо и влево.

Рис. 1.16. Соединение Get Actor Forward Vector и Add Movement Input

Далее добавьте узел обработки входных данных движения – *Add Movement Input*. Для этого на рабочем поле *Event Graph* щелкните правой кнопкой мыши и с помощью поисковой строки найдите данный узел, расположенный в разделе *Input*.

Чтобы корректно задать направление движения, необходимо учитывать текущее положение персонажа относительно его правой стороны. Для этого используйте узел *Get Actor Right Vector*, который возвращает вектор, напра-

вленный вправо относительно текущей ориентации объекта. Добавьте данный узел через поиск в *Blueprint* в разделе *Transformation*.

После добавления всех необходимых элементов соедините их аналогично ранее реализованной логике движения вперед и назад, обеспечив передачу значений оси в систему перемещения персонажа.

Для реализации управления обзором добавьте события, отвечающие за обработку движения мыши. В списке элементов *Blueprint* найдите события *LookHorizontal* и *LookVertical*, расположенные в разделе *Axis Events*. Эти события будут обрабатывать изменение положения камеры по горизонтали и вертикали.

Для управления поворотом камеры используйте узлы *Add Controller Yaw Input* и *Add Controller Pitch Input*. Первый отвечает за поворот камеры по оси *Yaw* (вращение влево и вправо), второй – по оси *Pitch* (наклон вверх и вниз). Добавьте данные узлы и соедините их с соответствующими событиями, как показано на рис. 1.17 и 1.18.

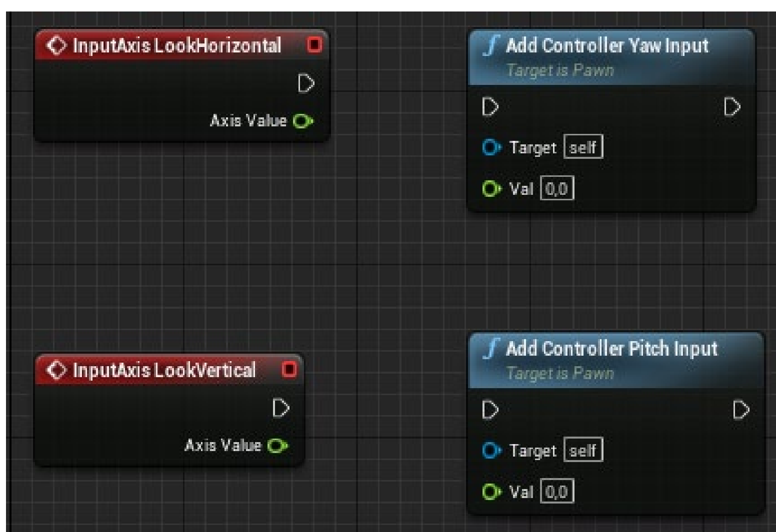

Рис. 1.17. Добавление событий LookHorizontal и LookVertical, а также узлов Add Controller Yaw Input и Add Controller Pitch Input

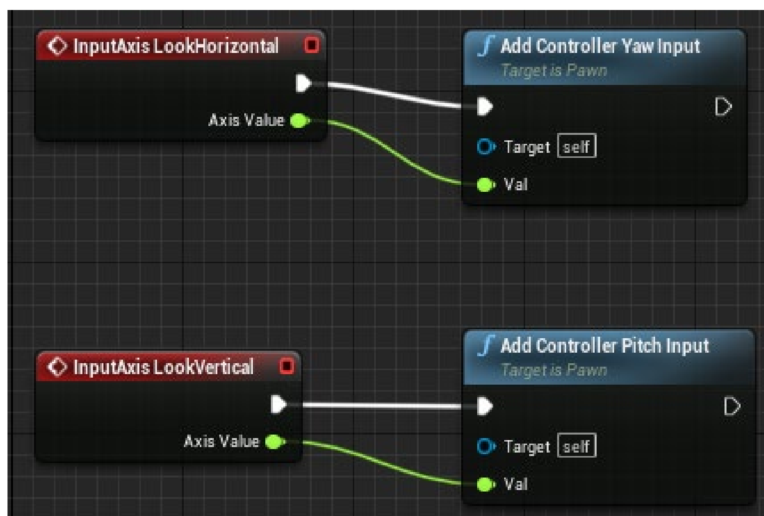

Рис. 1.18. Соединение событий *LookHorizontal* и *LookVertical* с узлами управления камерой

Для реализации действия «прыжок» добавьте соответствующее событие в системе *Blueprint*. В списке доступных элементов найдите событие *Jump*, расположенное в разделе *Action Events*. Оно отвечает за обработку нажатия клавиши, связанной с выполнением прыжка.

Далее добавьте узлы *Jump* и *Stop Jumping*. Узел *Jump* запускает выполнение действия прыжка при нажатии клавиши, а узел *Stop Jumping* завершает выполнение действия при ее отпускании. Такая схема позволяет корректно обрабатывать как начало, так и окончание действия, обеспечивая естественное поведение персонажа.

После добавления всех необходимых элементов разместите их на рабочем поле *Event Graph* и соедините между собой в соответствии с примером, представленным на рис. 1.19.

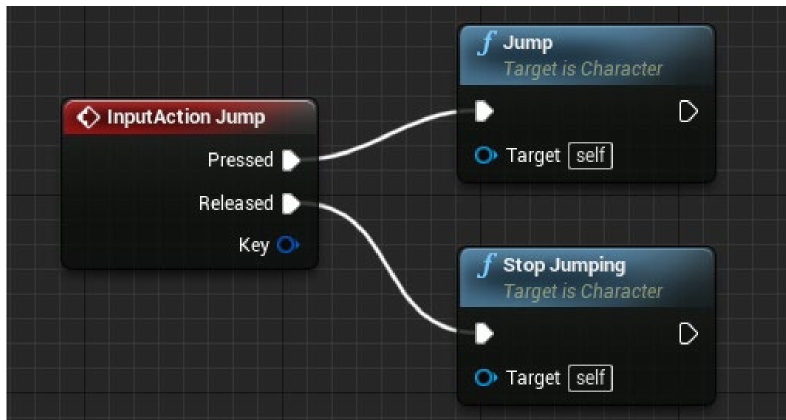

Рис. 1.19. Реализация прыжка

После завершения настройки всех элементов и при отсутствии нужного внесения дополнительных изменений перейдите к панели инструментов, представленной на рис. 1.20. В ней нажмите на кнопку *Compile* и дождитесь завершения процесса компиляции. Данная операция позволяет проверить корректность созданной логики и подготовить ее к дальнейшему использованию.

После успешной компиляции нажмите на кнопку *Save* для сохранения всех внесенных изменений. Это обеспечит сохранение текущей конфигурации и предотвратит потерю настроек при последующей работе с проектом.

Полученную реализацию управления можно сравнить с эталонным вариантом, представленным на рис. 1.21.

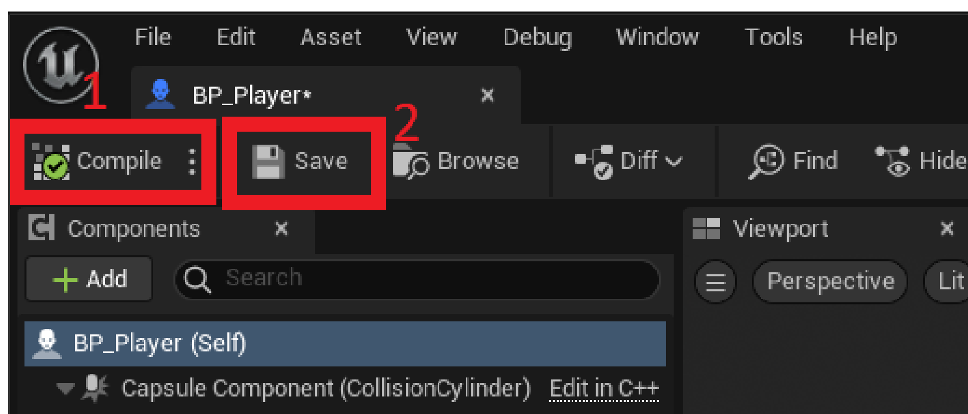

Рис. 1.20. Панель инструментов

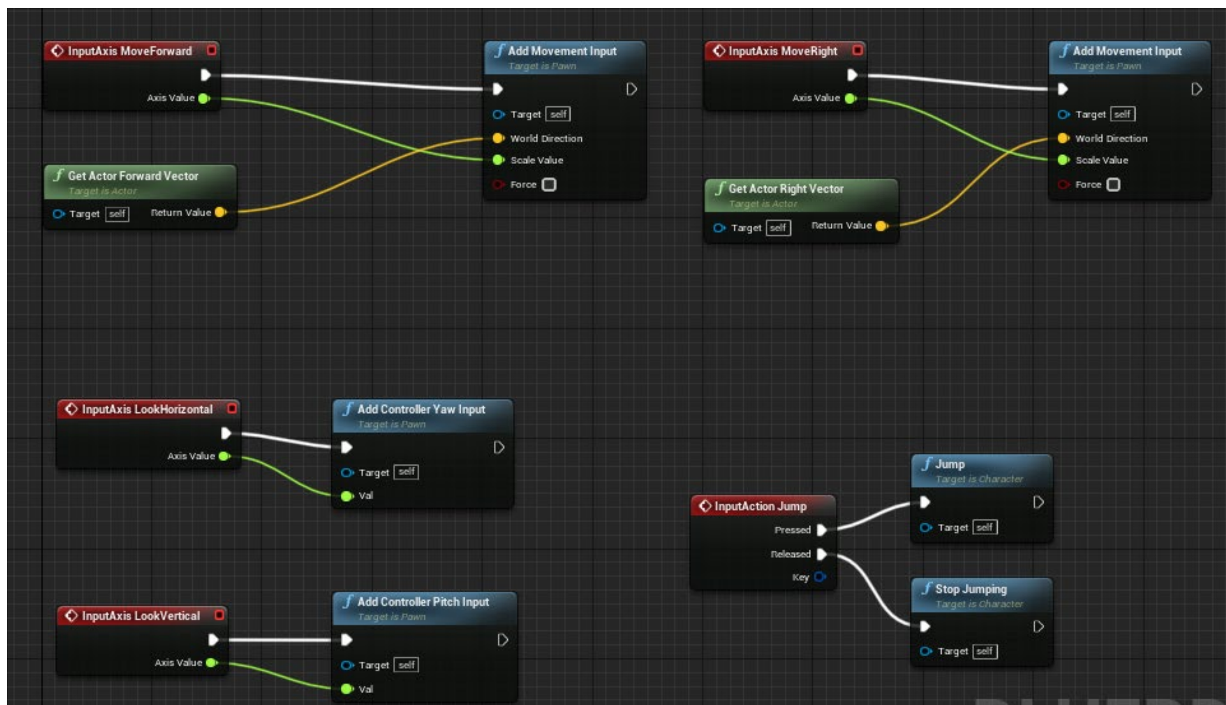

Рис. 1.21. Итоговый набор управления

После сохранения созданного *Blueprint Class* с именем *BP_Player* вернитесь к панели *Content Browser*. В папке *Character* создайте еще один *Blueprint Class*.

Для этого щелкните правой кнопкой мыши на пустом поле внутри папки *Character* и выберите соответствующий пункт из всплывающего меню. В открывшемся окне в качестве родительского класса выберите *Game Mode* и задайте новому классу имя *My_Game_Mode* (рис. 1.22).

Game Mode представляет собой основной класс, определяющий правила игры, а также логику управления и взаимодействия в пределах уровня. Именно он отвечает за базовое поведение игрового процесса.

По завершении создания в папке *Character* должно находиться два *Blueprint Class*.

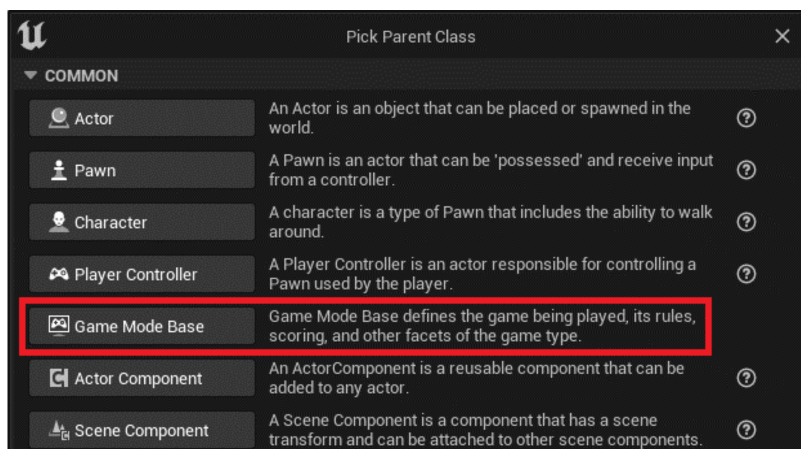

Рис. 1.22. Окно выбора родительского класса для *GameMode*

После создания собственного *Game Mode* («режима игры») назначьте его в качестве основного для текущего мира проекта. Для этого в главном окне откройте меню *Settings*, расположенное в верхней части экрана, и выберите в нем пункт *World Settings* (рис. 1.23).

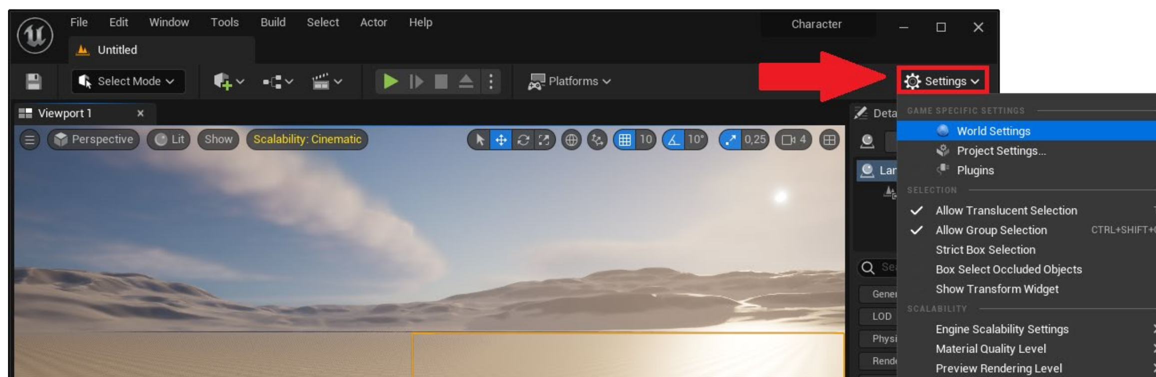

Рис. 1.23. Меню *Settings*

В правой нижней части экрана появится панель *World Settings*. В ней найдите раздел *Game Mode* и в поле *GameMode Override* выберите ранее созданный класс *My_Game_Mode*.

Далее в разделе *Selection GameMode* найдите параметр *Default Pawn Class* и выберите созданный персонаж *BP_Player* (рис. 1.24). Это позволит назначить данный класс в качестве управляемого объекта по умолчанию.

Настройка *GameMode* выполняется для каждого уровня, в котором предполагается использование одинаковой логики управления персонажем.

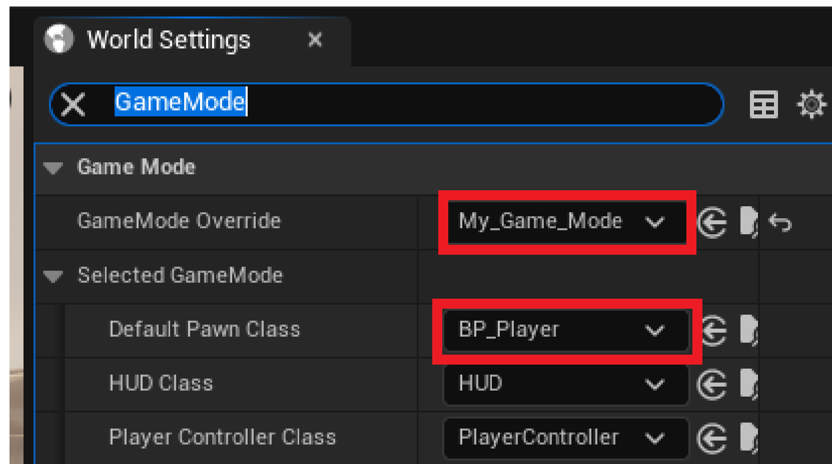

Рис. 1.24. Выбор режима *GameMode* и параметра *Default Pawn Class*

Для корректного появления персонажа на уровне разместите элемент *Player Start*, который задает точку появления персонажа при запуске уровня. Его можно найти во вкладке добавления объектов, расположенной на верхней панели редактора (рис. 1.25). Разместите *Player Start* в нужной точке сцены и убедитесь, что персонаж будет появляться именно в этом месте при старте уровня. Если ранее на уровне уже была добавлена другая точка появления, удалите ее и замените новой.

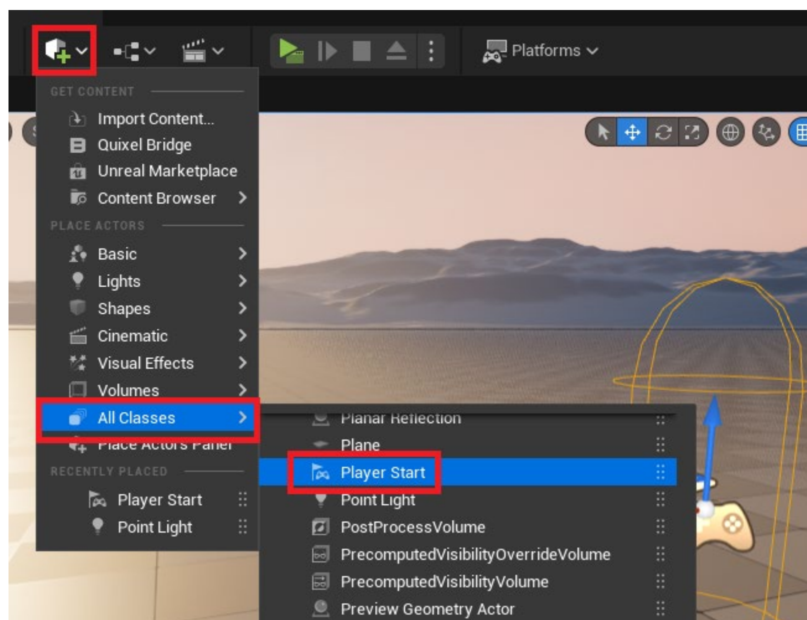

Рис. 1.25. Размещение элемента *Player Start*

Задания

Результат работы необходимо сохранить в виде проекта *Unreal Engine*, совместимого с версией *UE 5.3* и выше. Название проекта должно соответствовать шаблону «*Familiya_Character*».

Следует учитывать, что в названии проекта запрещается использование русских символов, пробелов и специальных знаков. Допускается применение только латинских букв, цифр и символа подчеркивания.

Задание 1. Реализация базового управления персонажем.

1. Создайте новый *Blueprint Class* на основе класса *Character* и назовите его *BP_Player*.
2. Настройте оси управления движением и вращением камеры.
3. Создайте действие *Action Jump* с привязкой к клавише «Пробел».
4. Настройте и протестируйте управление персонажем.

Задание 2. Реализация динамического изменения скорости.

1. В компоненте *Character Movement* у *BP_Player* найдите переменную *Max Walk Speed*.
2. Реализуйте изменение скорости персонажа при нажатии на клавиши *Shift* и *Ctrl*: при нажатии *Shift* скорость должна увеличиваться в 2 раза, а при нажатии *Ctrl* – в 3 раза.

3. При отпускании указанных клавиш следует возвращать значение скорости к стандартному.

Контрольные вопросы

1. Чем отличается класс *Character* от *Pawn* в *Unreal Engine*?
2. Что такое *Axis Mappings* и как они используются при создании управления?
3. Для чего используются *Action Mappings* в настройках ввода?
4. Как изменить скорость перемещения персонажа во время игры?
5. Для чего используется компонент *CharacterMovement* в классе *Character*?

Лабораторная работа № 2
СОЗДАНИЕ ИНТЕРАКТИВНЫХ ОБЛАСТЕЙ ВЗАИМОДЕЙСТВИЯ С ОБЪЕКТАМИ

Время выполнения – 8 часов (аудиторная работа – 4 часа, самостоятельная работа – 4 часа).

Цель работы: освоение навыков разработки интерактивных областей виртуальной среды, реагирующих на действия пользователя, и управления виртуальными объектами в *Unreal Engine*.

Задачи работы

1. Освоить принципы создания интерактивных объектов с возможностью динамической замены трехмерных моделей в *Unreal Engine*.
2. Изучить реализацию взаимодействия пользователя с объектами с использованием виджетов (*Widget*) и областей коллизии (*Collision*).
3. Разработать механизм управления заменой моделей с использованием ввода с клавиатуры.

Перечень обеспечивающих средств

1. Персональный компьютер с доступом в интернет.
2. Среда разработки *Unreal Engine* (версия не ниже 5.3).

Общие теоретические сведения

В процессе разработки проектов в *Unreal Engine* нередко возникает необходимость создания системы, обеспечивающей случайную генерацию объектов, размещаемых в виртуальной среде. При этом их расположение может изменяться при каждом запуске проекта, что позволяет повысить вариативность и интерактивность сцены. Дополнительно может потребоваться реализация возможности выбора или смены модели пользователем в процессе работы приложения.

Для решения подобных задач разработчик может создавать собственные инструменты, используя средства визуального программирования и встроенные механизмы движка. В рамках данной работы рассматривается один из возможных подходов к организации подобной системы.

На начальном этапе структурируйте содержимое проекта. Для этого в *Content Browser* создайте три отдельные папки (рис. 2.1), каждая из которых будет использоваться для хранения определенного типа ресурсов. Первую папку используйте для размещения моделей объектов, вторую – для хранения объектов типа *Blueprint Class*, а третью – для интерфейсных элементов, включая пользовательские виджеты и другие элементы взаимодействия с пользователем.

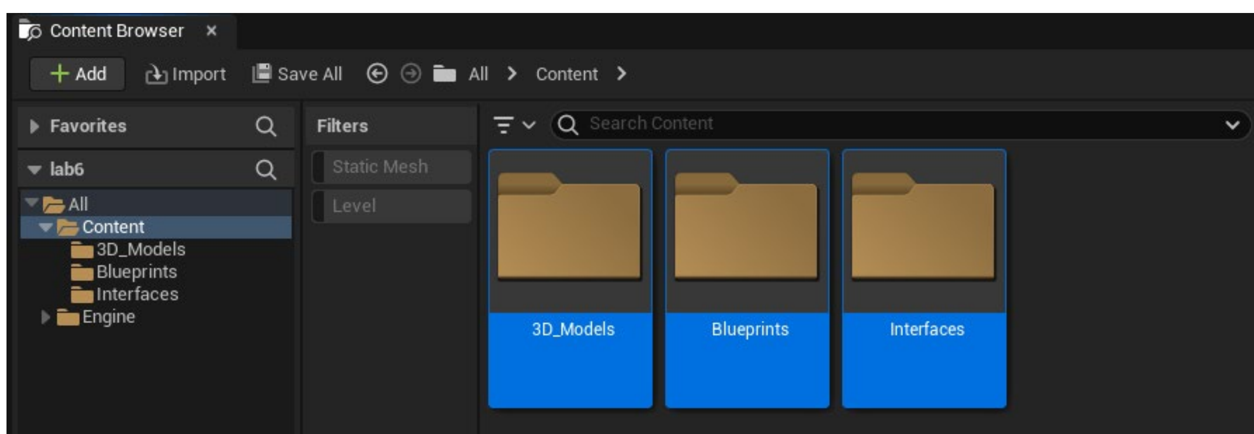

Рис. 2.1. Создание новых папок в проекте

Для создания нового объекта воспользуйтесь *Content Browser*. В папке «*Blueprints*» откройте контекстное меню, щелкнув правой кнопкой мыши по пустой области, и выберите пункт создания нового *Blueprint Class*.

В открывшемся окне выбора типа *Blueprint Class* укажите класс *Actor*. Объекты данного типа могут размещаться непосредственно на уровне и, как правило, выступают в роли основных интерактивных элементов сцены.

Создайте объект и сохраните его под именем *Random_BP* (рис. 2.2). После сохранения откройте его двойным щелчком левой кнопки мыши. Объект откроется в специализированном редакторе, предназначенном для настройки логики и структуры объектов данного типа.

Учтите, что объекты класса *Actor* могут включать в себя несколько компонентов, определяющих их поведение и внешний вид. Управление структурой объекта выполняйте во вкладке *Components*, где можно добавлять, удалять и настраивать отдельные элементы (рис. 2.3).

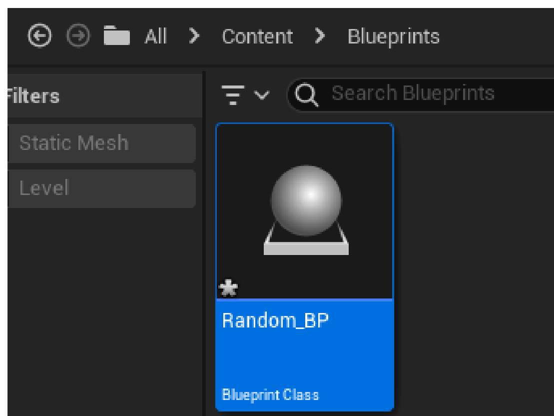

Рис. 2.2. Созданный объект *Random_BP*

После создания объекта добавьте в его структуру компонент, отвечающий за визуальное представление. Для этого в панели *Components* добавьте компонент типа *Static Mesh*. Данный компонент предназначен для хранения и отображения трехмерных моделей, которые будут использоваться в сцене (рис. 2.4).

Учтите, что положение отображаемой модели определяется положением самого компонента *Static Mesh* внутри объекта *Random_BP*. Иными словами, точка привязки модели совпадает с координатами размещения данного компонента, что напрямую влияет на ее расположение в пространстве сцены (рис. 2.5).

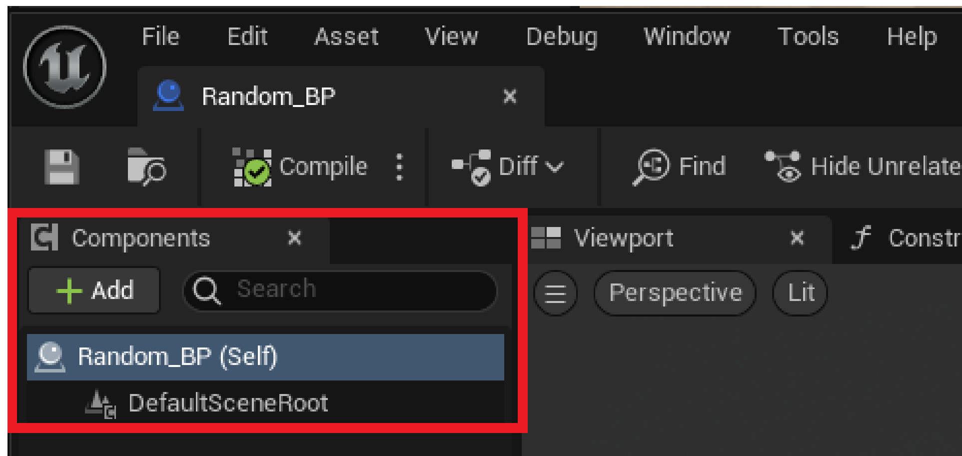

Рис. 2.3. Добавление новых компонентов в объект *Random_BP*

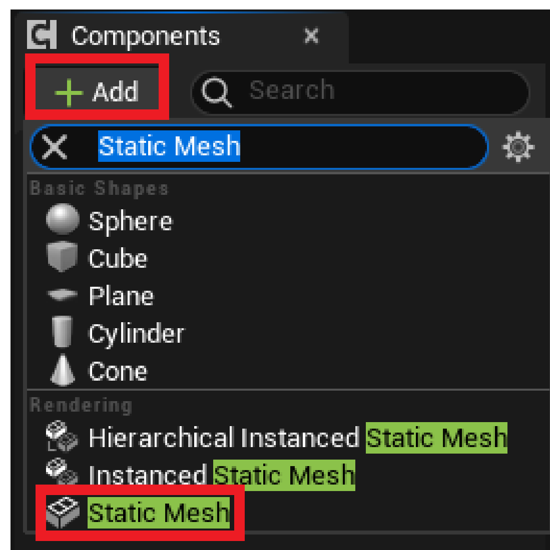

Рис. 2.4. Добавление компонента *Static Mesh*

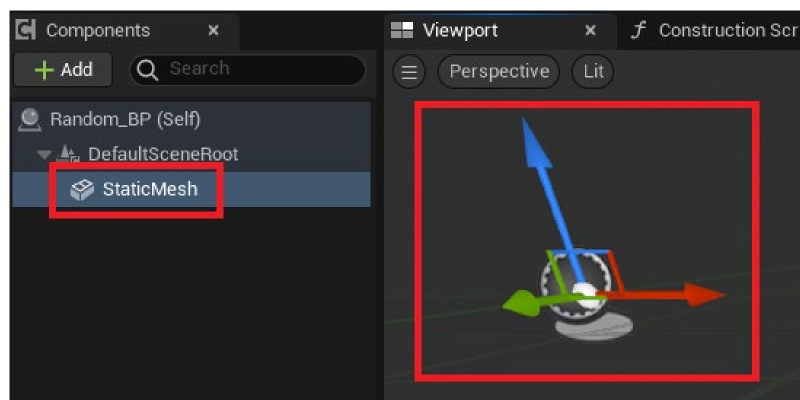

Рис. 2.5. Компонент *Static Mesh*

Компонент *Static Mesh* используется для отображения неподвижной (статической) трехмерной модели. В рамках одного объекта типа *Actor* таких компонентов может быть несколько, при этом каждый из них может иметь собственное имя. По сути данный компонент можно рассматривать как контейнер или переменную, в которую загружается конкретная модель (*mesh*-объект). Такой подход позволяет гибко изменять внешний вид объекта: для этого достаточно заменить модель в компоненте, не создавая новый *Actor*.

В параметрах компонента *Static Mesh* присутствует одноименная вкладка, содержащая ссылку на используемую трехмерную модель. Для ее назначения перетащите нужный объект из *Content Browser* в соответствующее поле. После этого модель будет отображаться в сцене. При необходимости оставьте данное поле пустым – в этом случае при запуске уровня объект не будет иметь визуального представления (рис. 2.6).

Набор моделей, используемых в компоненте *Static Mesh*, может быть сформирован самостоятельно на основе доступных ресурсов проекта либо предоставлен преподавателем в виде готовой подборки.

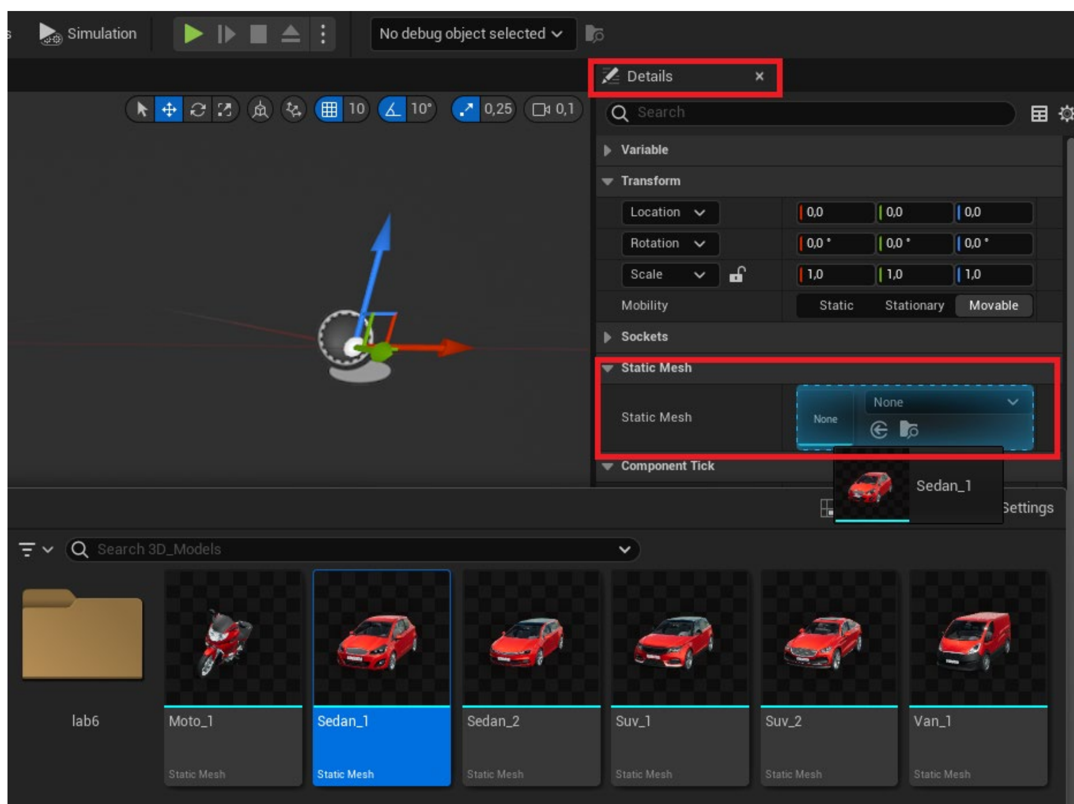

Рис. 2.6. Параметры компонента *Static Mesh*

После изменения данного параметра можно заметить, что компонент *Static Mesh* перестает отображаться в виде условной точки и начинает визуализировать выбранную трехмерную модель. Это позволяет наглядно контролировать внешний вид объекта непосредственно в редакторе (рис. 2.7).

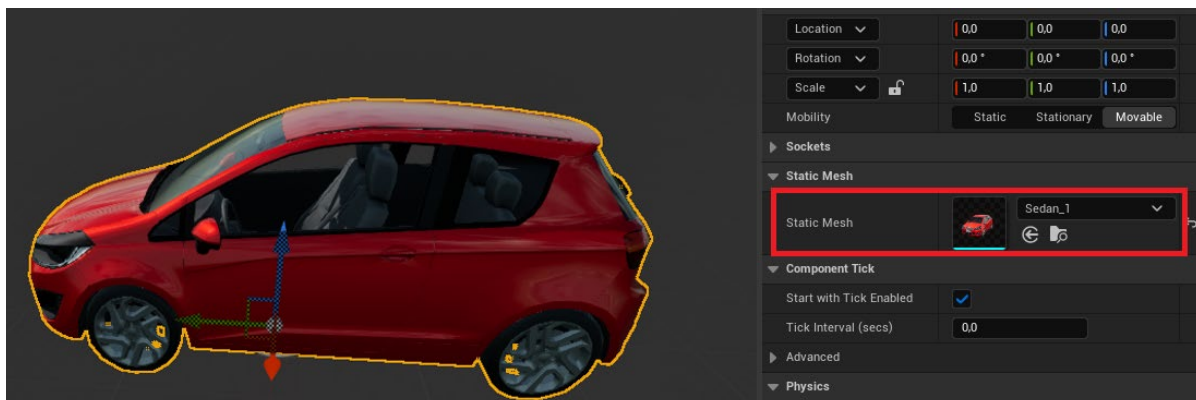

Рис. 2.7. Изменение параметра, отвечающего за используемую модель в *Static Mesh*

Модель, выбранная в данном поле, будет отображаться при запуске уровня, если иное поведение не задано в логике самого *Actor*. Для изменения логики работы объекта перейдите во вкладку *Event Graph*, предназначенную для визуального программирования и настройки поведения (рис. 2.8).

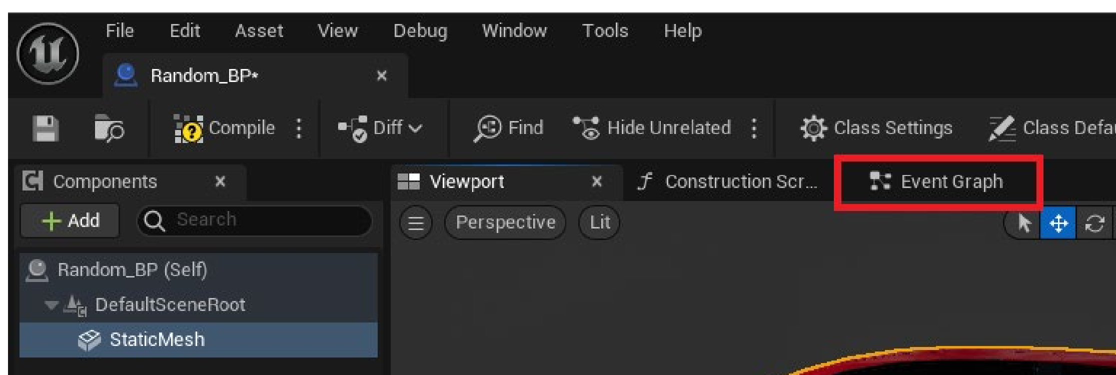

Рис. 2.8. Переход в режим *Event Graph*

Во вкладке *Event Graph* присутствуют стандартные события, которые по умолчанию не содержат реализованной логики. Одним из таких событий является *Event BeginPlay*, которое автоматически срабатывает при запуске уровня и может использоваться в качестве отправной точки для реализации различных сценариев, например случайной генерации объектов (рис. 2.9).

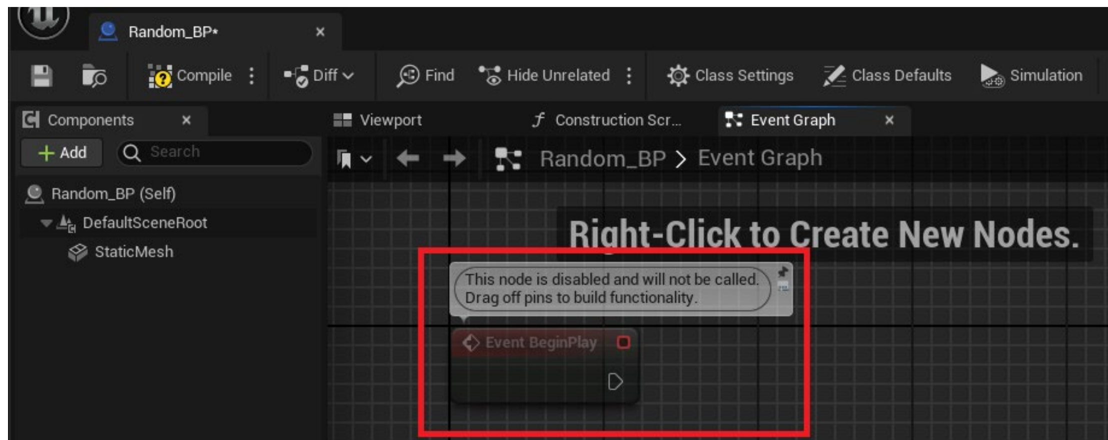

Рис. 2.9. Стандартное событие *Event BeginPlay*

Для реализации механизма случайного выбора моделей создайте переменную, в которой будет храниться набор ссылок на трехмерные объекты проекта. Назовите данную переменную *Array_StaticMeshes* (рис. 2.10).

По умолчанию создаваемая переменная в *Unreal Engine* имеет тип *Boolean* и предназначена для хранения одного значения, что не соответствует поставленной задаче. Поэтому измените ее тип и структуру хранения данных, чтобы обеспечить возможность работы с набором моделей.

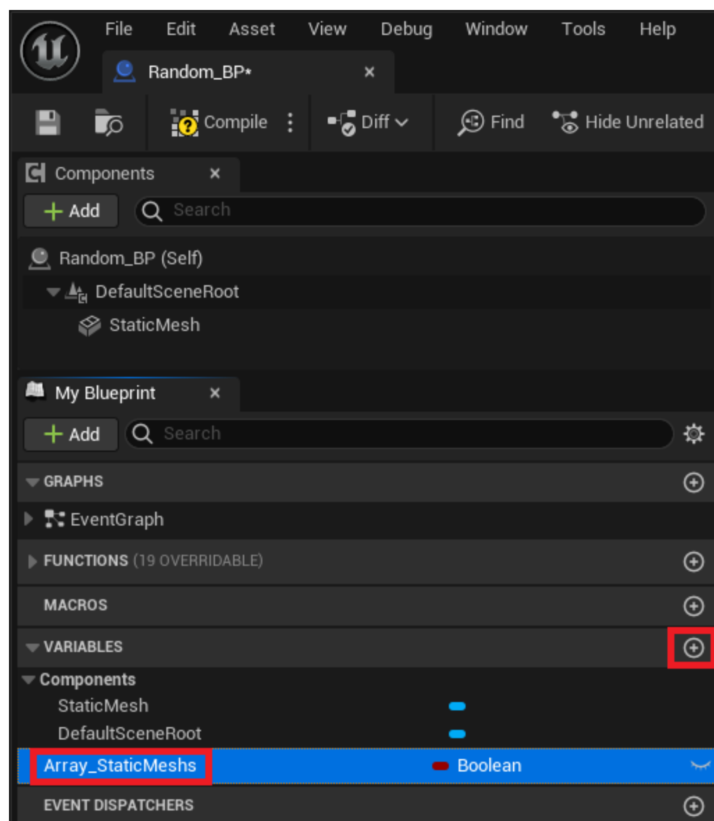

Рис. 2.10. Создание новой переменной *Array_StaticMeshes*

Для выполнения дальнейших действий измените структуру данных переменной, преобразовав ее из одиночного значения в массив (рис. 2.11). Для этого выберите переменную в списке переменных, затем в правой части окна, в панели *Details*, найдите параметр *Variable Type*, в котором задаются ее характеристики.

Далее измените тип структуры данных с *Single* на *Array*. Это позволит использовать переменную для хранения набора элементов, а не одного значения.

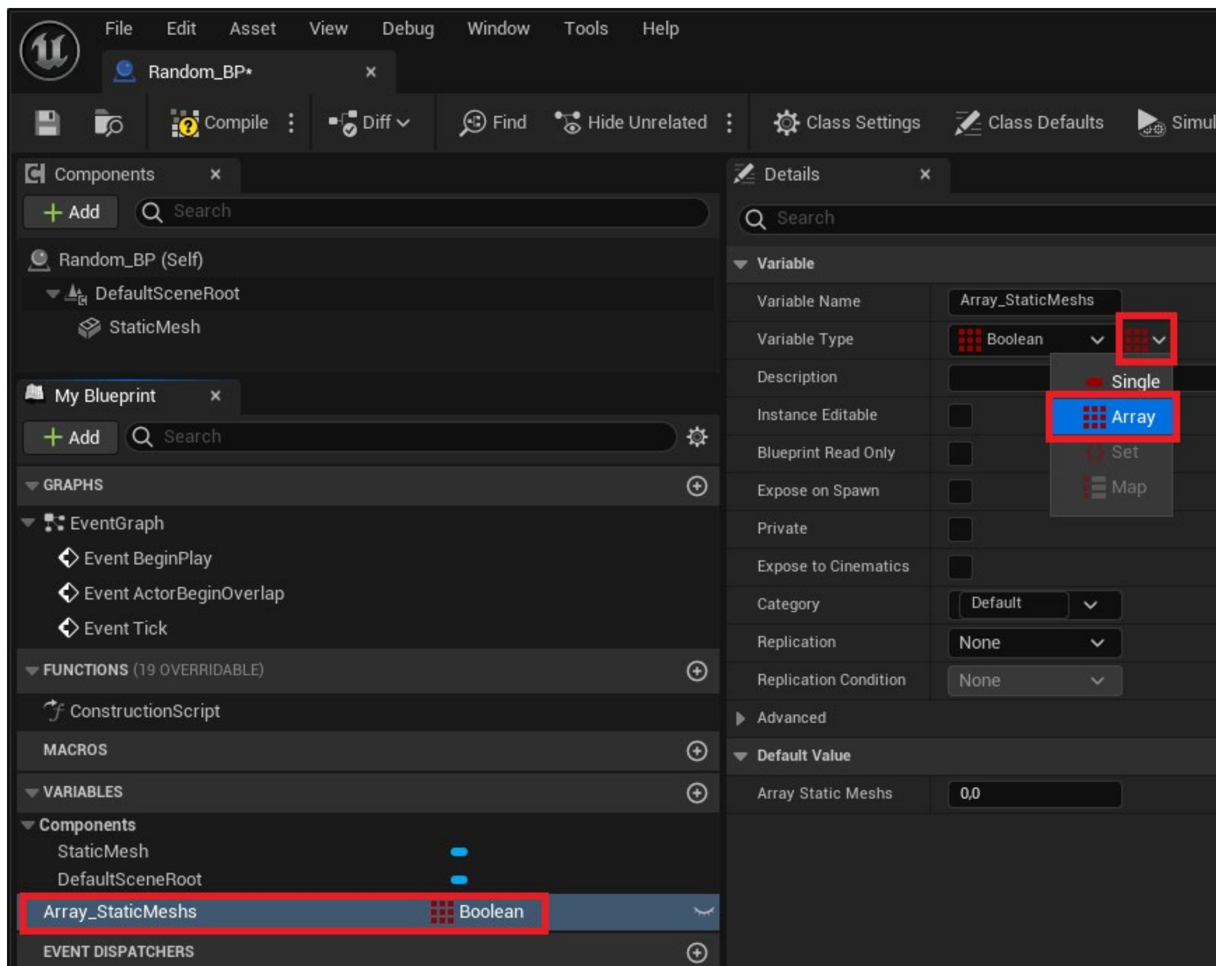

Рис. 2.11. Изменение структуры данных *Array_StaticMeshes*

Помимо изменения структуры данных задайте корректный тип данных переменной. Для этого выберите тип *Soft Object Reference (Static Mesh)*. Такой формат позволяет хранить ссылки на объекты без их непосредственной загрузки в память (рис. 2.12).

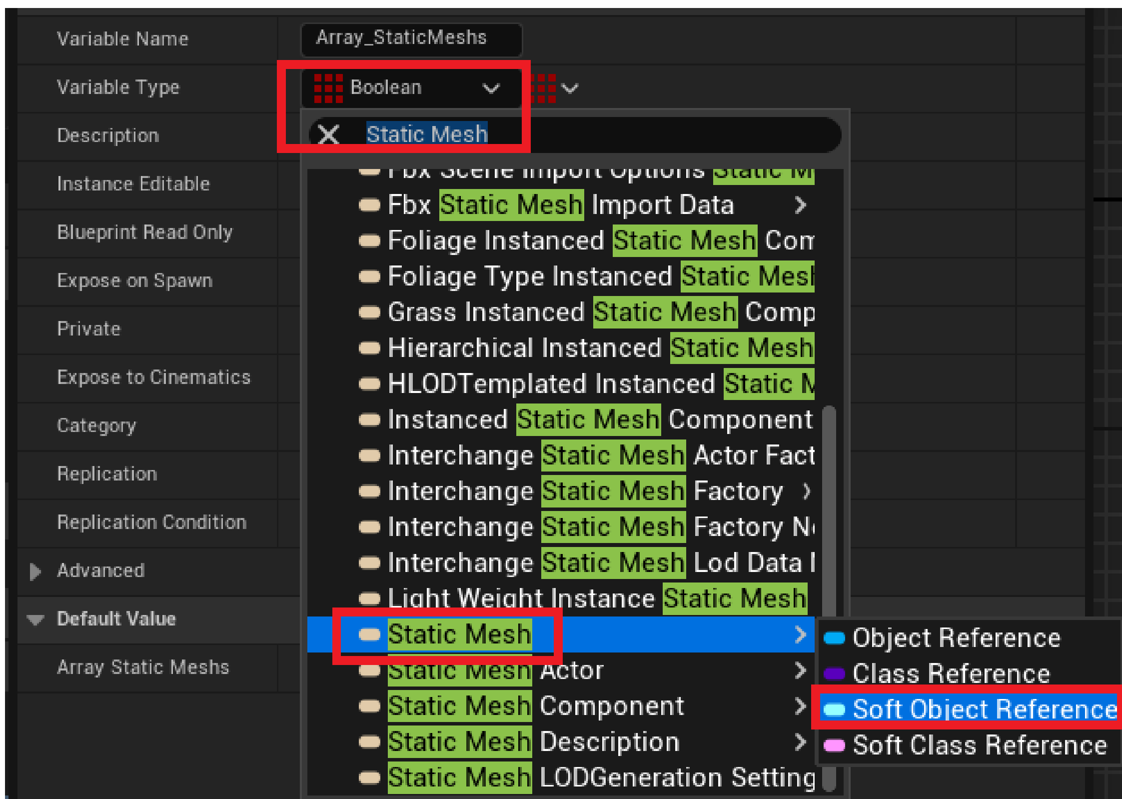

Рис. 2.12. Изменение типа данных *Array_StaticMeshes*

После изменения всех параметров переменной выполните компиляцию *Actor* (рис. 2.13). Это необходимо для применения внесенных изменений и обеспечения возможности дальнейшего редактирования переменной. После компиляции станет доступным добавление элементов в массив (рис. 2.14).

Добавьте новые элементы во вкладке *Array_StaticMeshes*, нажав на значок в виде плюса. После этого в массив можно добавлять необходимые ссылки на трехмерные модели.

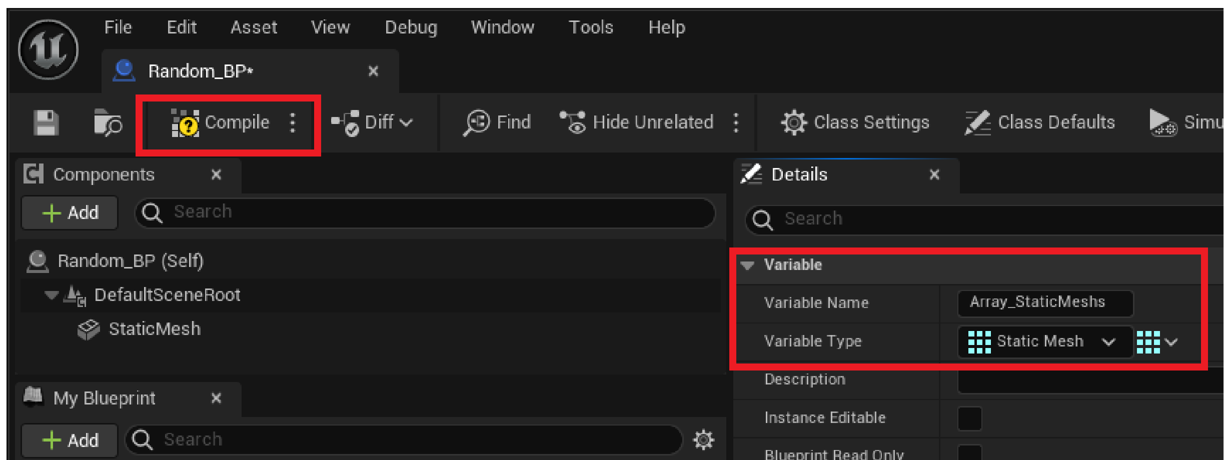

Рис. 2.13. Компиляция объекта с новой переменной *Array_StaticMeshes*

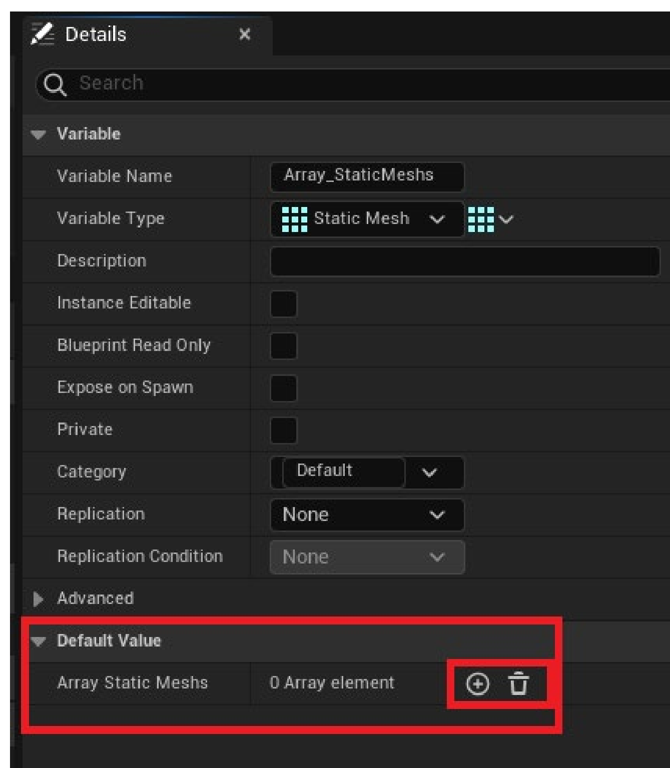

Рис. 2.14. Добавление новых элементов массива *Array_StaticMeshes*

Добавление элементов приводит к созданию пустых контейнеров с автоматически присвоенными индексами, начиная с нулевого (рис. 2.15). Каждый элемент массива представляет собой отдельную ячейку, в которую можно поместить ссылку на модель.

Заполните массив, перетаскивая нужные трехмерные модели из *Content Browser* в соответствующие ячейки массива (рис. 2.16).

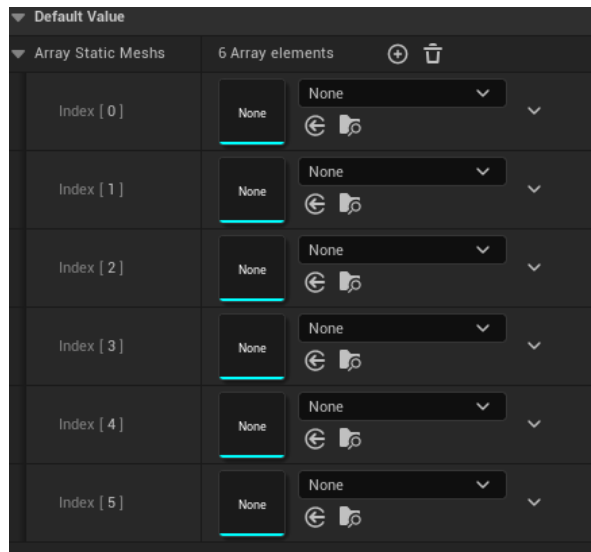

Рис. 2.15. Новые элементы массива *Array_StaticMeshes*

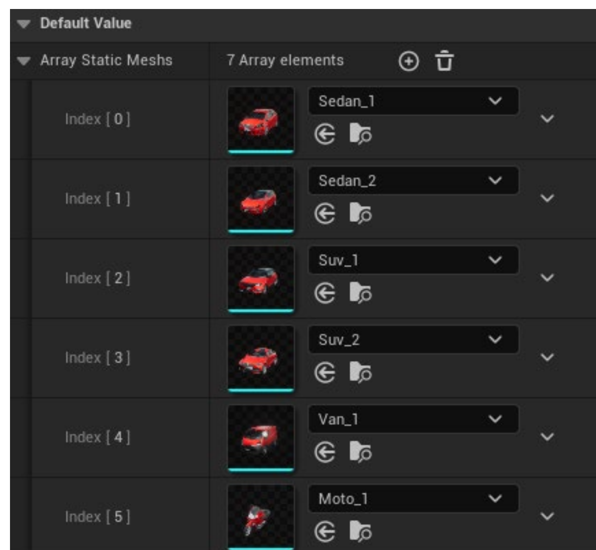

Рис. 2.16. Заполнение элементов массива *Array_StaticMeshes*

После заполнения массива обратитесь к созданной переменной *Array_StaticMeshes*. Для этого перетащите ее в рабочую область во вкладке *Event Graph*, удерживая левую кнопку мыши, и выберите операцию получения значения (*Get*) (рис. 2.17).

Рис. 2.17. Вызов переменной *Array_StaticMeshes*

Вызванная переменная *Array_StaticMeshes* содержит массив элементов. Для получения индекса последнего элемента используйте узел *Last Index* (рис. 2.18). Далее примените узел *Get* (*a copy*), который позволяет получить из массива копию элемента по указанному индексу (рис. 2.19).

Для генерации случайного значения типа *Integer* используйте узел *Random Integer In Range*, который возвращает случайное число в заданном диапазоне (рис. 2.20).

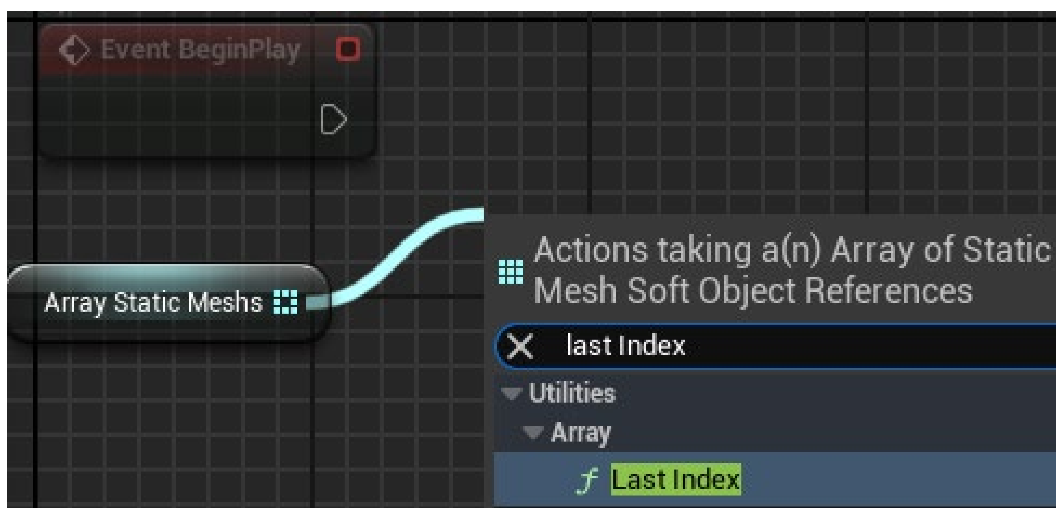

Рис. 2.18. Вызов узла *Last Index* из переменной *Array_StaticMeshes*

Рис. 2.19. Вызов узла *Get* (*a copy*) из переменной *Array_StaticMeshes*

Рис. 2.20. Вызов узла *Random Integer In Range*

После вызова всех трех указанных узлов соедините их между собой, как показано на рис. 2.21.

Используйте индекс последнего элемента массива для определения верхней границы диапазона в узле *Random Integer In Range*. Полученное случайное число передайте в узел *Get* (*a copy*), который возвращает копию элемента массива, соответствующего выбранному случайному индексу. Таким образом обеспечьте случайный выбор модели из массива.

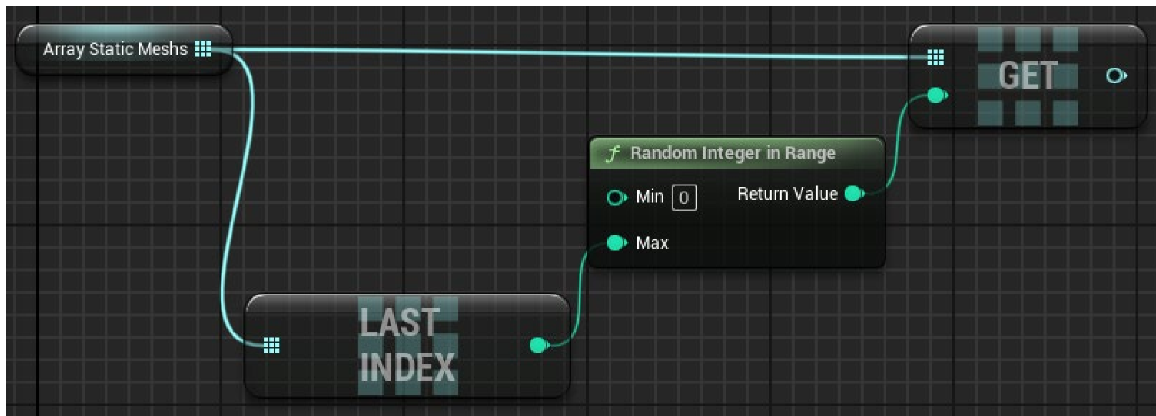

Рис. 2.21. Объединение группы узлов для вызова случайного элемента массива

Далее используйте набор узлов *Async Load Asset* (рис. 2.22) и *Cast To StaticMesh* (рис. 2.23).

Используйте узел *Async Load Asset* для асинхронной загрузки ресурса по ссылке, что позволяет загружать объект в память без блокировки выполнения программы. Затем примените узел *Cast To Static Mesh* для приведения загруженного объекта к типу статической модели перед его установкой в компонент.

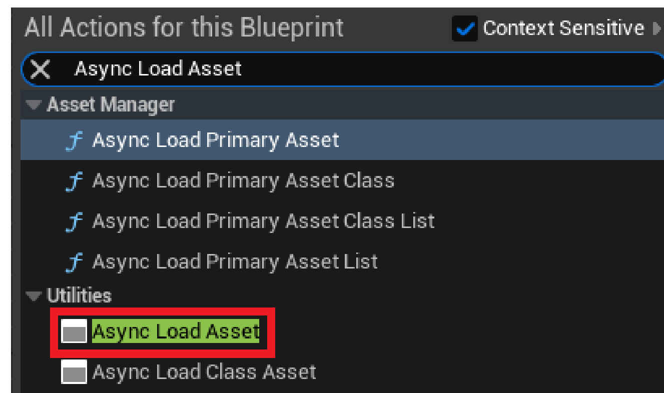

Рис. 2.22. Вызов узла *Async Load Asset*

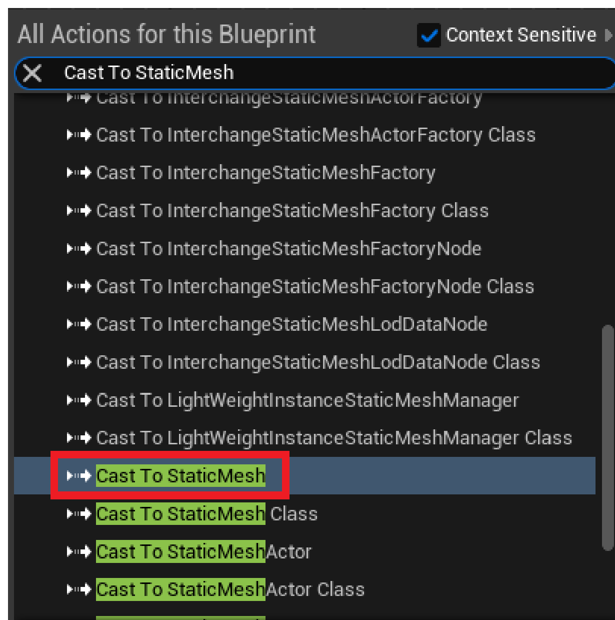

Рис. 2.23. Вызов узла *Cast To StaticMesh*

После получения объекта, который будет размещен на уровне, и его загрузки в память следует заменить содержимое компонента *Static Mesh*, созданного ранее в *Random_BP*. Для этого используйте узел *Set Static Mesh* (рис. 2.24). В списке он может отображаться с указанием имени соответствующего компонента в скобках.

Если имя компонента *Static Mesh* не изменялось, узел будет отображаться с подписью (*Static Mesh*), что указывает на стандартное имя компонента. В случае переименования компонента в скобках будет указано новое имя, соответствующее заданному пользователем.

После добавления всех необходимых узлов разместите их последовательно в порядке выполнения логики (рис. 2.25). Затем соедините узлы между собой: ссылка на объект передается из узла *Get* (*a copy*), а запуск всей последовательности выполняется через событие *Event BeginPlay* (рис. 2.26).

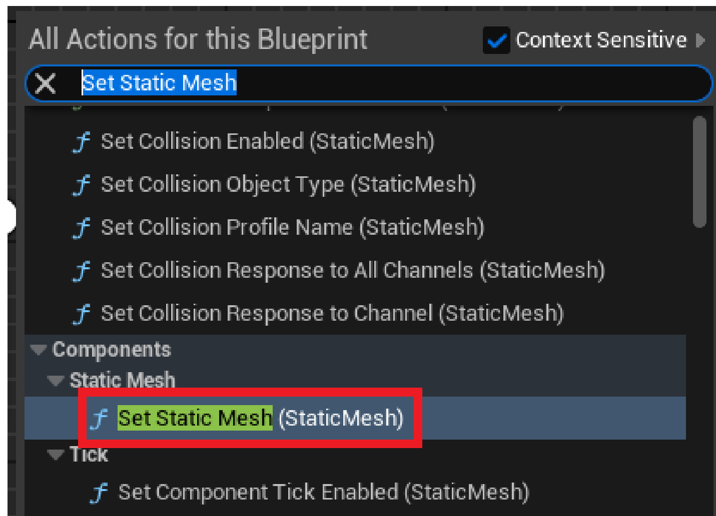

Рис. 2.24. Вызов узла *Set Static Mesh*

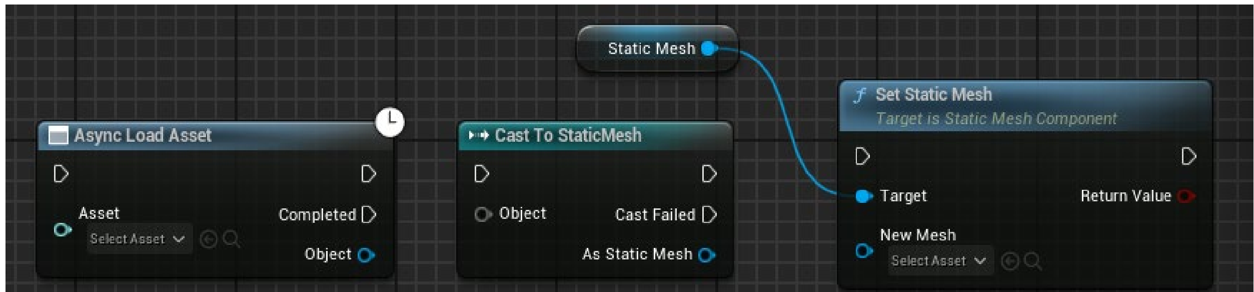

Рис. 2.25. Узлы для замены контейнера *Static Mesh* случайной моделью из массива

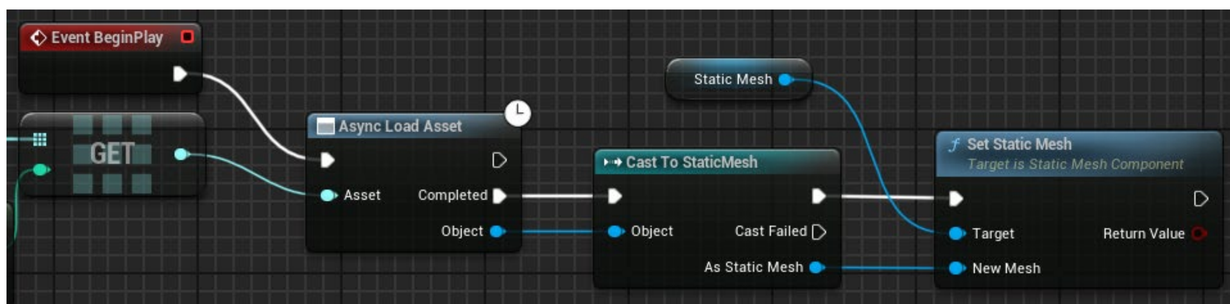

Рис. 2.26. Соединение узлов замены контейнера *Static Mesh* случайной моделью из массива

Итоговую логику работы алгоритма выбора случайной модели (рис. 2.27) скомпилируйте и сохраните. После успешного сохранения разместите объект *Random_BP* на уровне, перетащив его на сцену с помощью удержания левой кнопки мыши (рис. 2.28).

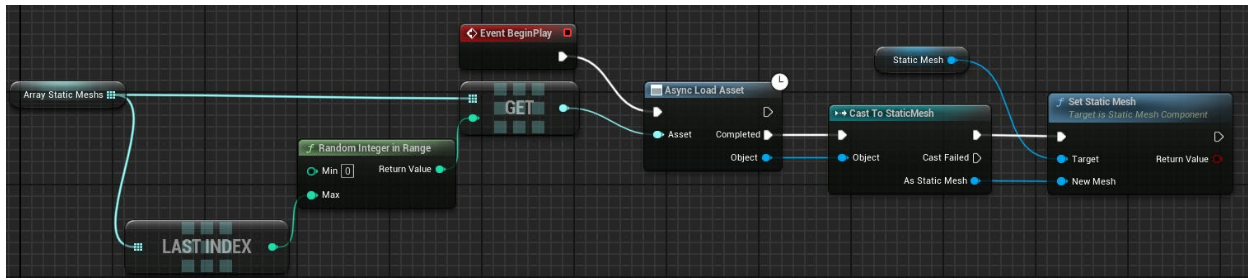

Рис. 2.27. Итоговая структура кода для случайного выбора модели из массива

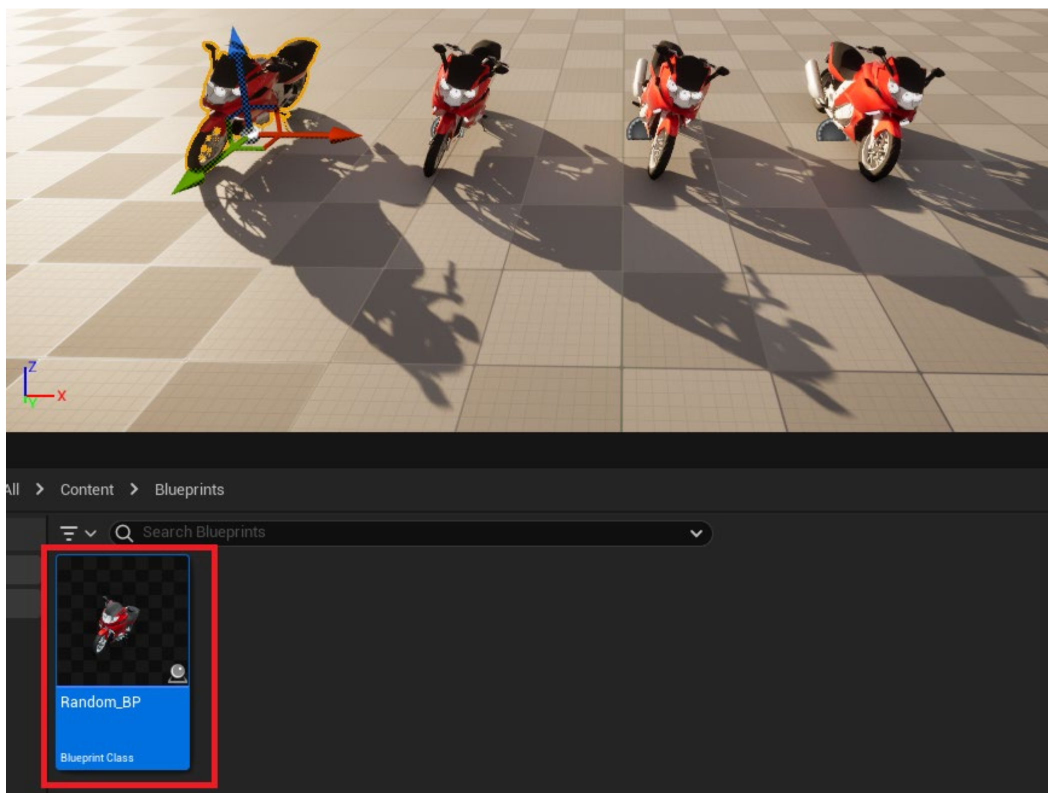

Рис. 2.28. Размещение объекта *Random_BP* на уровне

Разместите объекты на уровне, после чего приступите к тестированию сцены (рис. 2.29) и наблюдайте результаты работы алгоритма случайного выбора моделей (рис. 2.30).

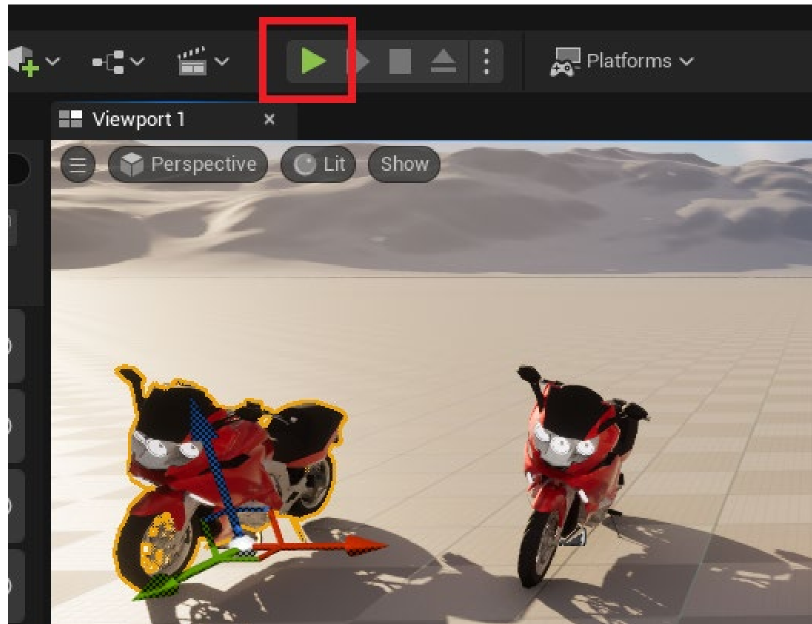

Рис. 2.29. Панель запуска уровня

Рис. 2.30. Результат случайной генерации

При создании объекта *Random_BP* ранее могла быть задана модель по умолчанию для компонента *Static Mesh*. Для реализации возможности отображения пустых ячеек удалите ссылку на выбранную модель в соответствующем поле компонента *Static Mesh* (рис. 2.31).

В результате объекты *Random_BP* в редакторе будут отображаться как пустые, при этом их наличие будет обозначаться только специальным значком (рис. 2.32).

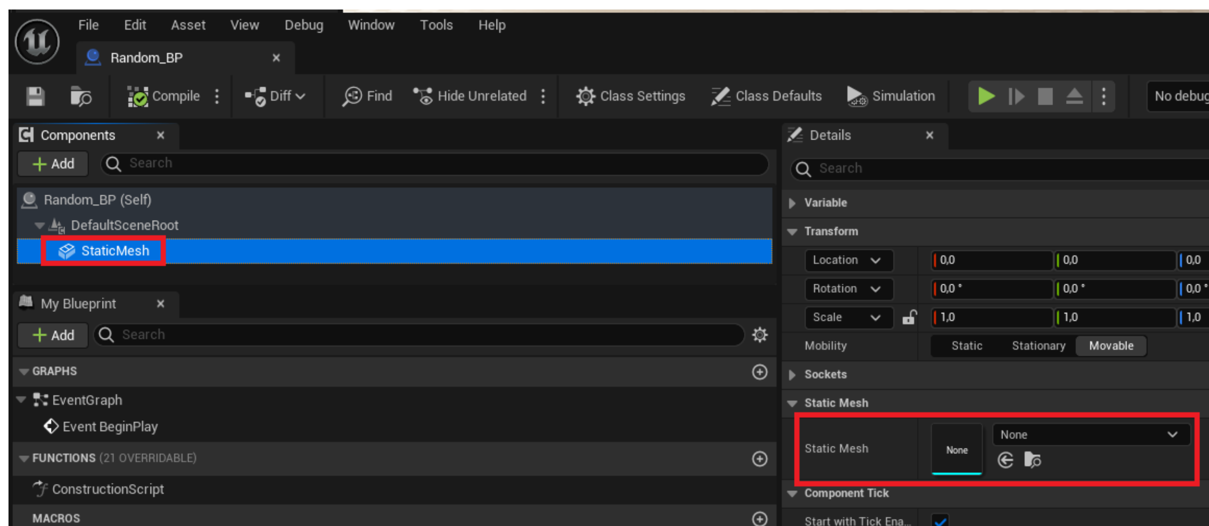

Рис. 2.31. Очищение контейнера *Static Mesh* стандартной модели

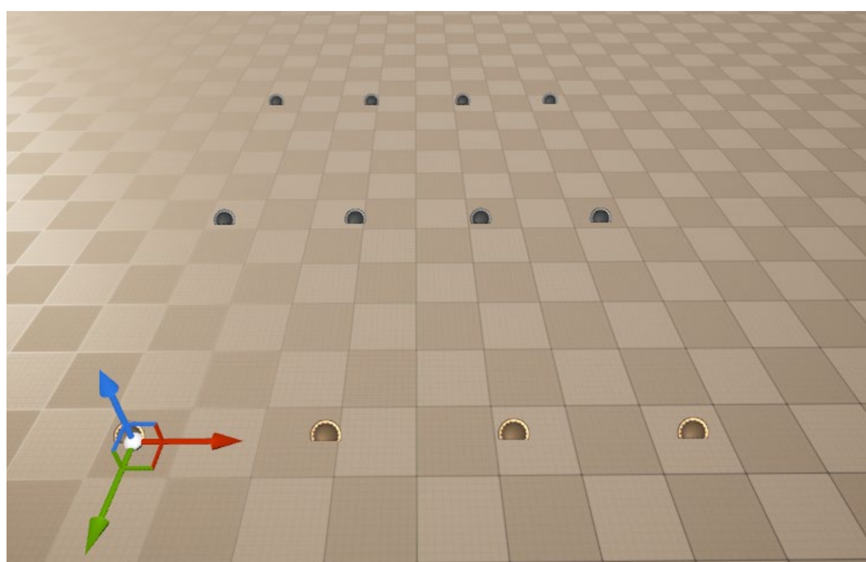

Рис. 2.32. Изменение отображения *Random_BP* в окне редактора

В массив ссылок на объекты могут быть добавлены пустые элементы (рис. 2.33). Это позволяет при генерации оставлять отдельные позиции без заданной модели, если изначально компонент *Static Mesh* был пустым (рис. 2.34).

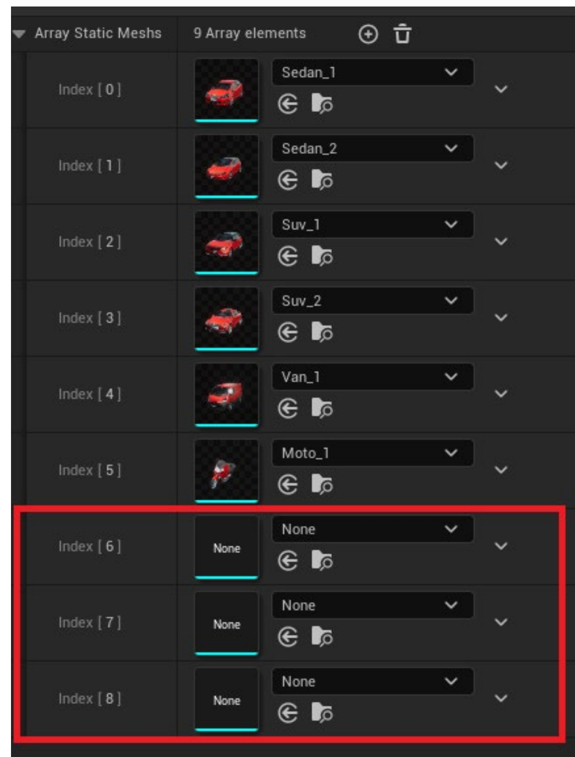

Рис. 2.33. Добавление пустых элементов в массив *Array_StaticMeshes*

Рис. 2.34. Результат случайной генерации с пустыми элементами

Простой *Actor*, предназначенный для генерации случайных моделей, можно расширить и превратить в более универсальный инструмент – объект, в котором пользователь сможет самостоятельно выбирать необходимые модели из заранее заданного массива.

Для реализации отображения пользователю информации о возможности ручной замены трехмерной модели создайте специальный *Widget* в соответствующей папке *Content Browser*. Для этого щелкните правой кнопкой мыши по пустой области и выберите пункт создания нового элемента (рис. 2.35).

Данный *Widget* будет отображаться при приближении пользователя к объекту и содержать текстовую подсказку о доступном взаимодействии. Кроме того, настройте *Widget* таким образом, чтобы он всегда был ориентирован лицевой стороной к персонажу пользователя независимо от его положения в пространстве.

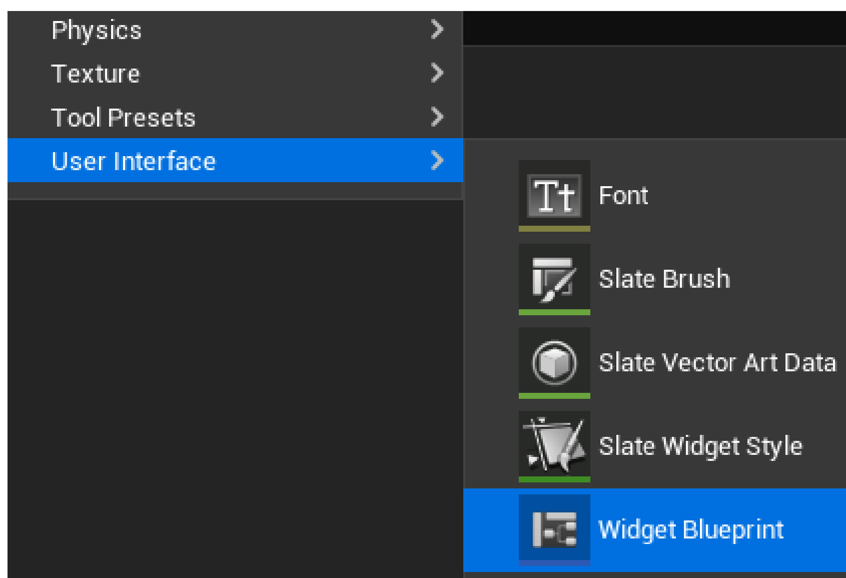

Рис. 2.35. Создание элемента *Widget*

Сохраните созданный виджет под именем *Replacing_model* (рис. 2.36). Откройте его двойным щелчком левой кнопкой мыши, чтобы перейти в окно редактирования виджетов.

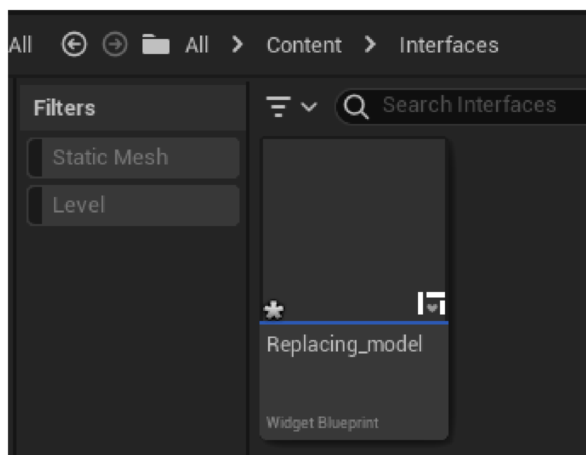

Рис. 2.36. Созданный виджет *Replacing_model*

Далее перейдите во вкладку *Palette*, расположенную в левой части экрана, и найдите элемент *Canvas Panel*. Добавьте его на рабочую область, чтобы задать базовую структуру интерфейса виджета (рис. 2.37).

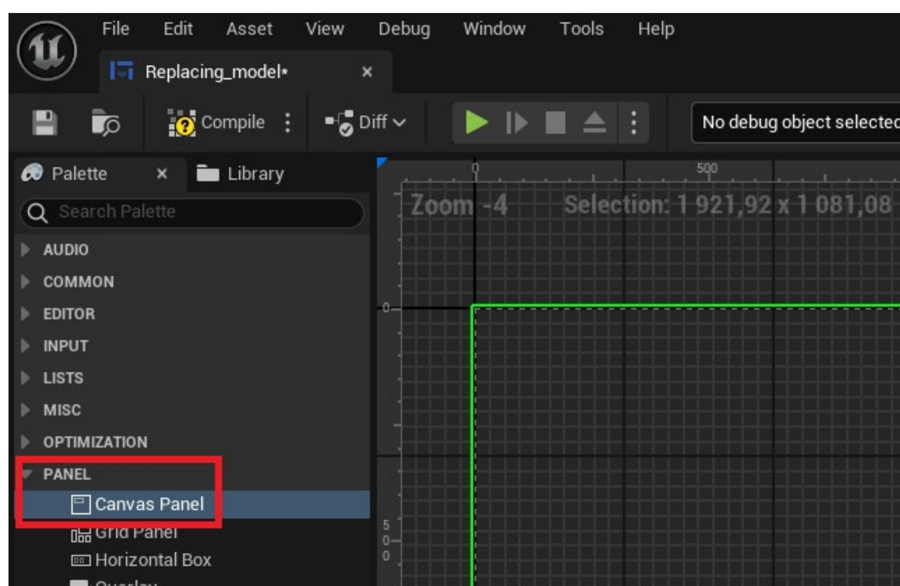

Рис. 2.37. Добавление *Canvas Panel*

Найдите элемент *Text* и добавьте его на рабочую область внутри *Canvas Panel* (рис. 2.38). В параметрах элемента перейдите во вкладку *Anchors* и установите выравнивание по центру, чтобы обеспечить корректное отображение текста на экране.

Далее найдите параметр, отвечающий за содержимое текстового блока. Он расположен во вкладке *Content* и называется *Text*. Заполните данный параметр информацией о возможности изменения трехмерной модели с помощью нажатия клавиш от 0 до 4 либо укажите любой другой текст, соответствующий логике работы вашего проекта.

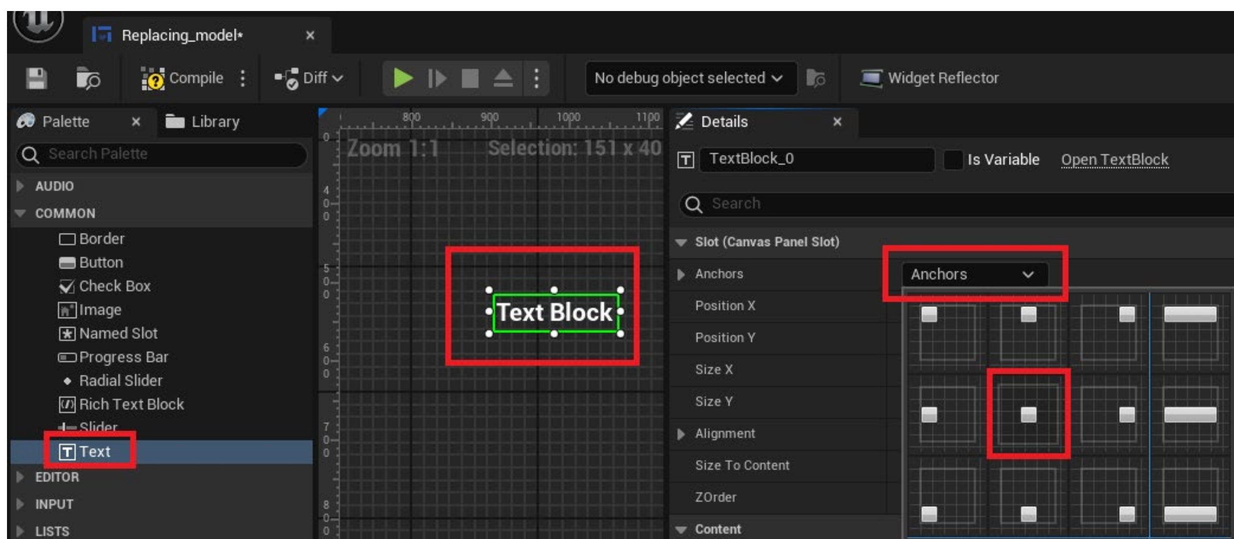

Рис. 2.38. Добавление *Text*

Добавьте виджет в качестве отдельного компонента внутри нового объекта. В качестве основы используйте ранее созданный объект *Random_BP*, так как он уже содержит логику выбора и отображения трехмерных моделей.

Чтобы избежать повторного создания всей структуры, создайте копию существующего *Actor* и внесите в нее необходимые изменения (рис. 2.39). Для этого вызовите контекстное меню, щелкнув правой кнопкой мыши по объекту, и выберите пункт *Duplicate*. Полученную копию переименуйте в *Replacing_BP*.

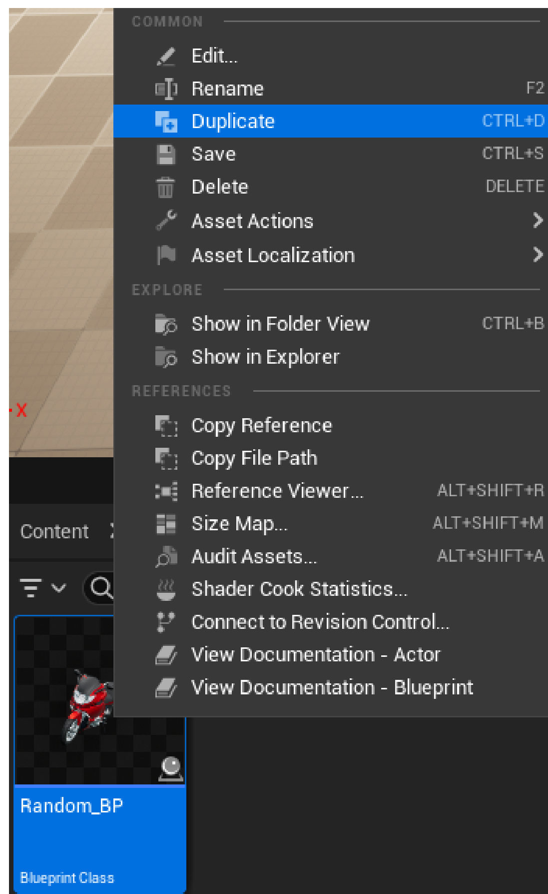

Рис. 2.39. Создание копии *Random_BP*

Двойным щелчком левой кнопкой мыши откройте окно редактирования *Replacing_BP*. В открывшемся интерфейсе перейдите во вкладку *Components* и добавьте компонент *Box Collision* для объекта *Replacing_BP* (рис. 2.40).

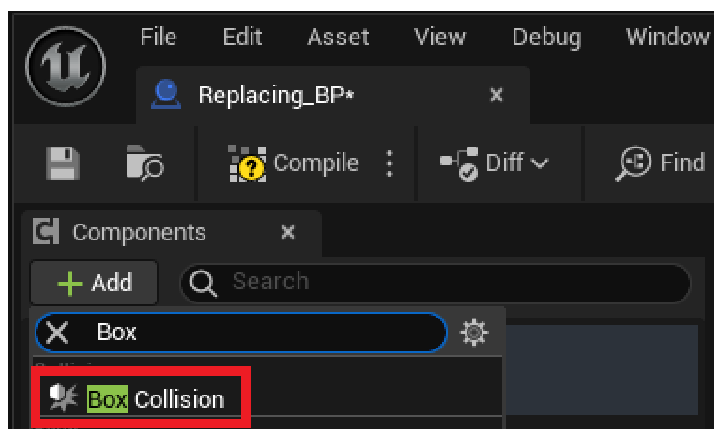

Рис. 2.40. Добавление нового компонента *Box Collision*

Далее настройте размеры *Box Collision* таким образом, чтобы он покрывал область размещения объектов с запасом, обеспечивая корректное взаимодействие с персонажем (рис. 2.41). Для удобства настройки временно назначьте модель в компонент *Static Mesh*, чтобы визуально оценить пропорции и размеры *Box Collision* относительно объекта. После завершения настройки удалите или замените модель.

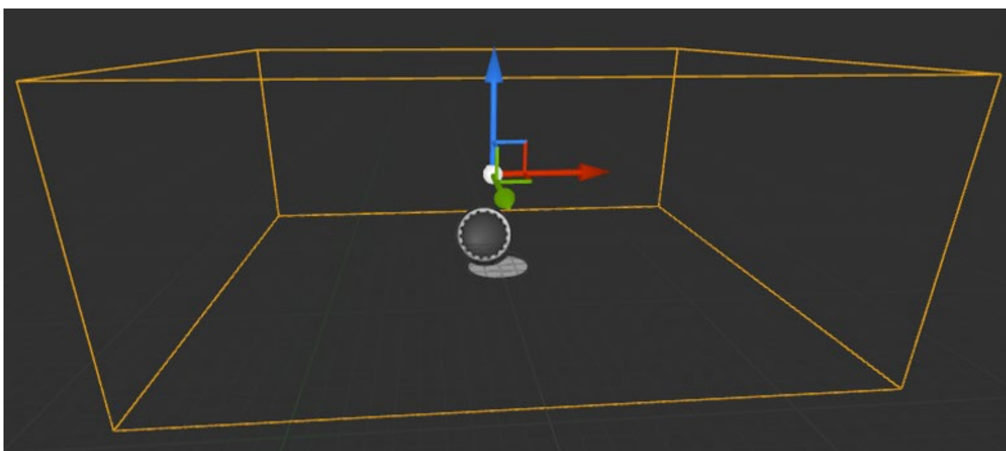

Рис. 2.41. Изменение размеров *Box Collision*

Добавьте компонент *Widget* для объекта *Replacing_BP* (рис. 2.42). В настройках данного компонента найдите параметр *Space* и установите его значение в режим *Screen*. Далее в поле *Widget Class* выберите созданный ранее виджет *Replacing_model* (рис. 2.43).

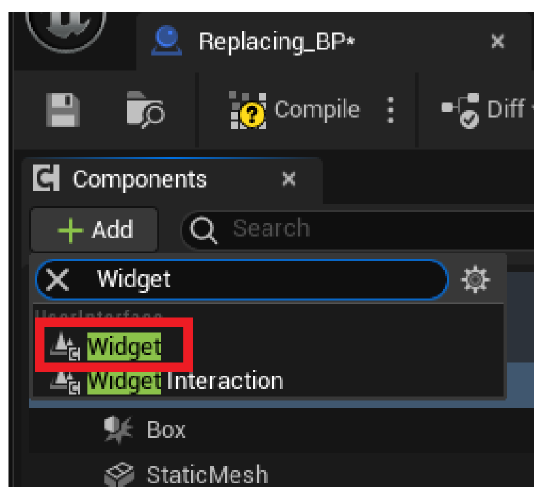

Рис. 2.42. Добавление нового компонента *Widget*

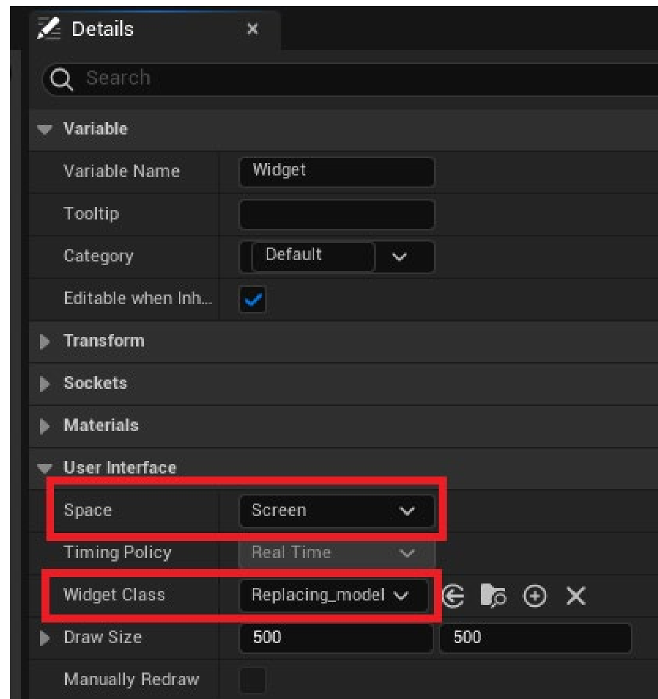

Рис. 2.43. Изменение принципов отображения *Widget*

После этого отключите видимость компонента *Widget* (рис. 2.44). По умолчанию он должен быть скрыт, чтобы не отображаться постоянно. Его отображение будет включаться только при входе персонажа в область действия *Box Collision*.

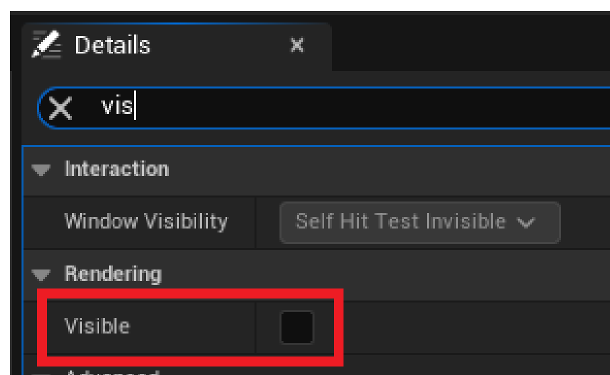

Рис. 2.44. Изменение параметра видимости *Widget*

Поднимите компонент *Widget* на высоту, удобную для восприятия пользователем (рис. 2.45). При корректной настройке виджет будет автоматически

ориентироваться в сторону камеры персонажа и отображаться на заданной высоте.

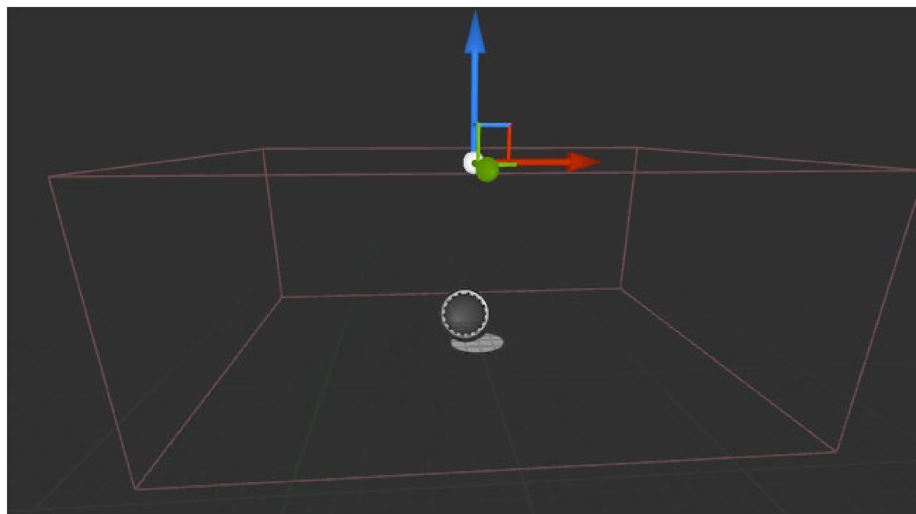

Рис. 2.45. Перемещение компонента *Widget*

Компонент *Box Collision* способен генерировать события начала и окончания пересечения его границ. Добавьте в логику объекта событие *Begin Overlap*, которое срабатывает при входе персонажа в область действия *Box Collision*, и событие *End Overlap*, которое срабатывает при выходе из этой области (рис. 2.46).

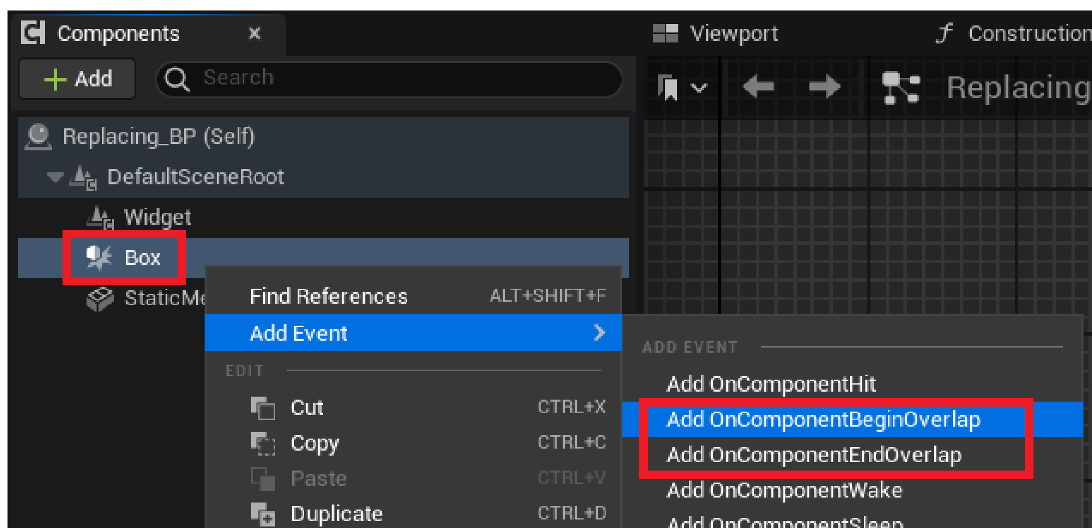

Рис. 2.46. Создание событий пересечения для *Box Collision*

Пересечение области *Box Collision* должно запускать логику отображения виджета в заданной точке. Для этого используйте два узла *Set Visibility* (*Widget*) (рис. 2.47).

При срабатывании события *Begin Overlap* установите значение *true* в узле *Set Visibility*, обеспечив отображение виджета. При срабатывании события *End Overlap* установите значение *false*, скрывая виджет при выходе персонажа из зоны действия. Соедините соответствующие события и узлы в логической последовательности (рис. 2.48).

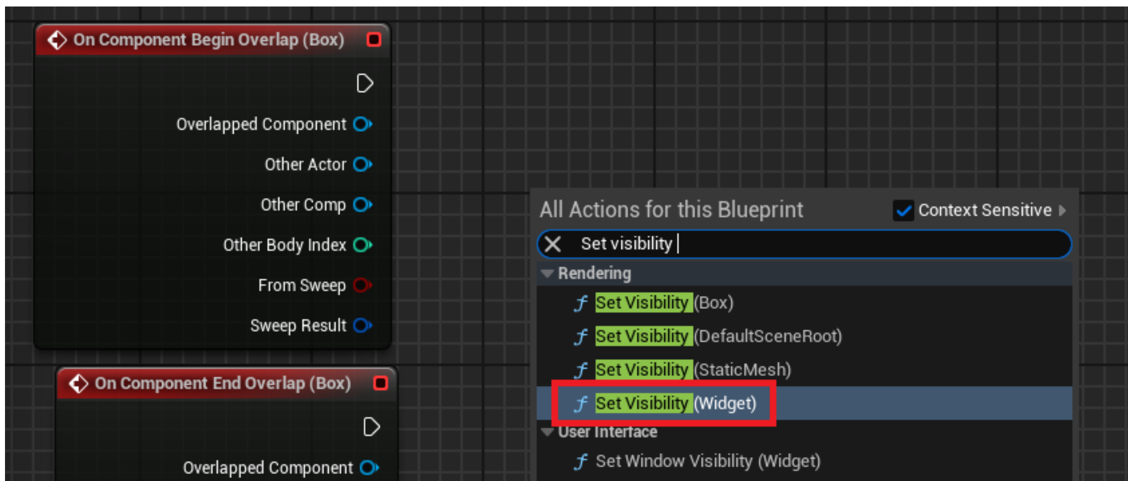

Рис. 2.47. Добавление узла *Set Visibility* для компонента (*Widget*)

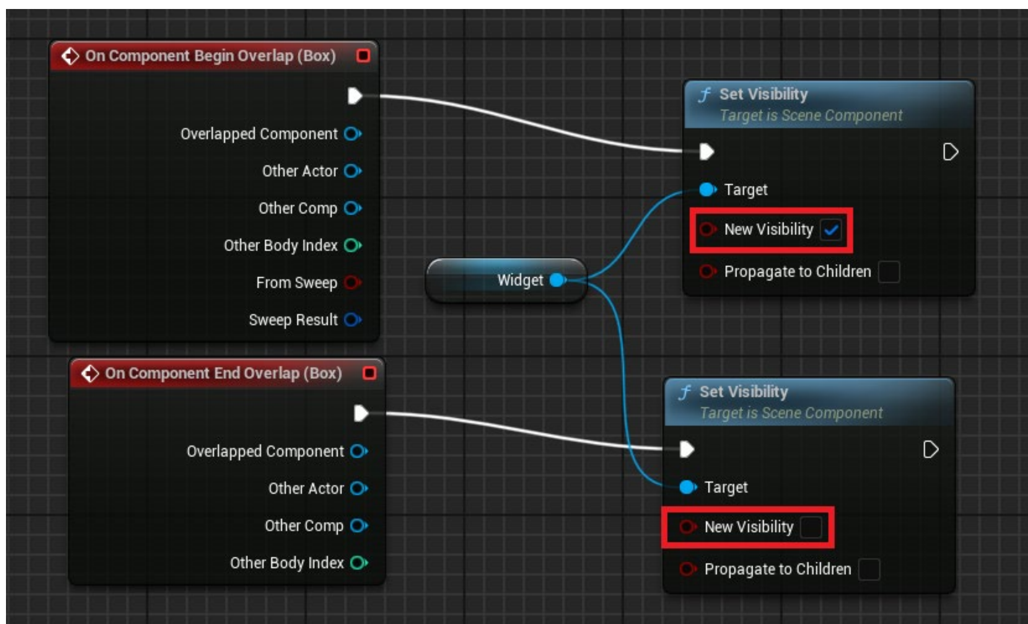

Рис. 2.48. Соединение событий пересечения и узлов *Set Visibility*

Для управления возможностью взаимодействия с объектом добавьте узлы *Enable Input* и *Disable Input*, которые позволяют управлять тем, будет ли данный объект обрабатывать действия пользователя или игнорировать их (рис. 2.49).

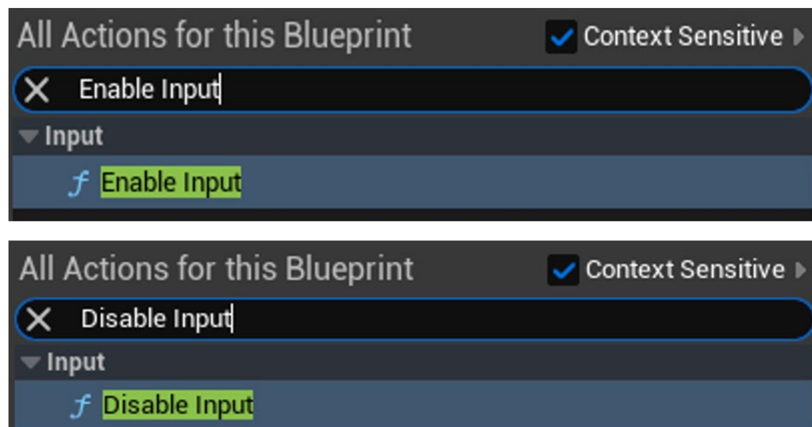

Рис. 2.49. Добавление узлов *Enable Input* и *Disable Input*

Узлам *Enable Input* и *Disable Input* для корректной работы требуется ссылка на текущий *Player Controller*. Для этого добавьте узел *Get Player Controller* (рис. 2.50).

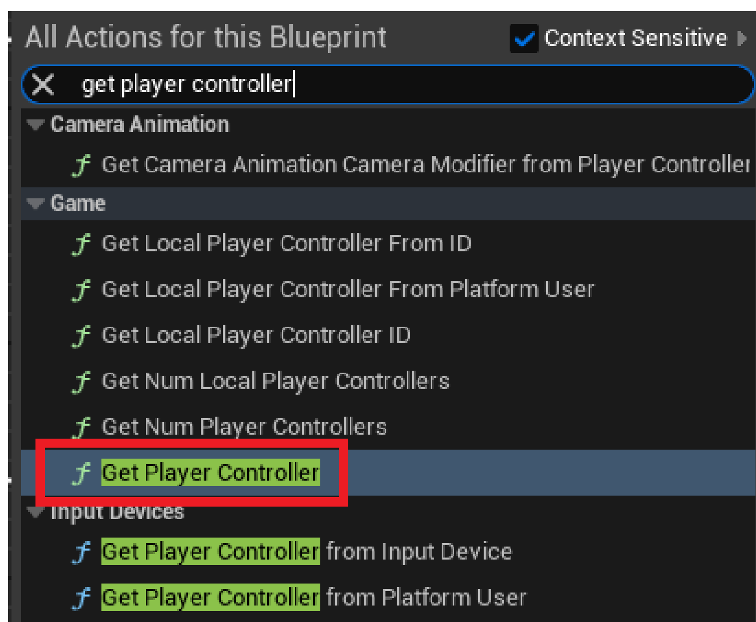

Рис. 2.50. Добавление узла *Get Player Controller*

После добавления узлов *Enable Input*, *Disable Input* и получения ссылки на *Player Controller* соедините их с ранее созданными событиями и узлами управления видимостью объекта. Обеспечьте, чтобы при выходе персонажа из зоны действия происходило отключение как видимости виджета, так и возможности ввода (рис. 2.51).

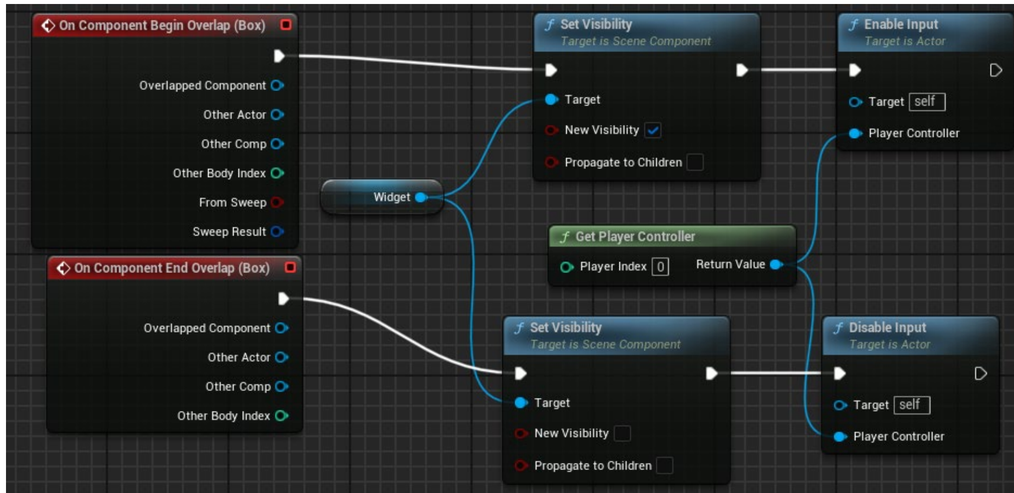

Рис. 2.51. Реализация отображения виджета

Щелкните правой кнопкой мыши по пустому пространству окна графа и найдите раздел *Keyboard Events* (рис. 2.52). Используйте данные события для обработки нажатий клавиш на клавиатуре. В данном случае создайте события для клавиш с цифрами от 0 до 4.

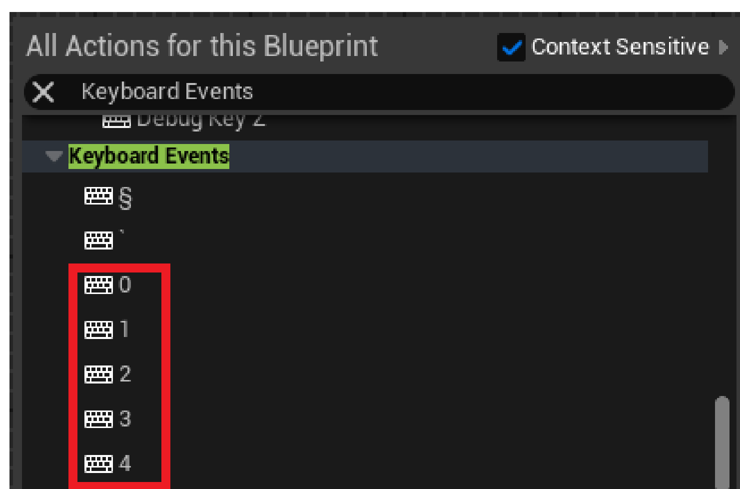

Рис. 2.52. Добавление событий *Keyboard Events* по нажатию клавиш

Создайте переменную типа *Integer*, предназначенную для хранения идентификатора (ID) выбранной пользователем модели. Данную переменную назовите *ID_Mesh* (рис. 2.53).

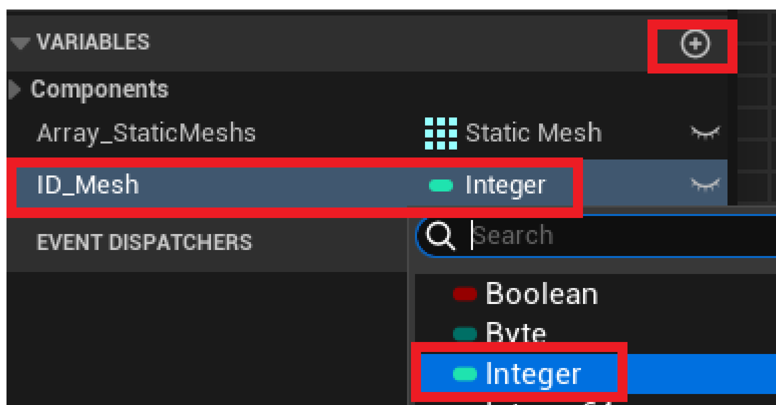

Рис. 2.53. Создание новой переменной *ID_Mesh*

Вызовите переменную *ID_Mesh* для последующего присвоения ей значений (рис. 2.54).

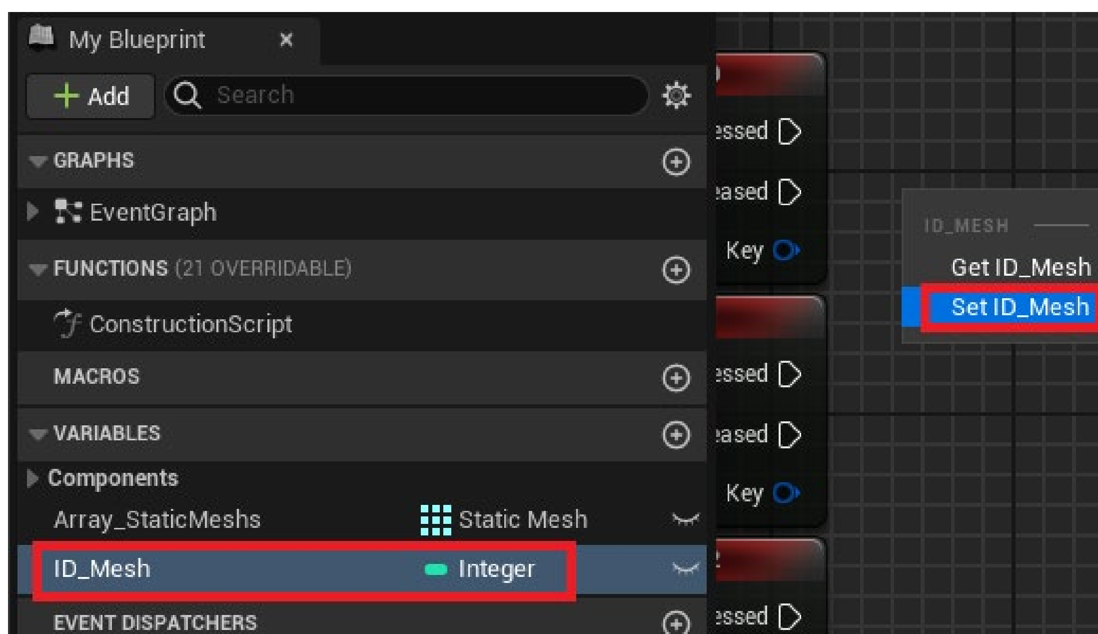

Рис. 2.54. Вызов переменной *ID_Mesh* для заполнения

Значение переменной *ID_Mesh* должно изменяться в зависимости от нажатия пользователем клавиш с цифрами от 0 до 4, принимая соответствующие значения. Для реализации данного поведения используйте узлы *Set ID_Mesh*, размещая их напротив соответствующих событий клавиатуры. Для каждого узла задайте свое значение, соответствующее выбранной клавише (рис. 2.55).

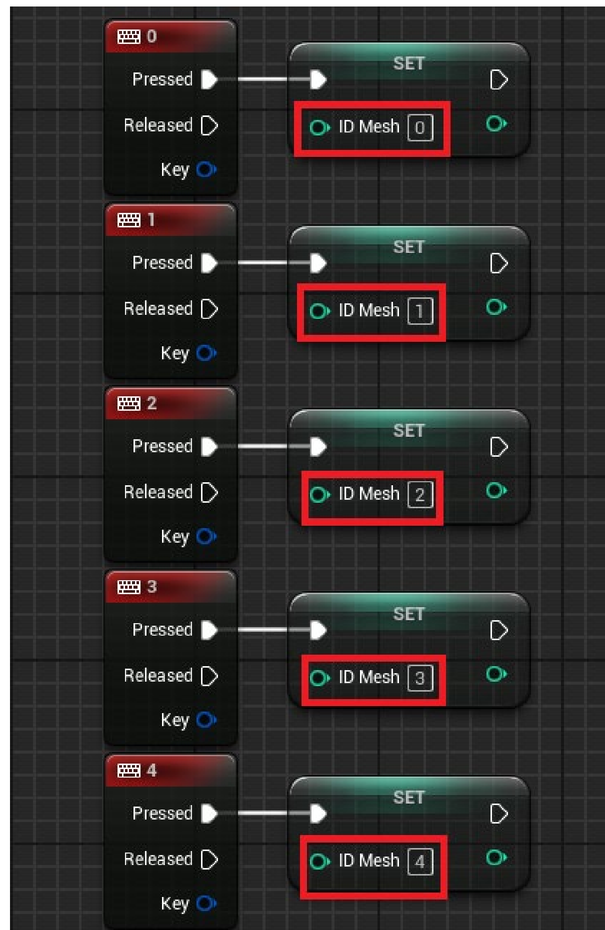

Рис. 2.55. Связь событий клавиатуры и переменной *ID_Mesh*

Создайте собственное событие с помощью узла *Add Custom Event* (рис. 2.56). После создания переименуйте его: для этого выделите событие левой кнопкой мыши и измените его название в панели параметров. Новое имя события задайте как *Update_SM* (рис. 2.57).

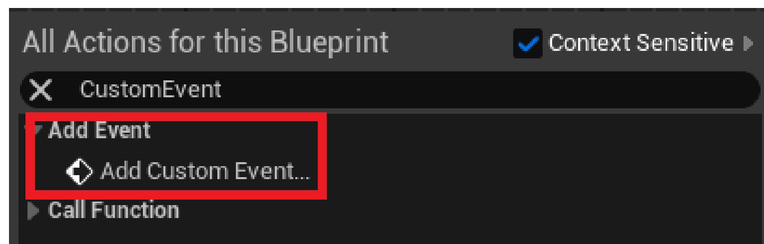

Рис. 2.56. Создание собственного события при помощи узла *Add Custom Event*

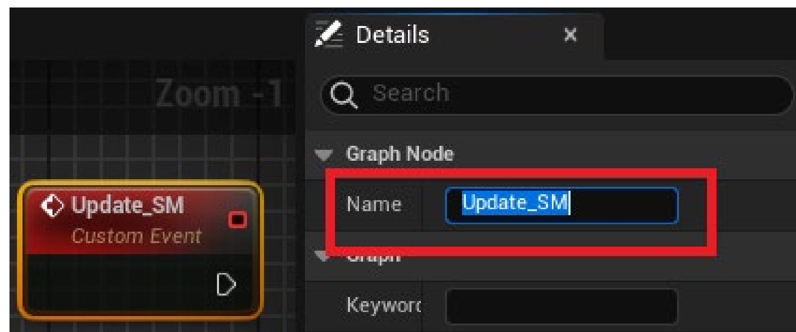

Рис. 2.57. Переименование *Custom Event* в *Update_SM*

Данное событие должно вызываться после ключевого изменения состояния системы. В этом случае таким изменением является обновление значения переменной *ID_Mesh*. Найдите место, где изменяется данная переменная, и свяжите выполнение события *Update_SM* сразу после этого действия (рис. 2.58).

Функция *Update_SM* запускает событие, которое, в свою очередь, активирует цепочку действий, отвечающую за замену трехмерной модели. На данном этапе к событию *Update_SM* не подключена никакая логика. В качестве основы ранее использовалась реализация объекта *Random_BP*, выполняющего случайный выбор модели из массива при запуске уровня. Данная логика сохраняется без изменений, однако создается ее копия для дальнейшей модификации.

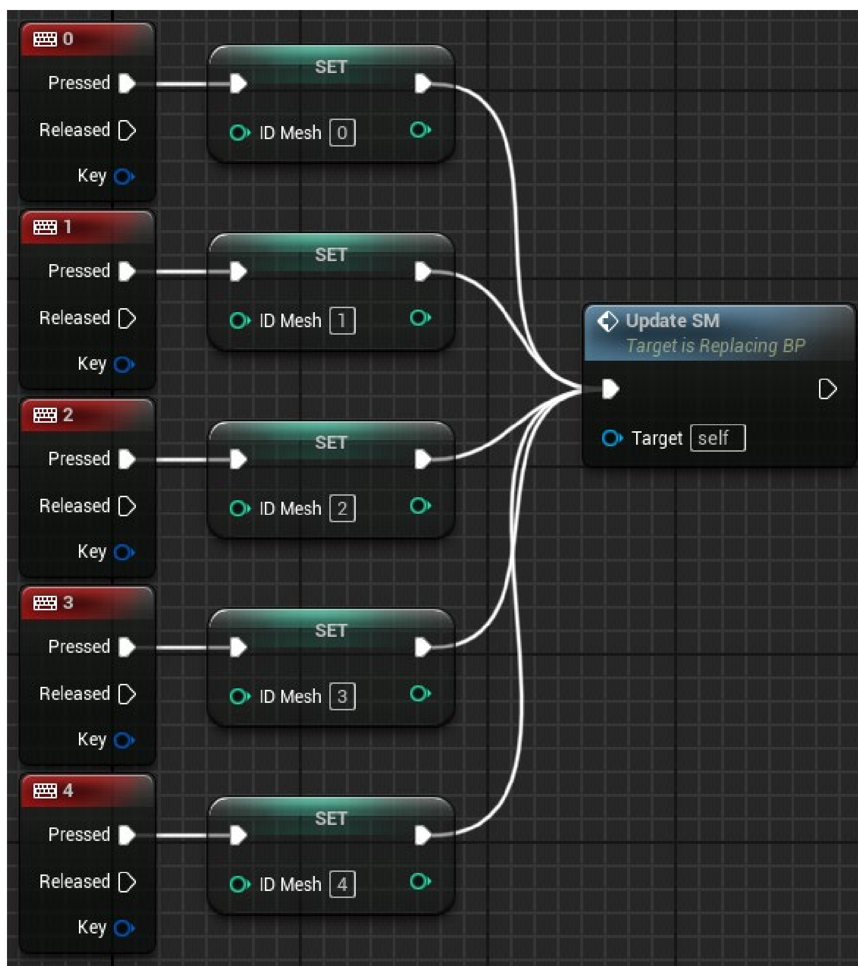

Рис. 2.58. Запуск функции *Update_SM*

В копии логики удалите узлы, отвечающие за генерацию случайного индекса массива, и замените событие *Event BeginPlay* на созданное событие *Update_SM* (рис. 2.59).

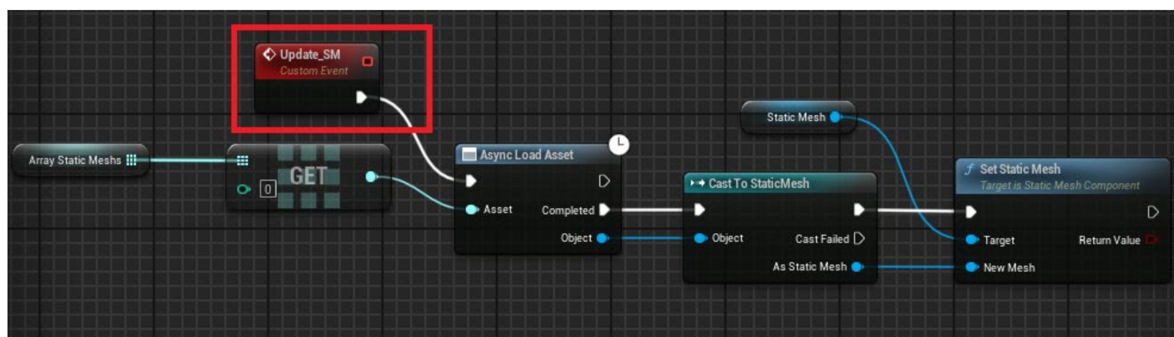

Рис. 2.59. Запуск события *Update_SM* для смены модели

Важно отметить, что логика случайной генерации при запуске уровня сохраняется и выполняется по событию *Event BeginPlay*, тогда как ветка, связанная с *Update_SM*, представляет собой модифицированную копию с внесенными изменениями. Одним из таких изменений является использование значения переменной *ID_Mesh* вместо случайно сгенерированного числа (рис. 2.60).

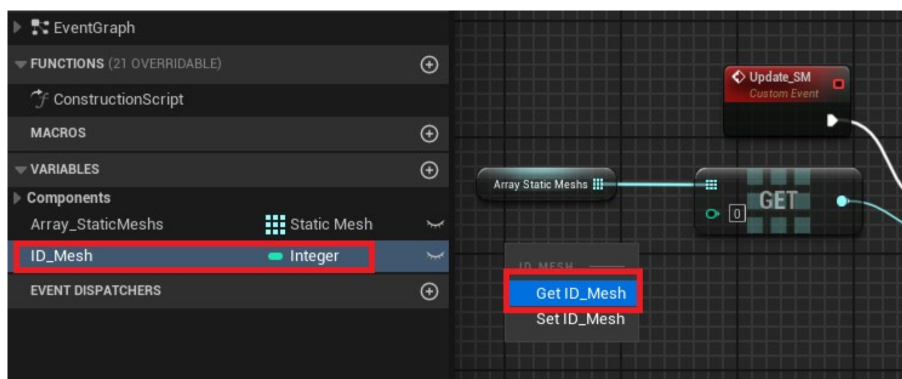

Рис. 2.60. Вызов переменной *ID_Mesh* для получения данных

В результате реализации данной логики при приближении пользователя объект должен корректно отображать виджет, информирующий о возможности смены модели с помощью клавиш клавиатуры (рис. 2.61). Перед проверкой работы выполните сохранение и компиляцию *Blueprint*-кода.

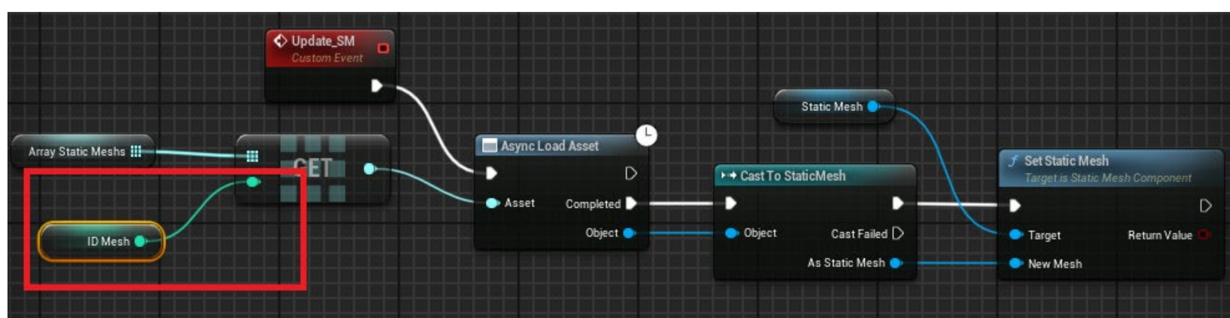

Рис. 2.61. Передача номера элемента массива из переменной

Задания

Результат работы необходимо сохранить в виде проекта *Unreal Engine*, совместимого с версией *UE 5.3* и выше. Название проекта должно соответствовать шаблону «*Familiya_Random*».

Следует учитывать, что в названии проекта запрещено использовать русские символы, пробелы и специальные знаки. Разрешены только латинские буквы, цифры и символ подчеркивания.

Задание 1. Реализация системы случайного размещения автомобилей и возможности их замены.

1. Создайте два *Blueprint*-класса на основе *Actor*:

– один для случайной генерации автомобилей (*RandomCar_BP*);

– второй для взаимодействия с ними (*ReplacingCar_BP*).

2. Разместите на уровне не менее 15 экземпляров объектов *RandomCar_BP* и *ReplacingCar_BP*, имитируя парковочные места.

Задание 2. Генерация деревьев с выбором из пользовательского набора.

1. Найдите или создайте самостоятельно не менее пяти трехмерных моделей деревьев в низкополигональном стиле.

2. Создайте *Blueprint*-класс *TreeSpawner_BP* на основе *Actor* для генерации деревьев случайного вида на уровне.

3. Разместите несколько объектов *TreeSpawner_BP* на уровне для формирования природного окружения.

Контрольные вопросы

1. Какой тип *Blueprint Class* необходимо выбрать при создании объекта *Random_BP* и почему?

2. Для чего используется компонент *Static Mesh* в *Blueprint*-объекте и каким образом он настраивается?

3. Какие изменения следует внести в параметры переменной, чтобы она могла хранить массив ссылок на трехмерные модели?

4. Какую роль выполняет узел *Random Integer In Range* при выборе случайной модели из массива?

5. Каково назначение компонента *Box Collision* и какие события он способен генерировать?

6. В чем заключается различие между типами данных *Soft Object Reference* и *Object Reference* при работе с массивом моделей в *Blueprint* и почему предпочтительно использовать именно *Soft Object Reference*?

Лабораторная работа № 3
СОЗДАНИЕ ВИРТУАЛЬНОГО ПЕРСОНАЖА С БАЗОВЫМ ИСКУССТВЕННЫМ ИНТЕЛЛЕКТОМ В СРЕДЕ UNREAL ENGINE

Время выполнения – 8 часов (аудиторная работа – 4 часа, самостоятельная работа – 4 часа).

Цель работы: освоить навыки разработки виртуального персонажа с элементами базового искусственного интеллекта для имитации поведения в *Unreal Engine*.

Задачи работы

1. Освоить процесс создания виртуального персонажа с базовым искусственным интеллектом в среде *Unreal Engine*.
2. Изучить принципы настройки *AIController* и взаимодействия с *Blueprint*-классами.
3. Реализовать передвижение персонажа по ключевым точкам с использованием навигационной сетки (*NavMesh*).

Перечень обеспечивающих средств

1. Персональный компьютер с доступом в интернет.
2. Среда моделирования *Blender* (версия не ниже 4.0).
3. Среда разработки *Unreal Engine* (версия не ниже 5.3).

Общие теоретические сведения

В интерактивных проектах, созданных в *Unreal Engine*, наряду с персонажем, управляемым пользователем, можно создавать дополнительных виртуальных персонажей. Для выполнения автономных действий используются персонажи с искусственным интеллектом (*AI Characters*). Их можно применять в качестве интерактивных объектов, движущихся целей, спутников или фоновых элементов, оживляющих виртуальную среду.

В рамках данной работы будет реализована базовая логика передвижения персонажа с искусственным интеллектом на примере робота-пылесоса. Его перемещение будет организовано между заранее заданными точками маршрута.

Подготовим трехмерную модель робота-пылесоса. Для этого можно использовать готовую модель, создать ее самостоятельно или получить у преподавателя. Добавьте модель в проект и настройте ее для корректного взаимодействия с окружающей средой (рис. 3.1).

Рис. 3.1. Добавление модели в проект

Откройте *Content Browser*, расположенный в нижней части интерфейса. Создайте новую папку для хранения элементов, связанных с искусственным интеллектом. Назовите папку *Blueprints_AI* (рис. 3.2).

Организуйте структуру проекта, размещая все связанные файлы, включая *Blueprint Classes*, точки маршрута и дополнительные элементы, в папке *Blueprints_AI*.

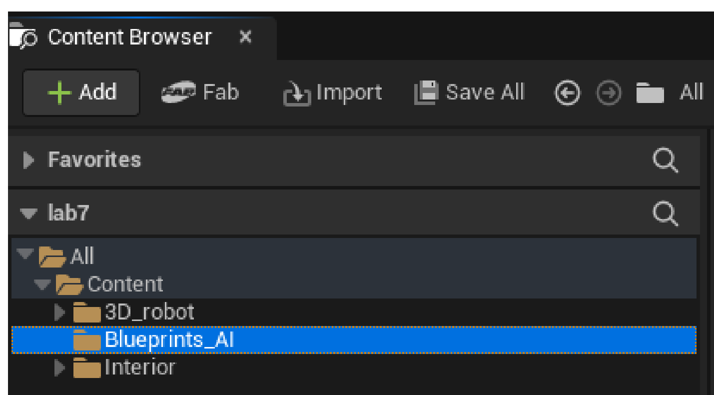

Рис. 3.2. Добавление папки *Blueprints_AI*

Перейдите в папку *Blueprints_AI* и щелкните правой кнопкой мыши по пустому пространству, чтобы открыть меню создания объектов (рис. 3.3). В открывшемся меню выберите *Blueprint Class* для создания нового класса.

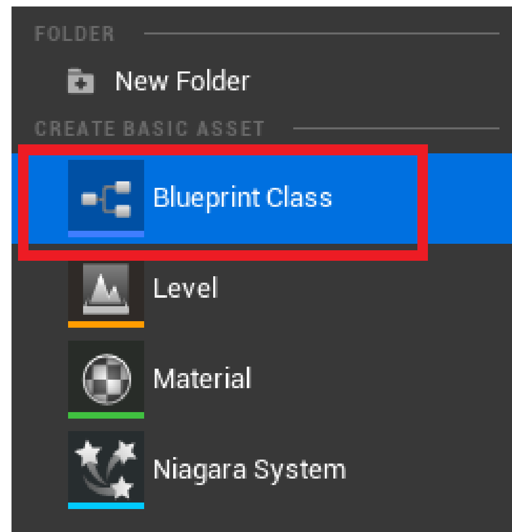

Рис. 3.3. Создание *Blueprint Class*

В окне выбора базового класса в разделе *All Classes* введите в строку поиска *AIController* и выберите одноименный класс (рис. 3.4). После подтверждения выбора присвойте созданному *Blueprint*-классу название, например, *AI_Controller_robot* (рис. 3.5).

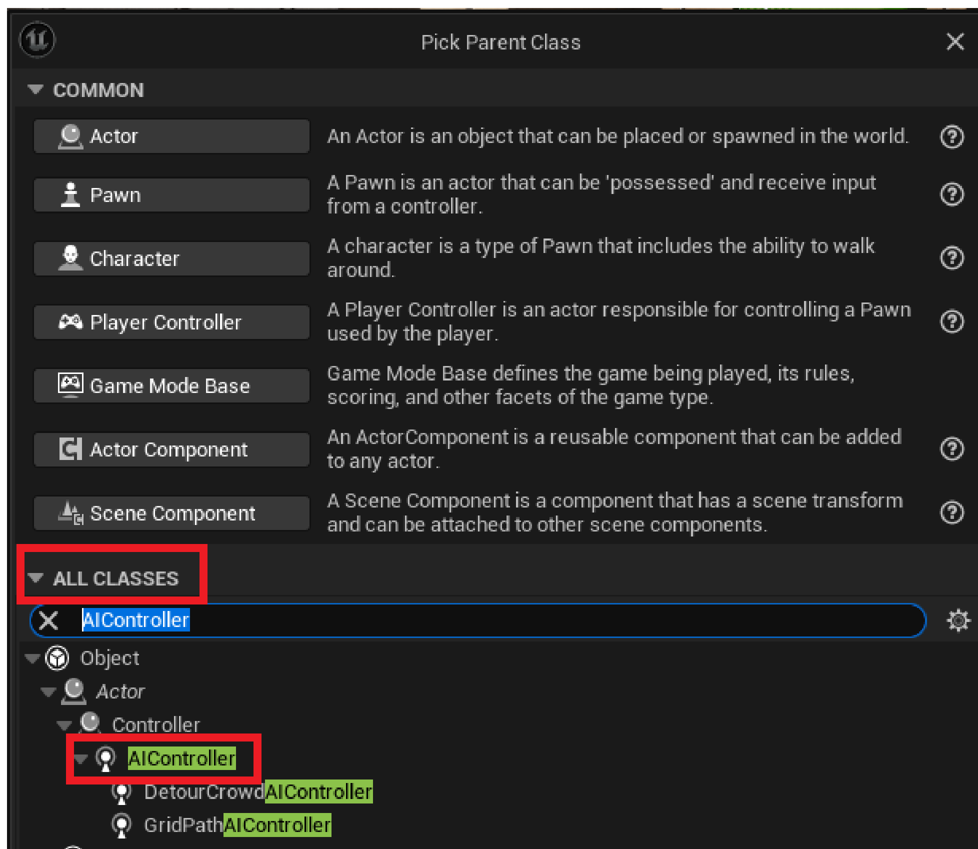

Рис. 3.4. Выбор типа Blueprint Class (*AIController*)

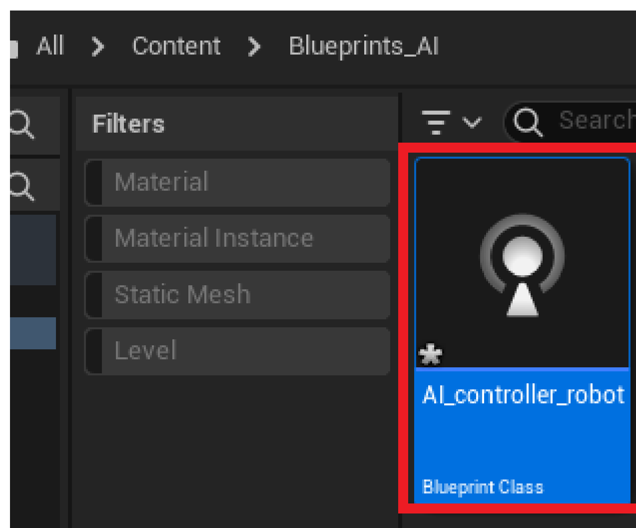

Рис. 3.5. Созданный объект *AI_Controller_robot*

Класс *AIController* в *Unreal Engine* представляет собой специализированный компонент, предназначенный для управления поведением персонажей с искусственным интеллектом.

AIController можно использовать как посредника между персонажем и окружающим миром. *AIController* применяется для создания автономных персонажей, таких как *NPC* или фоновых объектов, функционирующих без участия пользователя.

AIController необходимо назначать персонажу (*Pawn*), которым он управляет. Можно использовать как статические, так и динамические объекты с возможностью перемещения.

Команды движения и взаимодействия передаются через функции, такие как *Move To* и *Use Object*. Нужно использовать функцию *Move To Location* для перемещения персонажа в заданную точку, а *Move To Actor* – для следования за конкретным объектом.

Создайте персонажа для управления контроллером. Используйте собственный *Pawn*, однако учитывайте, что он требует ручной настройки параметров: размера, формы коллизии, скорости перемещения, гравитации и взаимодействия с поверхностями. Выполните такие настройки для получения корректного поведения, но учитывайте трудоемкость данного подхода.

Используйте готовый класс *Character*, расширяющий *Pawn* для ускорения разработки, так как он содержит преднастроенные механизмы управления движением и взаимодействием с окружением.

Используйте встроенный компонент *CharacterMovement* для обработки базовых действий персонажа: ходьбы, бега, прыжков и движения по наклонным поверхностям. Настройте параметры скорости перемещения, высоты прыжка и другие характеристики через параметры компонента.

Используйте капсульную коллизию, встроенную в *Character*, для определения формы взаимодействия персонажа с окружающим миром. Учитывайте, что форма коллизии ограничена цилиндрической капсулой. При необходимости выполните ее настройку, изменяя размеры в соответствии с формой вашего объекта.

Выполните настройку коллизии таким образом, чтобы она максимально соответствовала форме модели, например, в случае с роботом-пылесосом.

В папке *Blueprints_AI* щелкните правой кнопкой мыши по пустому пространству, чтобы открыть меню создания объектов. Выберите пункт *Blueprint Class* для создания нового класса.

В открывшемся окне выберите категорию *Character* (рис. 3.6).

Назовите созданный объект *AI_Robot* (рис. 3.7).

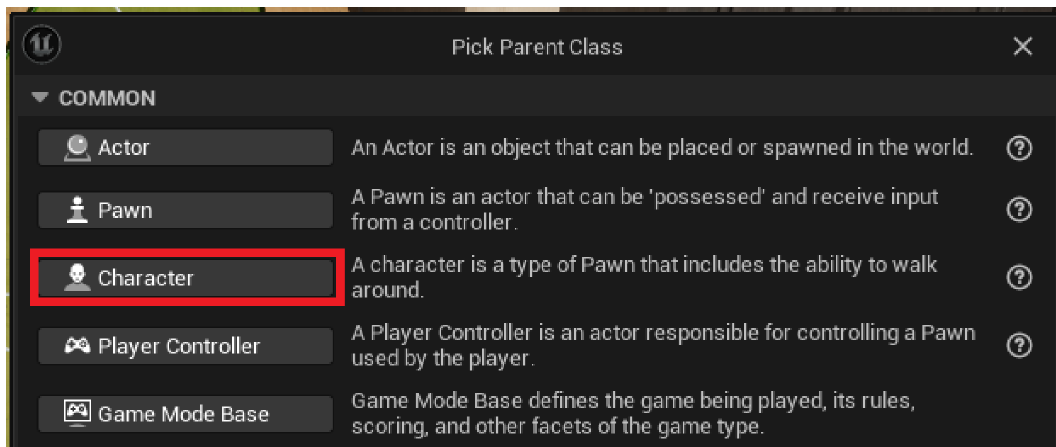

Рис. 3.6. Выбор типа Blueprint Class (*Character*)

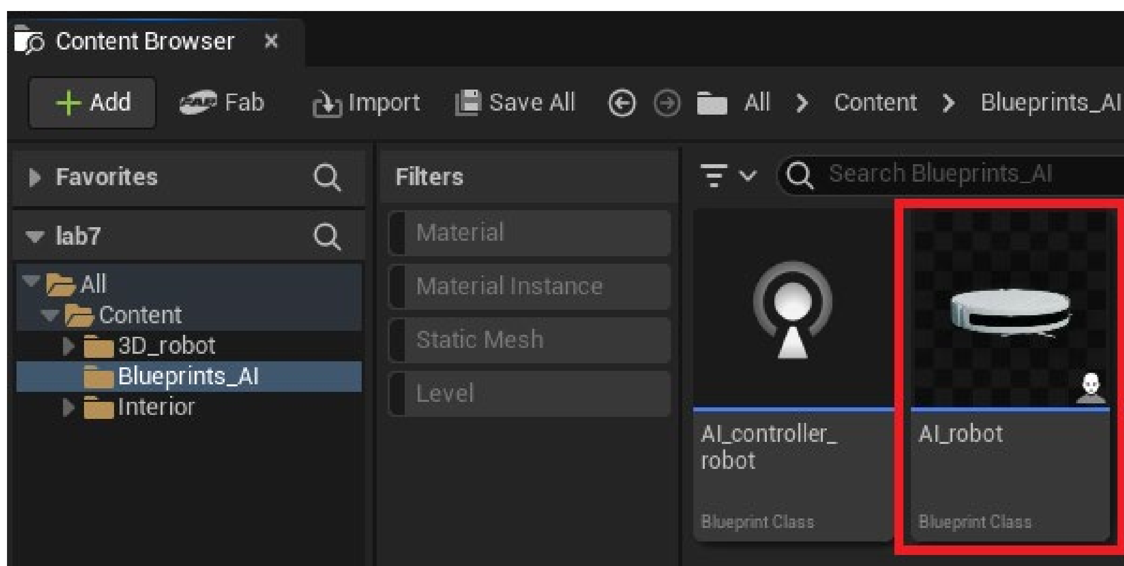

Рис. 3.7. Созданный объект *AI_robot*

В папке *Blueprints_AI* щелкните правой кнопкой мыши на пустом пространстве, чтобы открыть панель создания объектов. В появившемся меню

выберите пункт *Blueprint Class*, после чего откроется окно выбора базового класса. Выберите категорию *Actor*, предназначенную для создания базовых объектов, размещаемых на уровне (рис. 3.8).

Назначьте объекту имя *BP_TargetPoint*. Используйте данные объекты в качестве ключевых точек маршрута, к которым будет перемещаться робот-пылесос (рис. 3.9).

Сформируйте итоговую структуру проекта таким образом, чтобы в папке *Blueprints_AI* были представлены три объекта типа *Blueprint Class*. Используйте их в качестве основы для реализации поведения персонажа с базовым искусственным интеллектом.

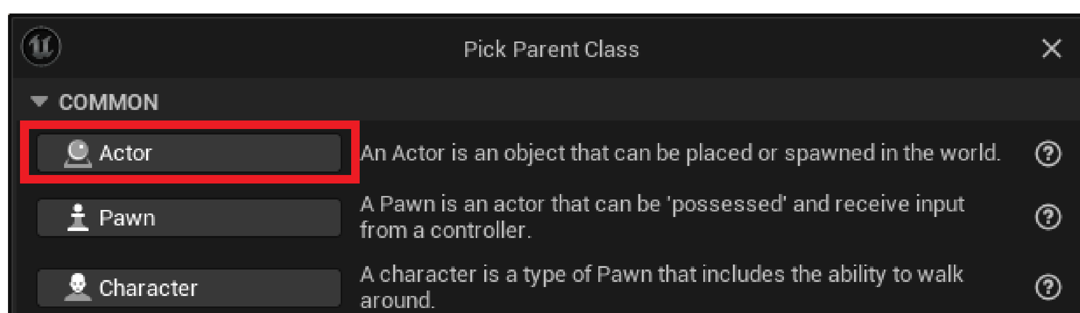

Рис. 3.8. Выбор типа Blueprint Class (*Actor*)

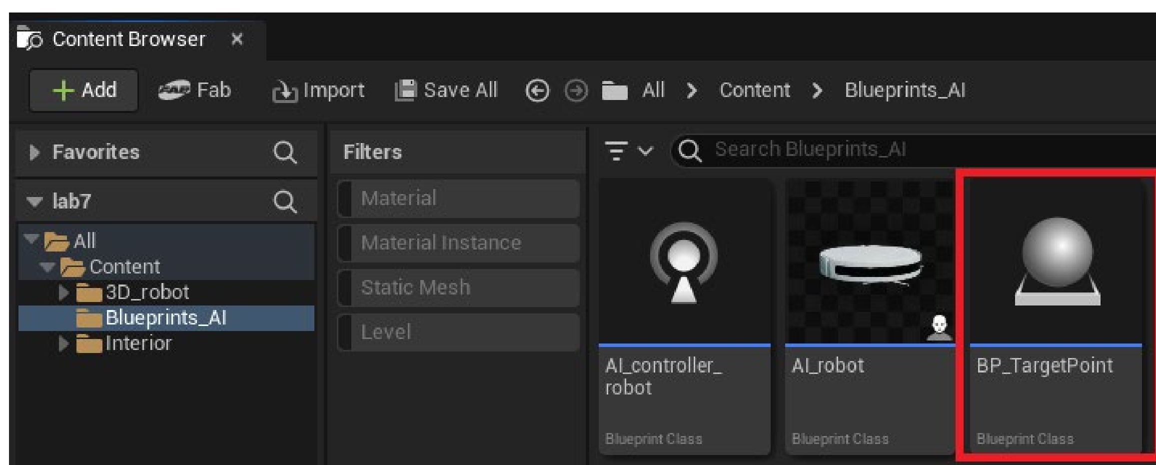

Рис. 3.9. Набор объектов *Blueprint Class* для создания персонажа с ИИ

Откройте ранее созданный *Blueprint* с именем *AI_Robot* (тип *Character*). В редакторе *Blueprint* найдите панель *Components*, расположенную слева.

Используйте ее для управления всеми составляющими персонажа, включая *Capsule Component*, *CharacterMovement* и другие базовые элементы.

Откройте *Content Browser* и перетащите трехмерную модель робота-пылесоса в окно редактора *Blueprint*. После добавления модель отобразится в списке компонентов и в рабочей области визуального редактора (рис. 3.10).

Разместите модель внутри капсулы таким образом, чтобы она находилась в ее центре и корректно соответствовала границам персонажа.

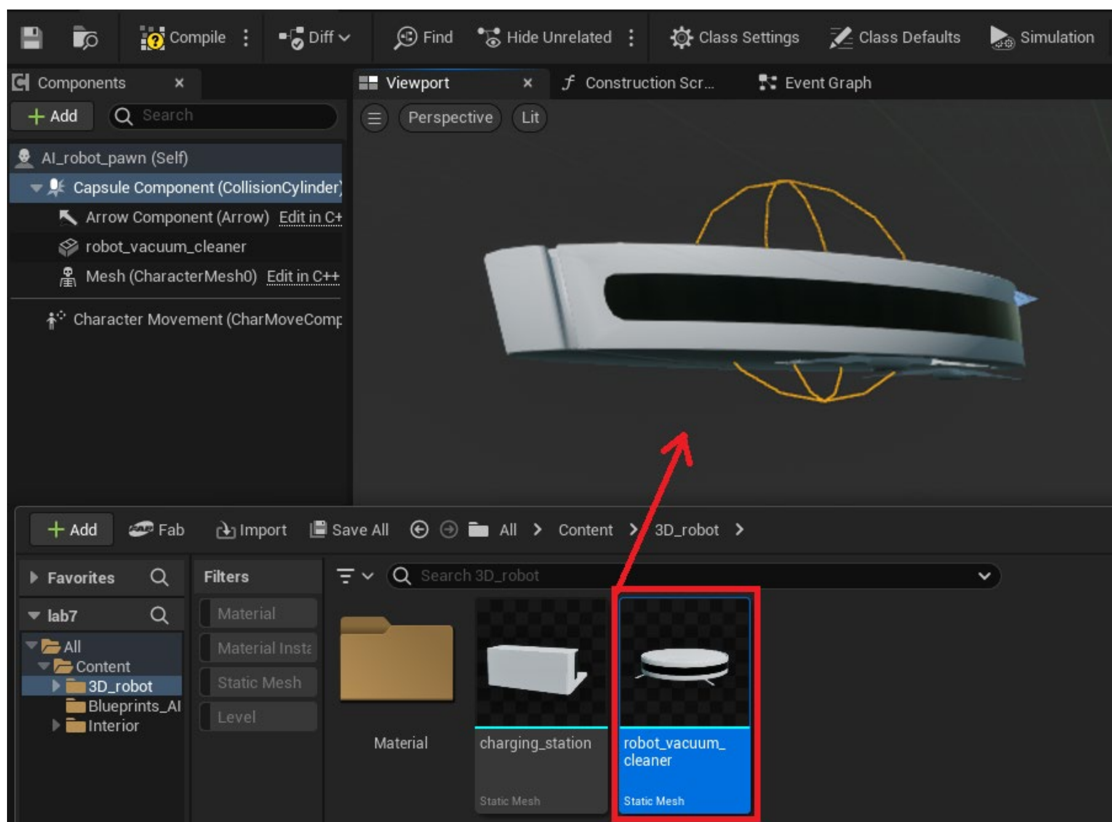

Рис. 3.10. Добавление модели в *Character*

Если размер трехмерной модели не соответствует масштабу виртуальной сцены, выполните настройку параметров.

Выделите компонент модели в списке компонентов, перейдите в панель *Details* и откройте раздел *Transform*. Измените параметр *Scale*, чтобы скорректировать масштаб модели. Уменьшите или увеличьте значения до достижения требуемого размера, обеспечивающего соответствие окружению (рис. 3.11).

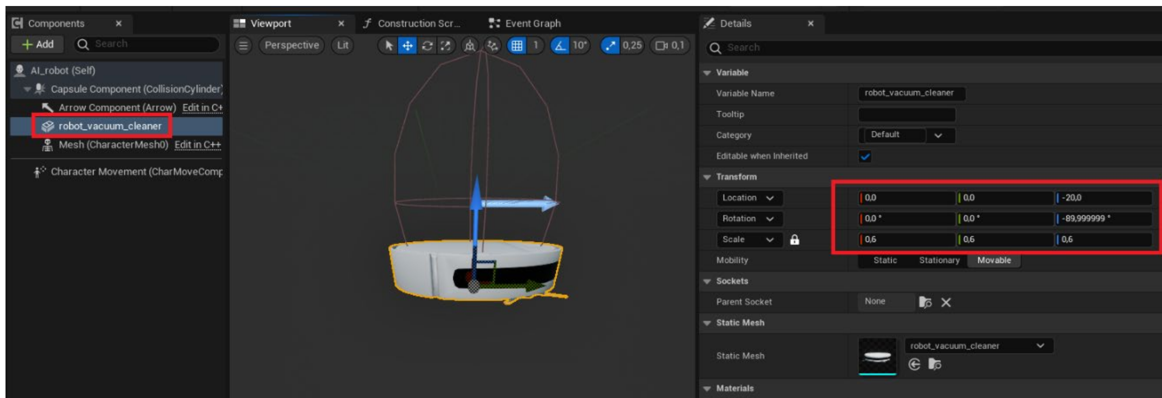

Рис. 3.11. Изменение масштаба модели в *Character*

Таким же образом выполните настройку размеров капсулы, чтобы они соответствовали габаритам модели робота-пылесоса. Учитывайте, что капсула имеет округлую форму и может не полностью совпадать с геометрией объекта. Адаптируйте ее параметры так, чтобы обеспечить корректное взаимодействие с окружающей средой.

Выделите компонент *Capsule Component* в панели компонентов редактора *Blueprint*. Перейдите в панель *Details* и найдите раздел *Shape*. Используйте параметры для изменения размеров капсулы:

– *Capsule Half Height* – задает высоту капсулы от центра до верхнего или нижнего края;

– *Capsule Radius* – определяет радиус капсулы в поперечном сечении.

Внесите необходимые изменения и убедитесь, что капсула корректно выровнена относительно модели (рис. 3.12).

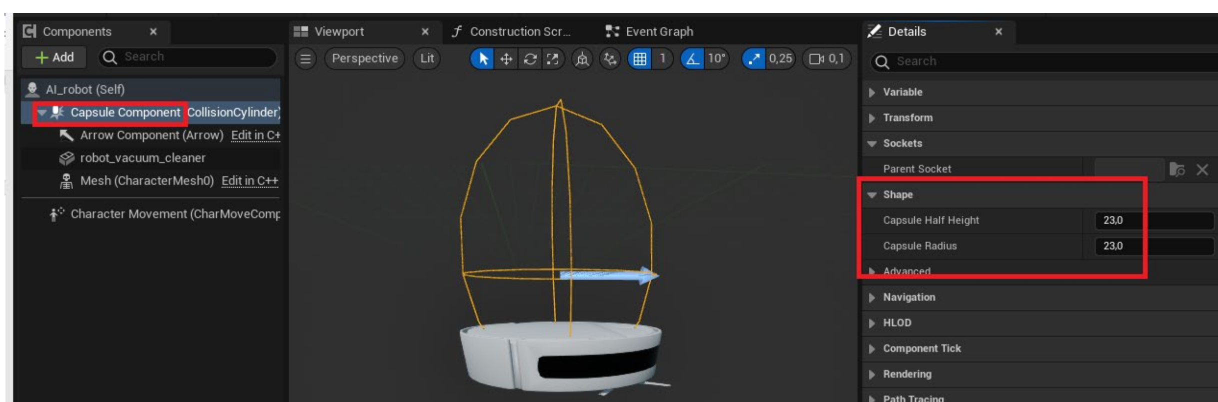

Рис. 3.12. Изменение размера капсулы *Character*

В панели *Components* слева выберите основной компонент вашего персонажа. Перейдите в панель *Details*, где отображаются параметры выбранного компонента.

Откройте раздел *Character Movement* (*Rotation Settings*). Найдите параметр *Orient Rotation to Movement*. Если параметр отключен, активируйте его, установив соответствующий флажок. Обеспечьте автоматический поворот персонажа в направлении движения (рис. 3.13).

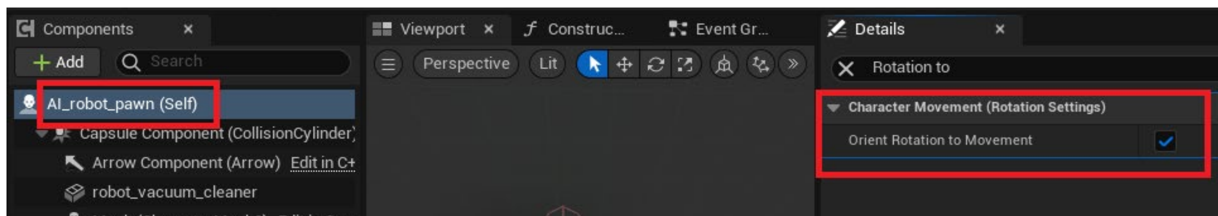

Рис. 3.13. Настройка поворота персонажа в направлении движения

Настройте связь персонажа с контроллером *AIController*. В основном компоненте персонажа перейдите в раздел *Pawn* и найдите параметр *AI Controller Class*, определяющий контроллер управления.

Откройте выпадающий список и выберите ранее созданный класс *AI_Controller_robot* (рис. 3.14).

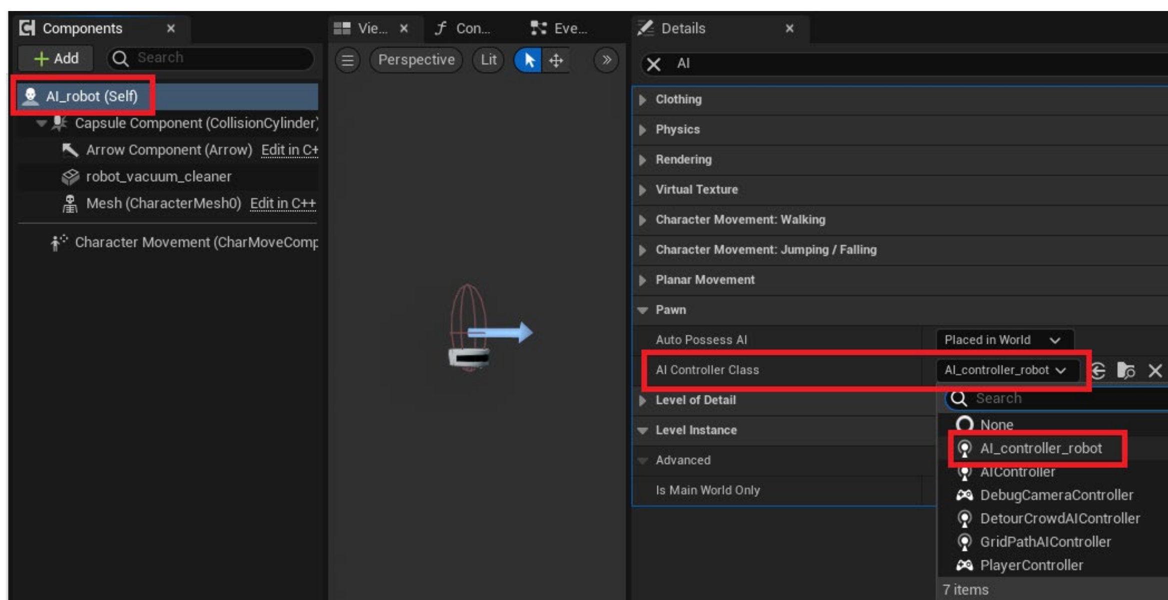

Рис. 3.14. Назначение контроллера управления персонажем

Поскольку стандартный компонент персонажа рассчитан на передвижение среднестатистического человека, выполните настройку параметров скорости для робота-пылесоса. Выберите компонент *CharacterMovement*, отвечающий за управление перемещением персонажа.

Перейдите в панель *Details* и откройте раздел *Character Movement* (*Walking*). Найдите параметр *Max Walk Speed*, определяющий максимальную скорость передвижения. Установите значение, соответствующее медленному движению, например, 50 единиц (рис. 3.15).

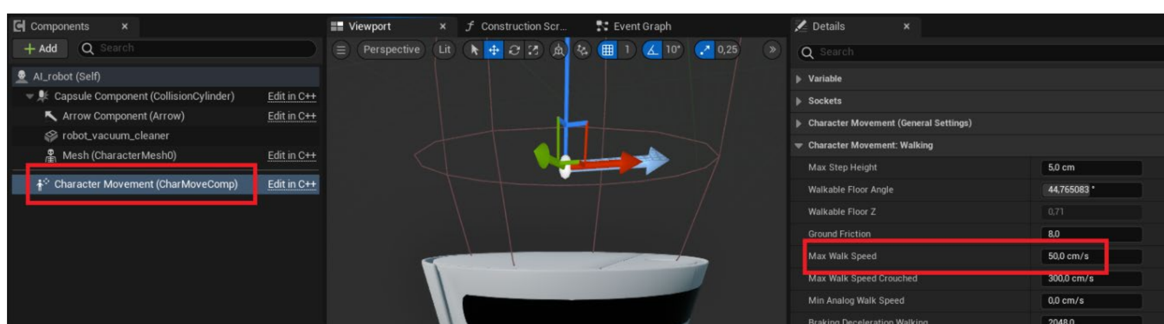

Рис. 3.15. Изменение скорости передвижения персонажа

Скомпилируйте и сохраните созданного персонажа, чтобы применить внесенные изменения (рис. 3.16).

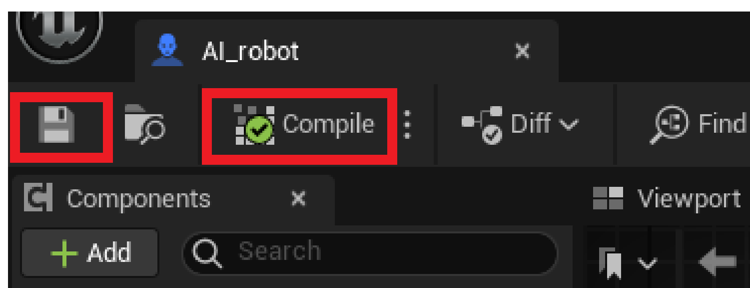

Рис. 3.16. Компиляция кода и сохранение персонажа

Откройте *AI_Controller_robot* – ранее созданный *AIController*, отвечающий за управление поведением робота-пылесоса (рис. 3.17). Реализуйте в нем основную логику: поиск целевых точек и выполнение перемещения к ним.

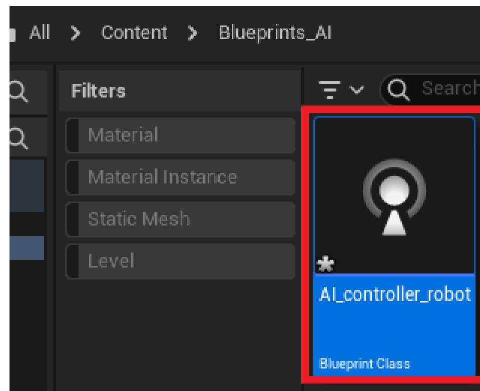

Рис. 3.17. Открытие AI_Controller_robot

Перейдите во вкладку *Event Graph* в редакторе *Blueprint* контроллера. Задайте здесь визуальную логику поведения персонажа. Реализуйте этапы выбора ближайшей точки (*Target Point*) и выполнения команды движения.

Создайте все точки маршрута на основе класса *Actor BP_TargetPoint*. Обеспечьте их доступность для контроллера, загрузив их в память.

Используйте функцию *Get All Actors of Class* для получения списка всех объектов заданного типа на уровне. В параметре *Actor Class* выберите ранее созданный класс *BP_TargetPoint* (рис. 3.18).

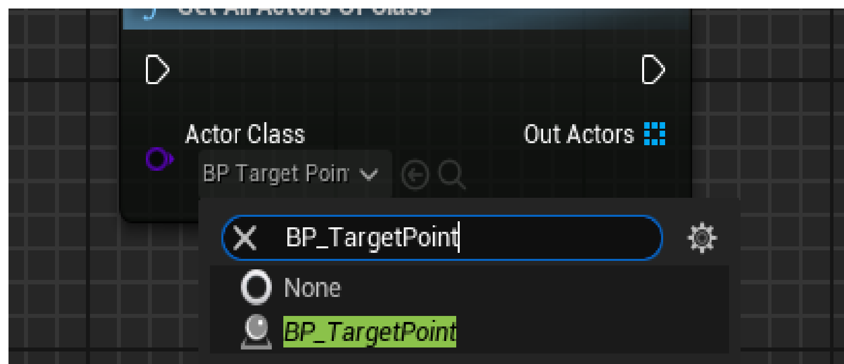

Рис. 3.18. Функция Get All Actors of Class

Функцию *Get All Actors of Class* выполняйте один раз для загрузки всех точек маршрута в память при запуске уровня. Используйте для этого событие *Event BeginPlay*, которое автоматически срабатывает при старте работы контроллера (рис. 3.19).

Рис. 3.19. Связь функции Get All Actors of Class и события Begin Play

Сохраните данные, возвращаемые функцией *Get All Actors of Class*, в переменную. Используйте выходной порт *Out Actors*. Щелкните правой кнопкой мыши по иконке массива и выберите *Promote to Variable*. Создайте переменную для хранения массива полученных объектов (рис. 3.20).

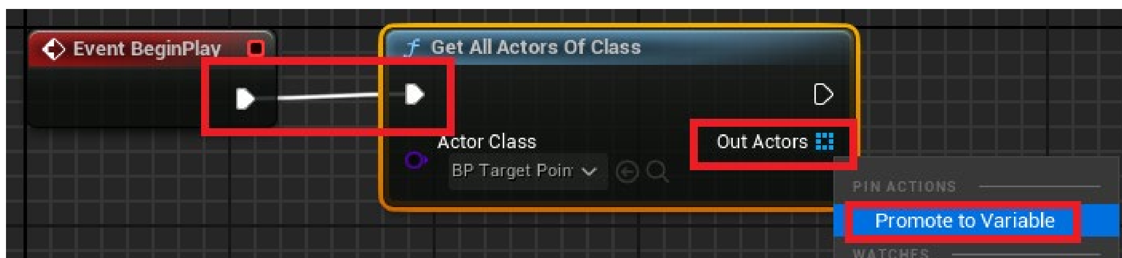

Рис. 3.20. Сохранение данных из *Get All Actors of Class* в переменную

После создания переменной она автоматически подключается к выходу *Out Actors* и данные массива функции сохраняются в этой переменной (рис. 3.21).

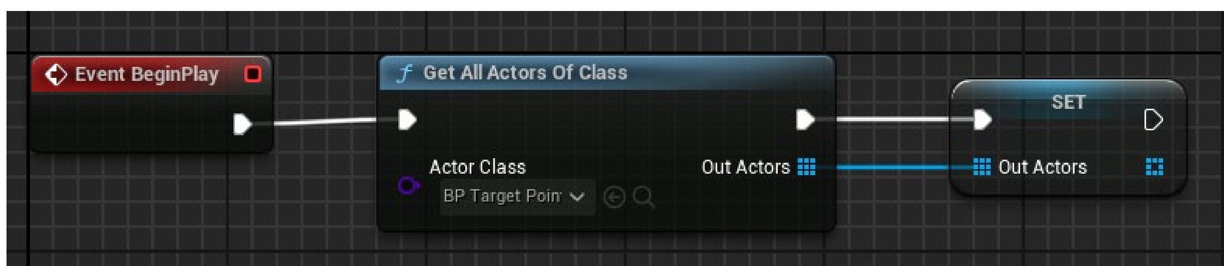

Рис. 3.21. Загрузка данных о точках в переменную

Измените название переменной с *Out Actors* на *TargetPoints*. Для этого выделите переменную и перейдите в панель *Details*. В верхней части панели найдите поле имени переменной и задайте новое название *TargetPoints* (рис. 3.22).

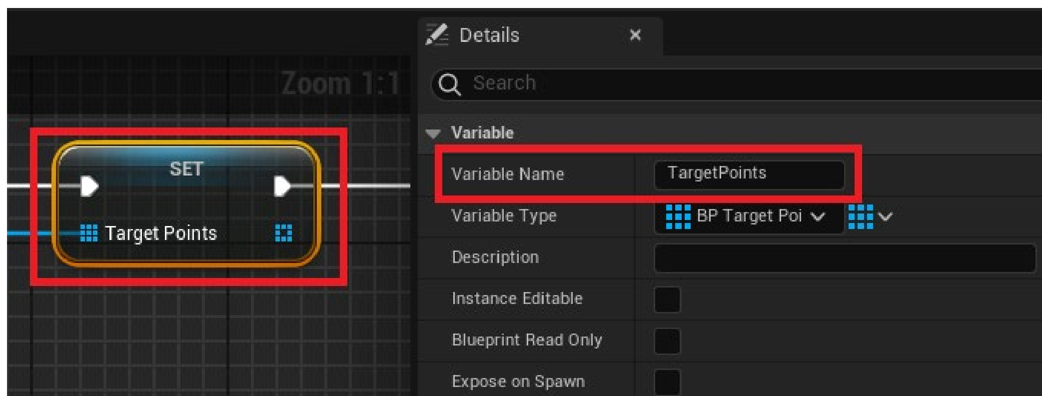

Рис. 3.22. Изменение названия переменной

Движение персонажа запускайте по отдельному событию. В *Event Graph* щелкните правой кнопкой мыши по пустому пространству, чтобы открыть меню добавления элементов. В появившемся списке выберите пункт *Add Custom Event* (рис. 3.23).

Переименуйте созданное событие в *MoveToRandomPoint*, используя панель *Details* или через контекстное меню.

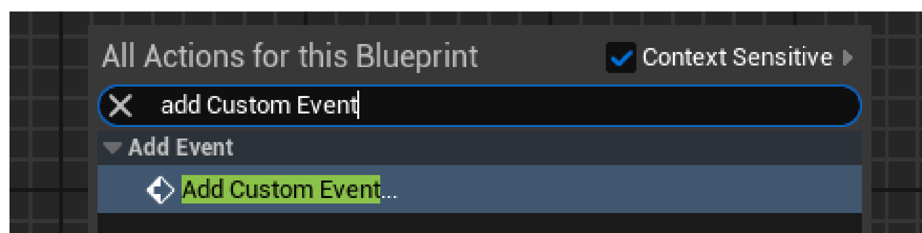

Рис. 3.23. Добавление *Custom Event*

Привяжите к созданному событию дальнейшую логику, включая выбор случайной точки из массива. Для этого добавьте функцию *Random Integer in*

Range, которая позволит выбрать случайный индекс в заданном диапазоне (рис. 3.24).

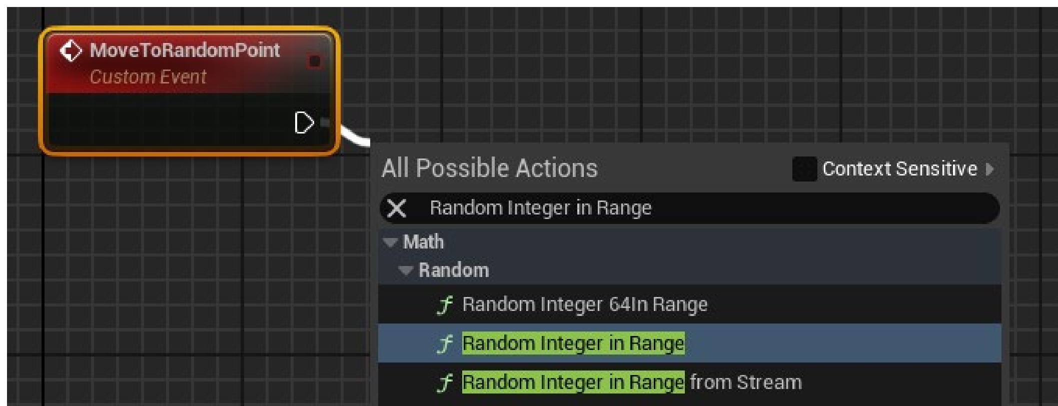

Рис. 3.24. Добавление Random Integer in Range

Для получения доступа к элементам массива переместите переменную *TargetPoints* в рабочую область графа. Перетащите переменную из списка слева и в появившемся меню выберите вариант *Get*. Это создаст узел для получения доступа к данным массива (рис. 3.25).

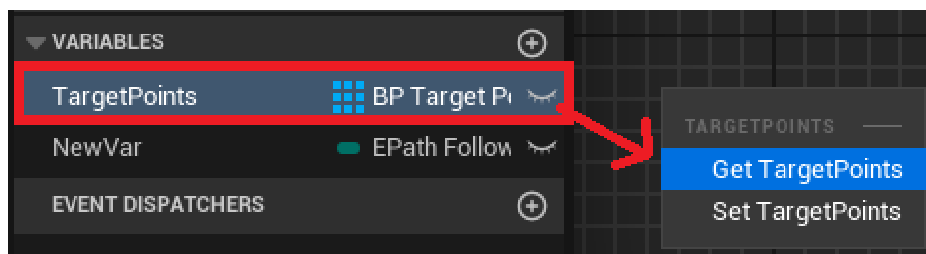

Рис. 3.25. Вызов переменной *TargetPoints*

Определите последний индекс массива, используя функцию *Last Index* (рис. 3.26).

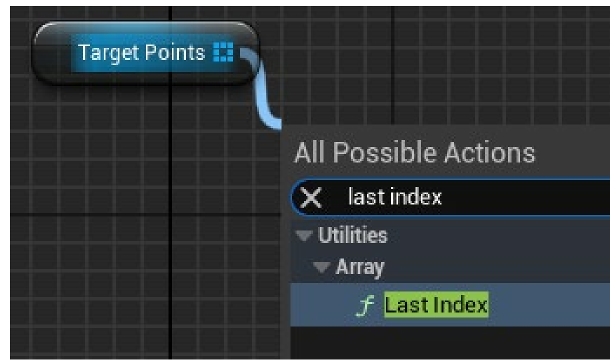

Рис. 3.26. Вызов функции получения последнего индекса массива

Получите элемент массива по случайному индексу. От выходного порта переменной *TargetPoints* протяните линию и вызовите функцию *Get a Copy*, которая позволяет получить копию элемента массива, не изменяя исходные данные (рис. 3.27).

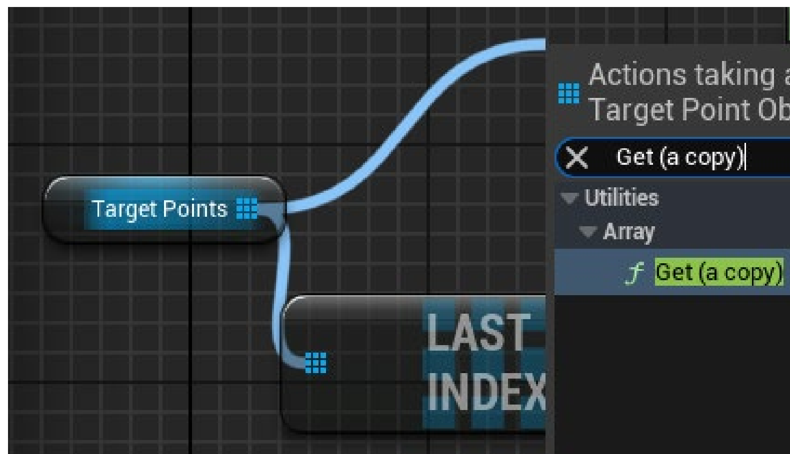

Рис. 3.27. Вызов функции получения копии элемента массива с указанным индексом

Подключите выход *Last Index* к входу *Max* функции *Random Integer in Range*. Таким образом, случайный индекс будет генерироваться в диапазоне от 0 до последнего индекса массива.

Подключите выход функции *Random Integer in Range* к входу *Index* функции *Get a Copy*. Это обеспечит получение случайного элемента массива из доступных значений (рис. 3.28).

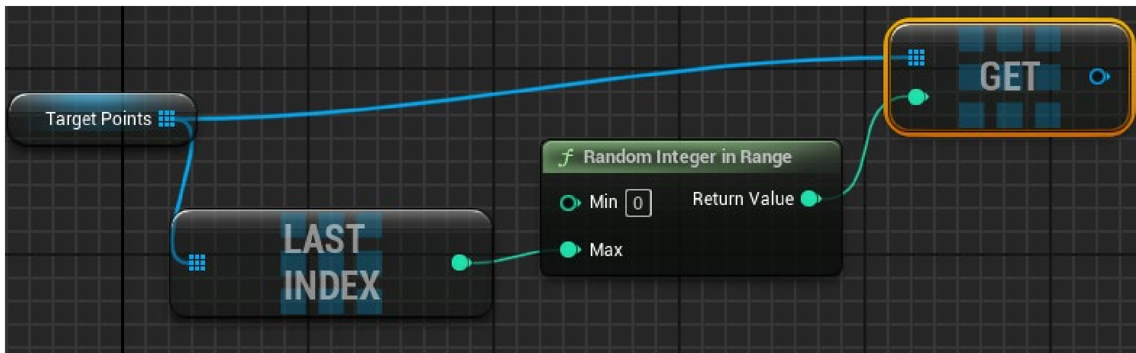

Рис. 3.28. Логика вызова случайного элемента массива

Сформировав случайную точку из массива, реализуйте перемещение персонажа. Найдите функцию *AI Move To* (рис. 3.29).

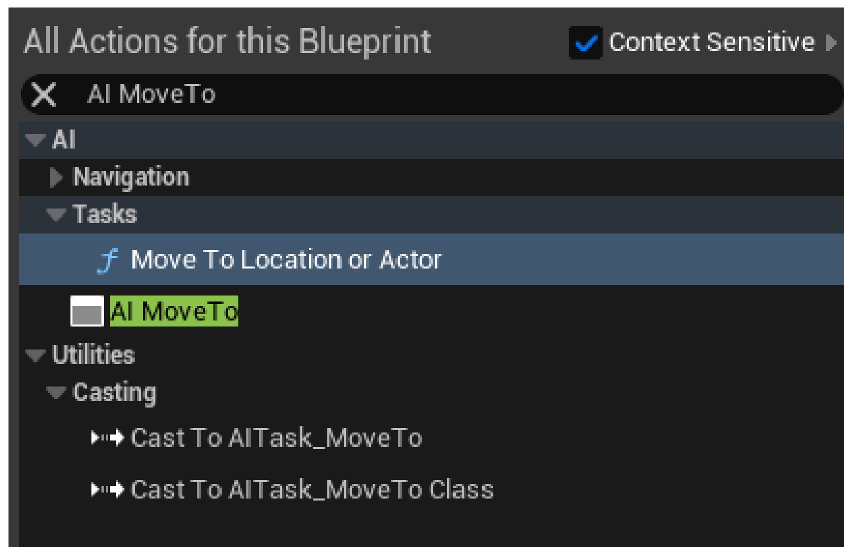

Рис. 3.29. Вызов функции *AI Move To*

Используйте функцию *AI Move To* для задания цели перемещения персонажа, управляемого *AIController*. Указывайте цель в виде объекта (*Actor*). Функция автоматически рассчитывает маршрут с использованием системы навигации *NavMesh*, обеспечивая корректное перемещение с учетом препятствий. Также обеспечьте обработку событий завершения движения, включая успешное достижение цели или прерывание пути.

Соедините событие *MoveToRandomPoint* с функцией *AI Move To*. Передайте данные из копии элемента массива во вход *Target Actor* функции *AI Move To* (рис. 3.30).

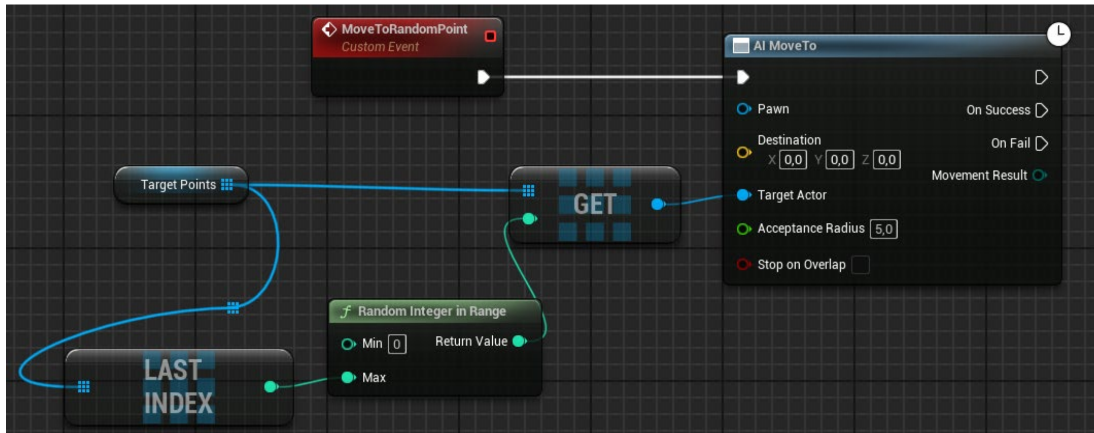

Рис. 3.30. Подключение функции *AI Move To*

Добавьте три функции, которые будут повторно запускать событие *MoveToRandomPoint* после завершения движения (рис. 3.31).

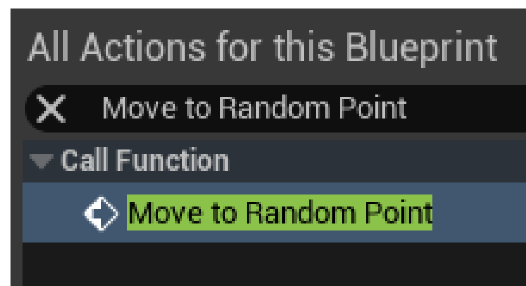

Рис. 3.31. Повторный запуск события *MoveToRandomPoint*

Добавьте функцию *Delay* для задания задержки между последовательными перемещениями персонажа (рис. 3.32).

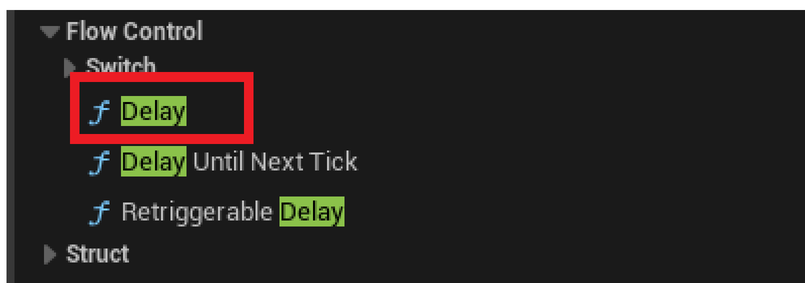

Рис. 3.32. Добавление функции *Delay*

Для отображения на экране информации о текущих действиях используйте функцию *Print String* (рис. 3.33).

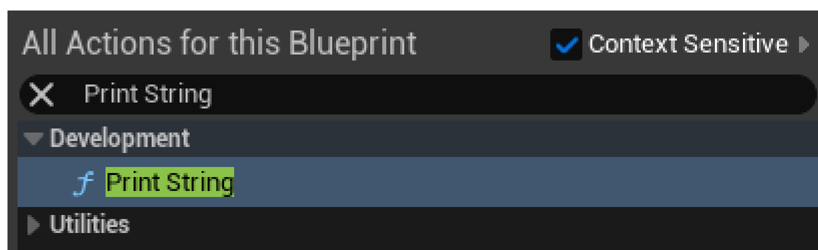

Рис. 3.33. Вызов функции *Print String*

Оставьте один вызов события *MoveToRandomPoint* в резерве – он будет использоваться в дальнейшем.

Сформируйте из оставшихся функций две последовательности (рис. 3.34).

Первая цепочка: выведите строку «Цель достигнута. Меняю направление», затем выполните задержку в 1 секунду с помощью *Delay* и повторно вызовите событие *MoveToRandomPoint*.

Вторая цепочка: выведите строку «Двигаюсь к точке» и сразу запустите событие *MoveToRandomPoint*.

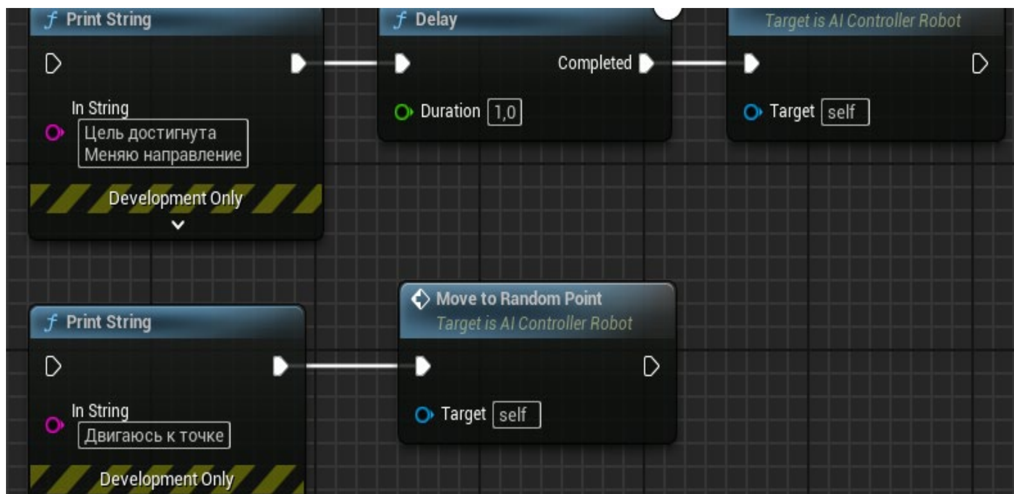

Рис. 3.34. Две цепочки действий перед повторным запуском события

Первая цепочка должна запускаться при достижении персонажем цели. Используйте выход *On Success* функции *AI Move To* для ее активации.

Вторая цепочка должна запускаться при невозможности достижения цели. Подключите ее к выходу *On Fail* функции *AI Move To*, чтобы повторно инициировать движение при возникновении ошибки.

Соедините функцию *AI Move To* с двумя созданными цепочками действий (рис. 3.35).

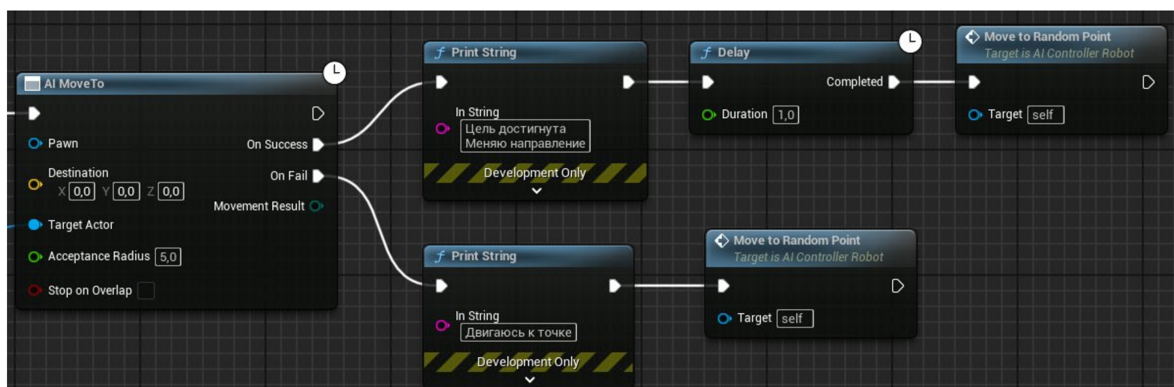

Рис. 3.35. Соединение цепочек вызова события *MoveToRandomPoint* и функции *AI Move To*

Добавьте последний вызов события *MoveToRandomPoint* к логике, связанной с событием *Event Begin Play* (рис. 3.36).

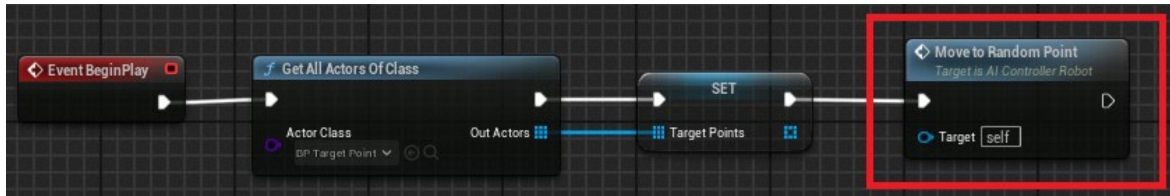

Рис. 3.36. Дополнение логики, связанной с *Event Begin Play*

Сформируйте итоговую логику перемещения персонажа к случайной точке (рис. 3.37).

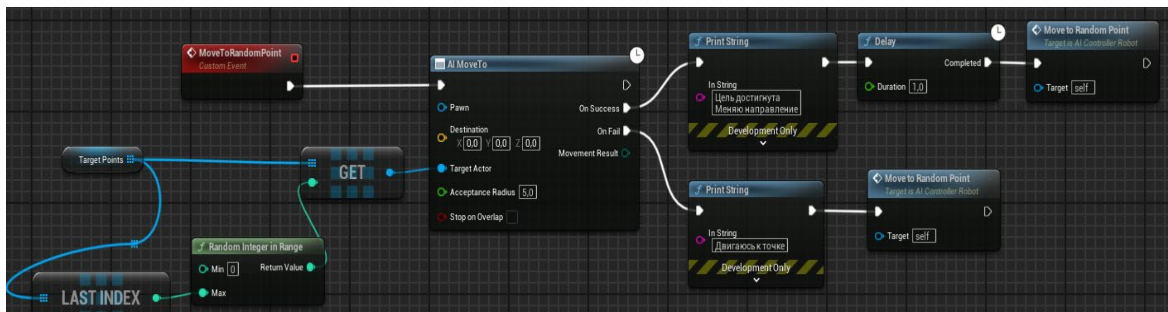

Рис. 3.37. Итоговый результат создания логики передвижения персонажа к случайной точке

Скомпилируйте и сохраните *AIController* (рис. 3.38).

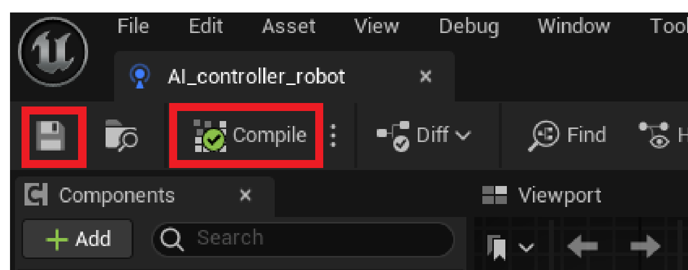

Рис. 3.38. Компиляция кода и сохранение контроллера

Переместите персонажа *AI_Robot* из *Content Browser* на уровень. Разместите не менее двух целевых точек *BP_TargetPoint* (рис. 3.39).

Для обеспечения поиска маршрутов между точками добавьте навигационную сетку *Nav Mesh*. Используйте ее для расчета всех возможных путей, доступных для перемещения *AI*-персонажей. Найдите *Nav Mesh* в панели *Place Actor Panel* (рис. 3.40).

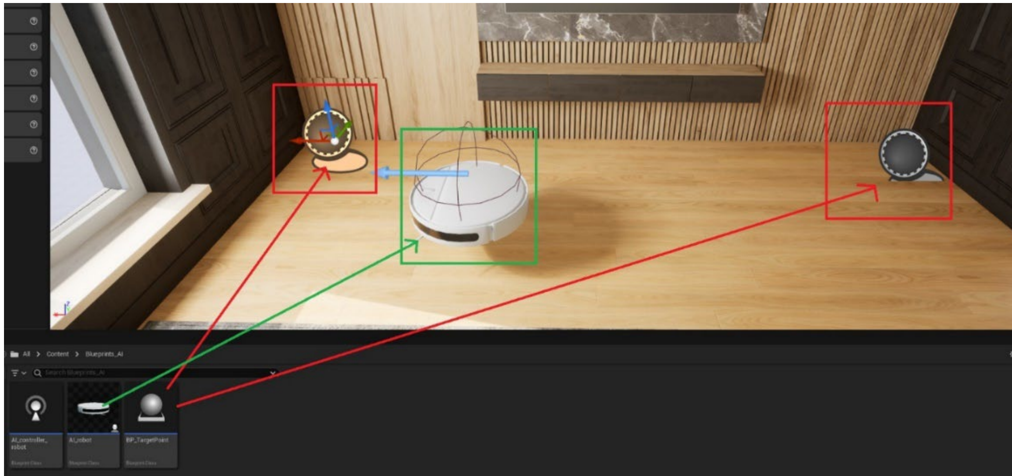

Рис. 3.39. Добавление целевых точек и персонажа на уровень

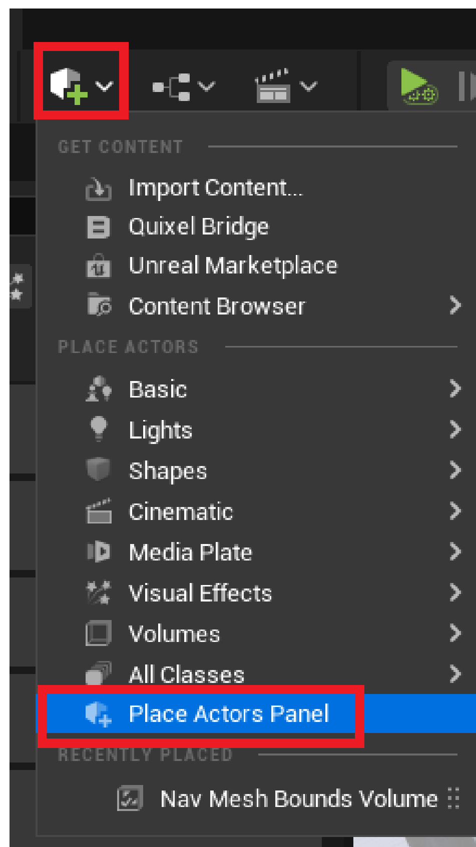

Рис. 3.40. Вызов панели *Place Actor Panel*

Nav Mesh Bounds Volume задает область, в пределах которой вычисляется путь для персонажей, управляемых искусственным интеллектом (*AI*). Используйте эту область как основу для генерации всех возможных маршрутов движения и расчета доступных зон перемещения с учетом препятствий.

Настройте *Nav Mesh Bounds Volume* таким образом, чтобы он охватывал все участки уровня, по которым должен перемещаться персонаж. При необходимости измените его размеры, чтобы покрыть всю игровую территорию или только нужную часть сцены, например зону перемещения робота-пылесоса.

Добавьте *Nav Mesh Bounds Volume* на уровень и нажмите клавишу *P* на клавиатуре для отображения сетки навигации (рис. 3.41).

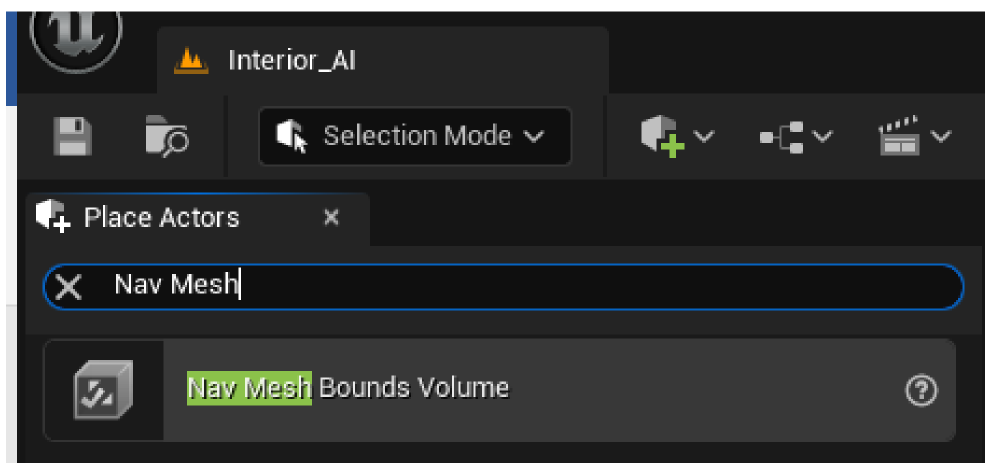

Рис. 3.41. Добавление на уровень *Nav Mesh Bounds Volume*

Территория покрытия *Nav Mesh* в редакторе отображается зеленым цветом (рис. 3.42).

Размещайте все точки внутри зеленой зоны. Используйте эту область для обеспечения доступности маршрутов для *AI*-персонажа. Если точка находится вне зоны покрытия, маршрут к ней не будет построен и персонаж должен перейти к следующей точке из массива.

Старайтесь размещать целевые точки исключительно в пределах подсвеченной области.

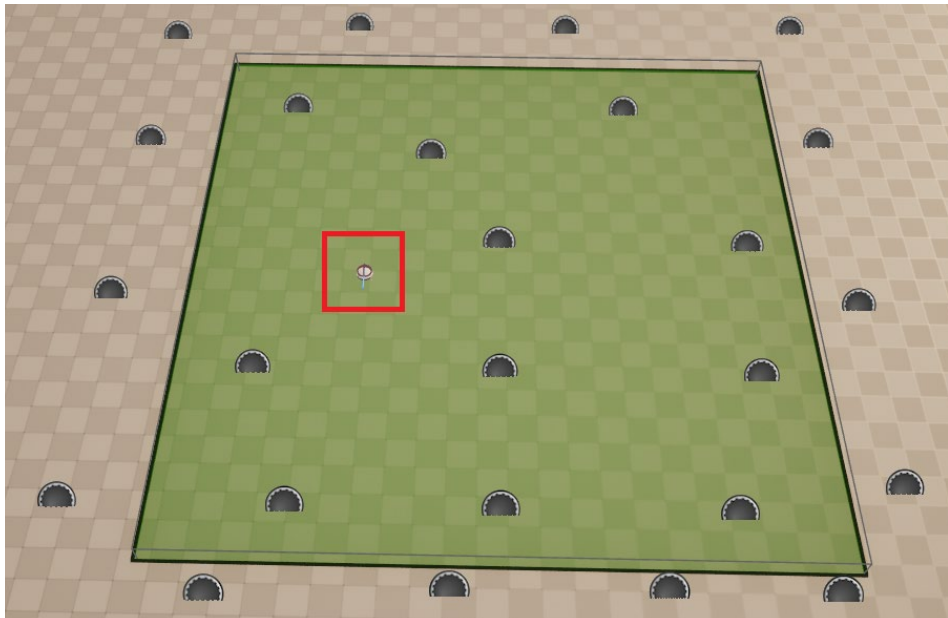

Рис. 3.42. Отображение *Nav Mesh*

Возможность прохождения маршрутов зависит от настроек *NavMesh*, которые управляются объектом *RecastNavMesh*. Используйте этот объект для хранения данных навигационной сетки и расчета путей для персонажей (рис. 3.43).

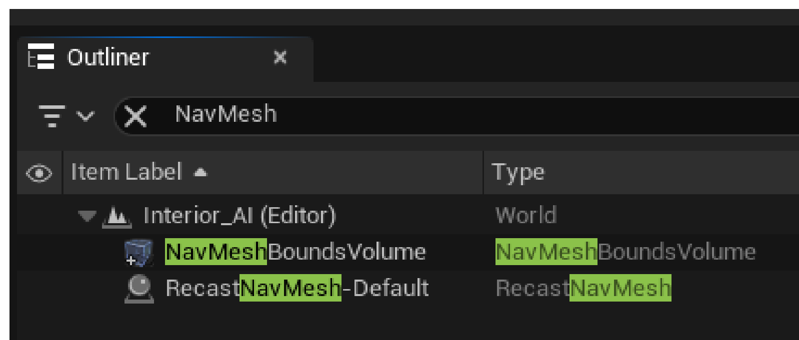

Рис. 3.43. Параметры настроек *RecastNavMesh*

Для корректного прохождения персонажем узких участков настройте параметры *Agent Radius* и *Agent Height* в *NavMesh* таким образом, чтобы они соответствовали размерам капсулы коллизии персонажа.

Синхронизируйте значения *Agent Radius* и *Agent Height* с параметрами капсулы, чтобы обеспечить корректный расчет проходимости маршрутов.

Учитывайте, что при несоответствии этих параметров возможны ошибки построения пути. Если размеры персонажа превышают допустимые значения для участка (например, дверного проема), *NavMesh* будет считать маршрут непроходимым и не позволит персонажу двигаться через такие области (рис. 3.44).

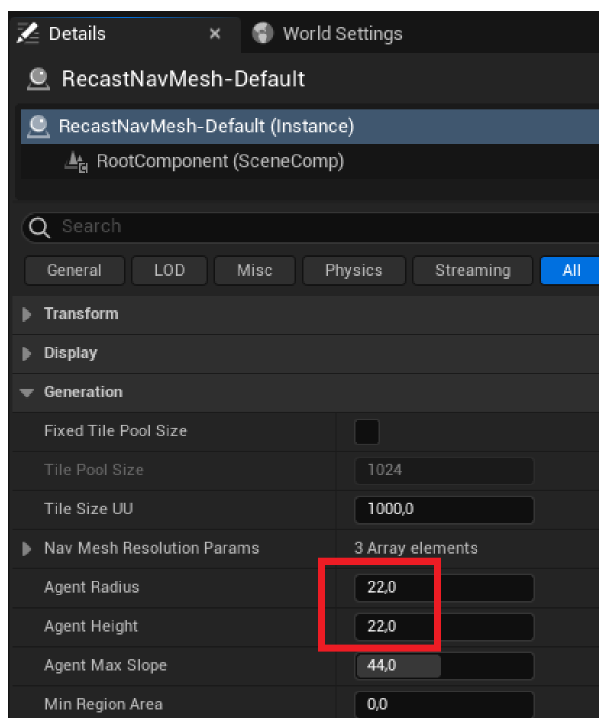

Рис. 3.44. Изменение параметров радиуса и высоты капсулы персонажа для расчета пути

После выполнения всех настроек запустите проект и проверьте корректность работы логики перемещения персонажа.

Задания

Сохраните результат работы в виде проекта *Unreal Engine*, совместимого с версией *UE 5.3* и выше. Назовите проект по шаблону *Familiya_AICharacter*.

Соблюдайте требования к названию проекта: используйте только латинские буквы, цифры и символ подчеркивания. Не применяйте русские символы, пробелы и специальные знаки.

Задание 1. Реализация базового поведения *AI*-персонажа с перемещением по случайным точкам.

1. Создайте три класса: *AI_Robot* (*Character*), *AI_Controller_robot* (*AIController*) и *BP_TargetPoint* (*Actor*). Разместите их в папке *Blueprints_AI*.

2. Разместите на уровне не менее трех экземпляров *BP_TargetPoint* в разных точках виртуального пространства.

3. Настройте *AI_Controller_robot* таким образом, чтобы *AI_Robot* перемещался к случайной точке из массива, используя функцию *MoveTo*.

4. Разместите на уровне *AI_Robot* и *Nav Mesh Bounds Volume*. Проверьте корректность работы логики перемещения при запуске уровня.

Задание 2. Реализация последовательного движения *AI*-персонажа по точкам маршрута.

1. Создайте и модифицируйте копию класса *AI_Controller_robot*, чтобы *AI_Robot* перемещался по точкам *BP_TargetPoint* в порядке их следования (от первой до последней в массиве).

2. Обеспечьте возврат к начальной точке или завершение маршрута после достижения последней точки (выберите вариант по своему усмотрению).

3. Проверьте корректность перемещения *AI_Robot* между точками при запуске проекта.

Контрольные вопросы

1. Что представляет собой класс *AIController* в *Unreal Engine* и какова его основная функция?

2. Для чего используется компонент *Nav Mesh Bounds Volume* и как он влияет на передвижение *AI*-персонажа?

3. Почему важно согласовывать размеры капсулы персонажа с параметрами *Agent Radius* и *Agent Height* в *NavMesh*?

4. Что произойдет, если целевая точка окажется вне зоны покрытия *NavMesh*, и как система будет реагировать в этом случае?

5. В чем заключается различие между классами *Pawn* и *Character*, и почему в данной лабораторной работе используется именно *Character*?

Лабораторная работа № 4
СОЗДАНИЕ БАЗОВОГО ПРОЕКТА ДЛЯ ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ (VR), ЕГО ЗАПАКОВКА И ТЕСТИРОВАНИЕ

Время выполнения – 8 часов (аудиторная работа – 4 часа, самостоятельная работа – 4 часа).

Цель работы: овладеть навыками создания базового VR-приложения в среде разработки *Unreal Engine*, а также освоить процесс его упаковки и тестирования на совместимость с устройствами виртуальной реальности.

Задачи работы

1. Ознакомиться с устройством и принципами работы шлема виртуальной реальности *Pico 4* и его контроллеров.
2. Изучить процесс подключения VR-шлема к персональному компьютеру с использованием утилиты *PICO Connect* и платформы *Steam VR*.
3. Настроить запуск проекта в режиме предпросмотра *VR*, выполнить тестирование сцены в шлеме и оценить корректность отображения и взаимодействия.

Перечень обеспечивающих средств

1. Персональный компьютер с доступом к сети Интернет.
2. Система виртуальной реальности *Pico 4*.
3. Среда разработки *Unreal Engine* (версия не ниже 5.3).
4. Программное обеспечение *Steam VR* для обеспечения взаимодействия между персональным компьютером и устройствами виртуальной реальности.
5. Программа *PICO Connect* для подключения шлема *Pico 4* к персональному компьютеру.

Общие теоретические сведения

В работе обучающимся предстоит использовать систему виртуальной реальности (*VR*) *Pico* 4. Это автономное устройство, предназначенное для

отображения и взаимодействия с виртуальными мирами. В комплект входит шлем виртуальной реальности и 2 контроллера (рис. 4.1).

Рис. 4.1. Система виртуальной реальности *Pico 4*

Шлем имеет разрешение дисплея 2 160 × 2 160 пикселей на каждый глаз и угол обзора до 105° с частотой обновления 72 или 90 Гц. В систему встроен 8-ядерный процессор *Qualcomm Snapdragon XR2* с 8 ГБ оперативной и 128 ГБ постоянной памяти.

Возможна как автономная работа системы на собственной ОС, так и подключение к ПК по кабелю или по *Wi-Fi*. Для автономной работы шлем имеет встроенный аккумулятор на 5 300 *mAh*.

Контроллеры шлема *Pico 4* симметричны по конструкции и имеют схожие элементы управления, выполняющие разные функции, в зависимости от контекста использования (рис. 4.2).

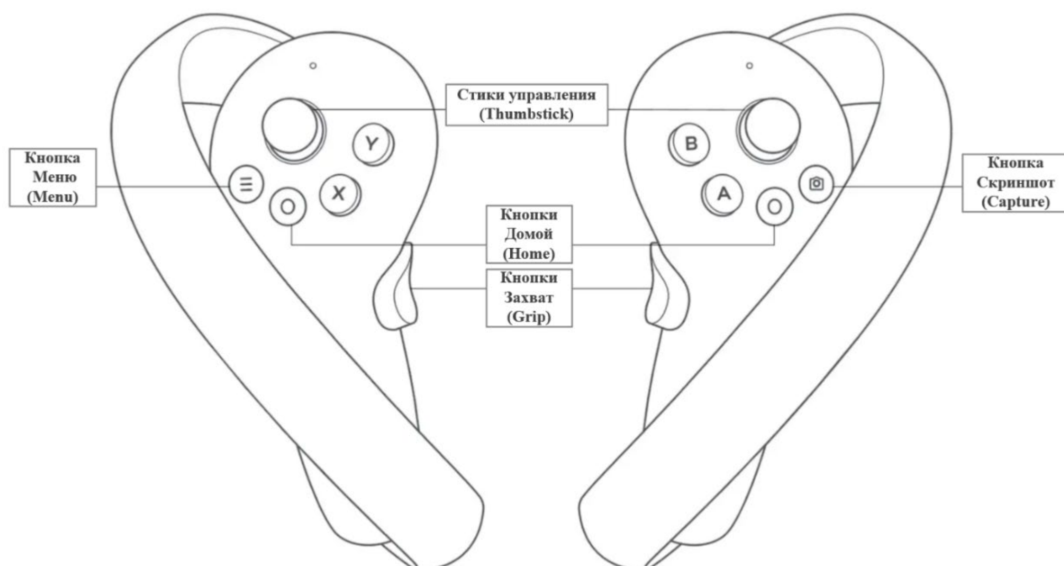

Рис. 4.2. Контроллеры *Pico 4*

Основные элементы управления, расположенные на контроллерах:

– *Thumbstick* – стики управления, предназначенные для управления большим пальцем руки. Используются для перемещения, поворота камеры и взаимодействия с интерфейсом. По принципу работы индентичны аналоговым джойстикам в игровых устройствах;

– кнопки *X* и *Y* (на левом контроллере), а также *A* и *B* (на правом контроллере) применяются для выполнения различных действий в зависимости от приложения. Они могут использоваться для подтверждения выбора, взаимодействия с объектами, прыжков и других команд. Например, кнопка *A* часто используется для подтверждения выбора, а кнопка *B* – для возврата или открытия меню паузы;

– кнопка *Menu* расположена на левом контроллере и предназначена для открытия меню приложения или вызова дополнительных функций. При подключении устройства к ПК она может использоваться для вызова интерфейса *SteamVR*;

– кнопка *Capture*, обозначенная значком камеры и расположенная на правом контроллере, предназначена для создания снимков экрана;

– кнопка *Home* расположена на обоих контроллерах и используется для возврата в главное меню устройства *Pico* 4, а также для вызова системного интерфейса;

– кнопки *Grip* расположены на боковых сторонах контроллеров и нажимаются средним пальцем. Они используются для захвата и удержания объектов в виртуальной среде.

В отличие от контроллеров, шлем *Pico* 4 имеет минимальное количество элементов управления. Кнопка питания (*Power*) расположена на левой стороне корпуса шлема (со стороны расположения камер) и используется для включения и выключения устройства. Для включения необходимо нажать и удерживать кнопку в течение нескольких секунд (рис. 4.3).

На верхней части корпуса шлема также расположены кнопки регулировки громкости встроенных динамиков.

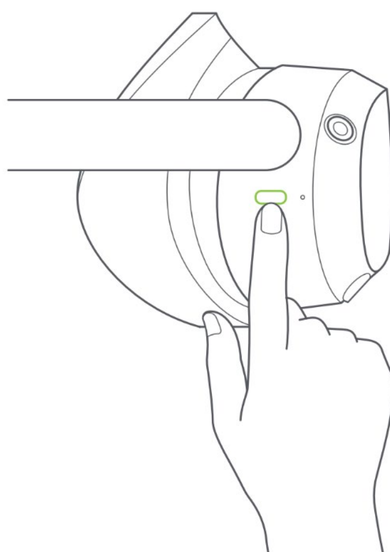

Рис. 4.3. Кнопка включения шлема *Pico* 4

Для подключения шлема виртуальной реальности к персональному компьютеру используется кабель с разъемом *USB Type-C*, поддерживающий стандарт *USB 3.0* или выше. Это необходимо для обеспечения стабильной передачи данных и минимизации задержек при передаче изображения.

Компьютер должен соответствовать следующим минимальным требованиям:

– процессор: *Intel Core i5-4590* / *AMD FX 8350* или более производительные решения;

– оперативная память: 8 ГБ и более;

– видеокарта: *NVIDIA GeForce GTX 1060* (6 ГБ) / *AMD Radeon RX 480* или более производительные решения;

– операционная система: *Windows 10* или *Windows 11*.

Перед началом работы необходимо запустить промежуточное программное обеспечение – приложение *SteamVR*. Взаимодействие шлема с *Unreal Engine* осуществляется через платформу *SteamVR* с использованием стандарта *OpenXR*, обеспечивающего унифицированный обмен данными и управление устройствами виртуальной реальности.

Для передачи данных между шлемом и компьютером используется специализированное программное обеспечение *PICO Connect* (ранее *Pico Streaming Assistant*). Данное приложение обеспечивает трансляцию изображения и взаимодействие с виртуальной средой по *Wi-Fi* или через *USB*-подключение.

После подключения шлема необходимо запустить *PICO Connect* как на компьютере, так и на устройстве виртуальной реальности. Далее следует выбрать способ подключения – через *USB*. После установления соединения компьютер автоматически распознает устройство (рис. 4.4).

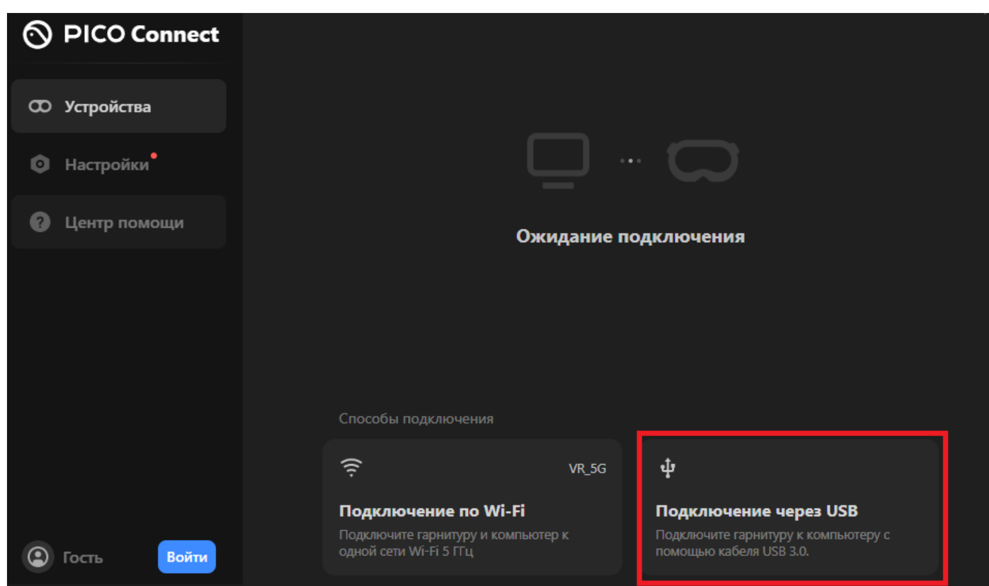

Рис. 4.4. Меню подключения по кабелю на ПК

В шлеме *Pico* 4 в разделе «Библиотека» необходимо найти приложение *Connect*. Данное приложение является аналогом программного обеспечения,

установленного на персональном компьютере, и используется для установления соединения (рис. 4.5).

Версии приложения на компьютере и в шлеме должны совпадать либо быть совместимыми. В случае несоответствия необходимо выполнить обновление обоих приложений.

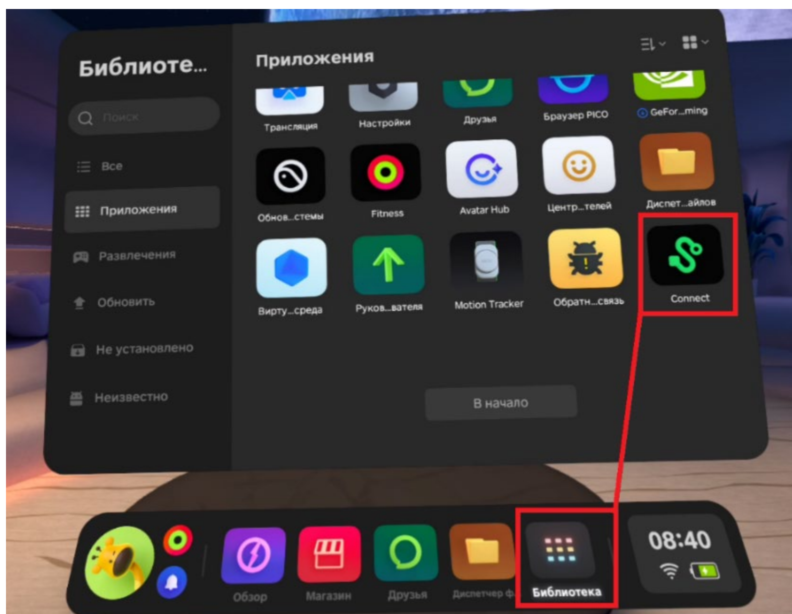

Рис. 4.5. Приложение для подключения в шлеме *VR*

После запуска приложения *Connect* следует убедиться, что подключение осуществляется через кабель стандарта *USB 3.0*. Далее необходимо выбрать требуемый персональный компьютер из списка доступных устройств.

Если один и тот же компьютер подключен по кабелю и находится в одной локальной сети, он может отображаться в списке несколько раз – в этом случае следует выбрать нужный вариант подключения. После выбора необходимо дождаться установления соединения (рис. 4.6).

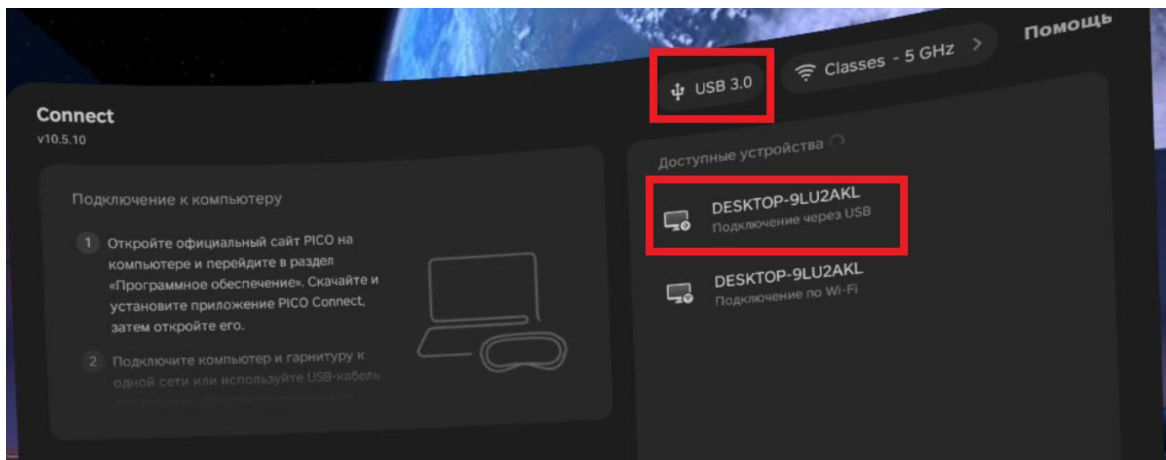

Рис. 4.6. Меню подключения по кабелю в шлеме *VR*

Когда приложение *Steam VR* запущено, а шлем *Pico 4* подключен к персональному компьютеру с использованием *PICO Connect*, можно приступить к созданию проекта в *Unreal Engine* с использованием стандартных настроек для виртуальной реальности (рис. 4.7).

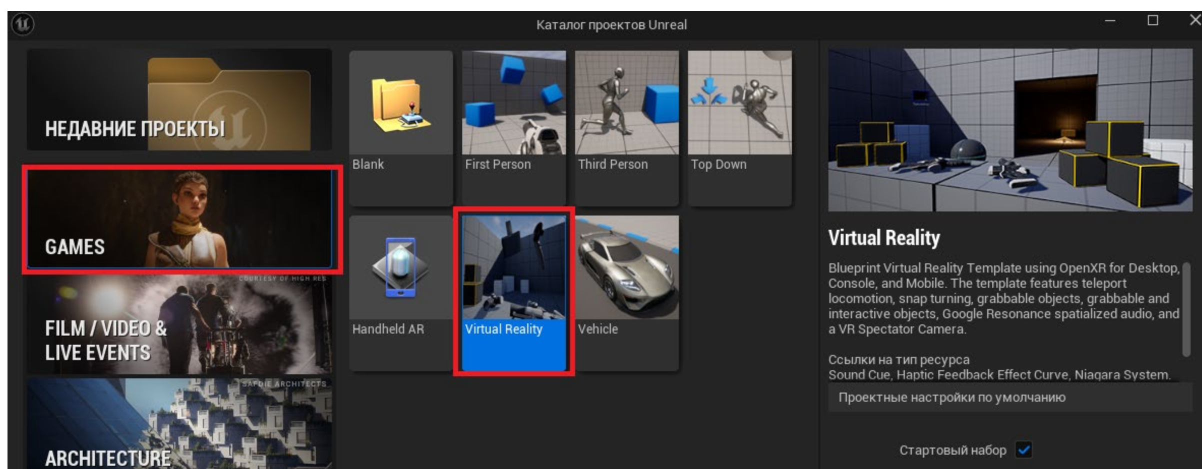

Рис. 4.7. Создание нового проекта для *VR*

При первом запуске VR-проекта в *Unreal Engine* выполняется длительная компиляция шейдеров. В зависимости от характеристик вычислительной системы данный процесс может занимать значительное время, вплоть до одного часа. Необходимо дождаться завершения компиляции перед переходом к следующему этапу работы (рис. 4.8).

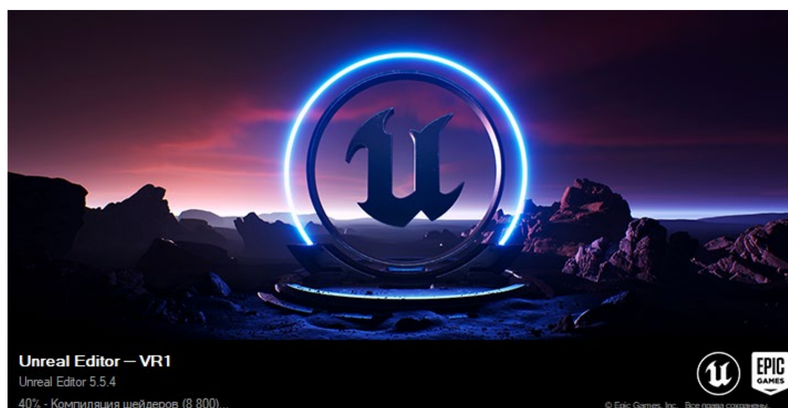

Рис. 4.8. Компиляция шейдеров для *VR*-проекта

Когда процесс завершится, будет открыт стандартный проект с набором тестовых объектов. Он включает базовые элементы сцены и объекты для взаимодействия, предназначенные для изучения и первоначальной отладки виртуальной среды. Рекомендуется внимательно ознакомиться со всеми элементами сцены, чтобы понять структуру проекта и возможные варианты взаимодействия (рис. 4.9).

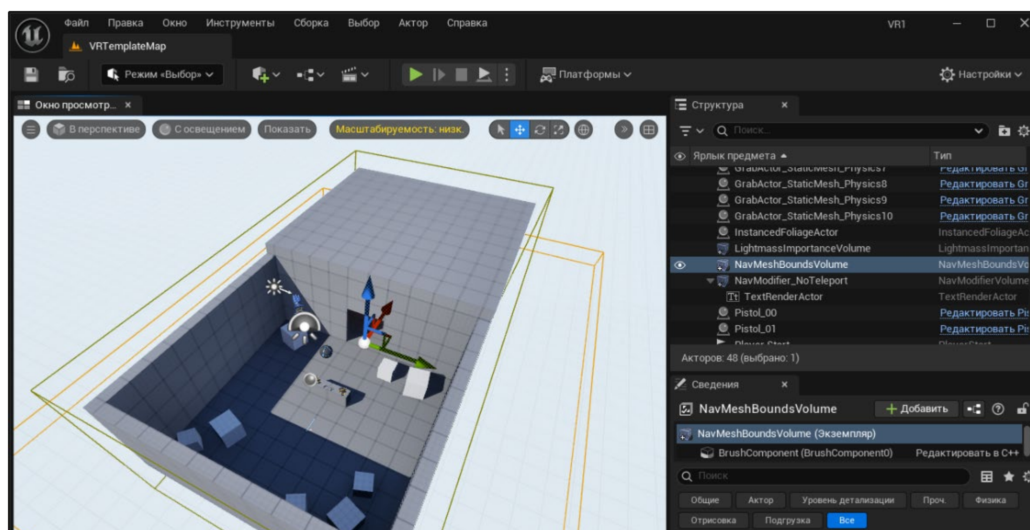

Рис. 4.9. Запущенная стандартная *VR*-сцена

Перед запуском VR-проекта необходимо переключить шлем в приложении *PICO Connect* в режим работы через *Steam VR*. Данная настройка выполняется с помощью специальной кнопки, расположенной в верхнем

правом углу окна трансляции. Без выполнения этого действия запуск проекта в шлеме будет невозможен, поскольку *Unreal Engine* взаимодействует с устройством через интерфейс *Steam VR* (рис. 4.10).

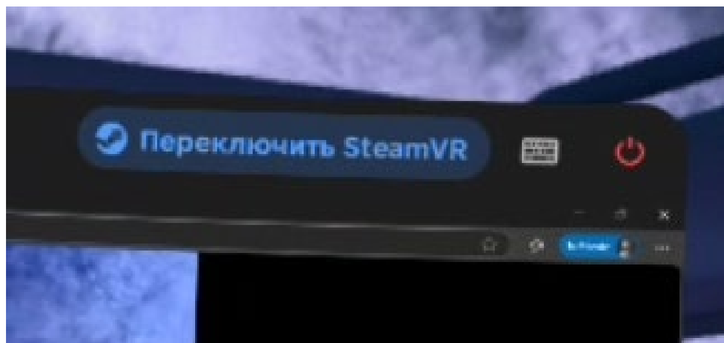

Рис. 4.10. Переключение шлема на *Steam VR*

Найдите в *Unreal Engine* панель запуска проекта, нажмите на кнопку с тремя точками в правой части панели и выберите из выпадающего списка пункт «Предпросмотр в VR» для запуска проекта (рис. 4.11).

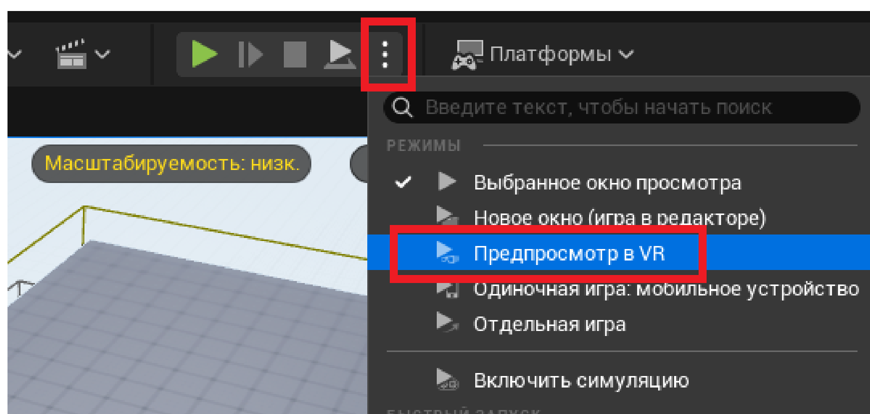

Рис. 4.11. Запуск проекта для предпросмотра в *VR*

После нажатия на кнопку запуска проект откроется в отдельном окне и начнется трансляция изображения на подключенный *VR*-шлем.

На данном этапе можно протестировать сцену в реальном времени, проверить удобство взаимодействия, качество визуализации и общее поведение проекта в шлеме (рис. 4.12).

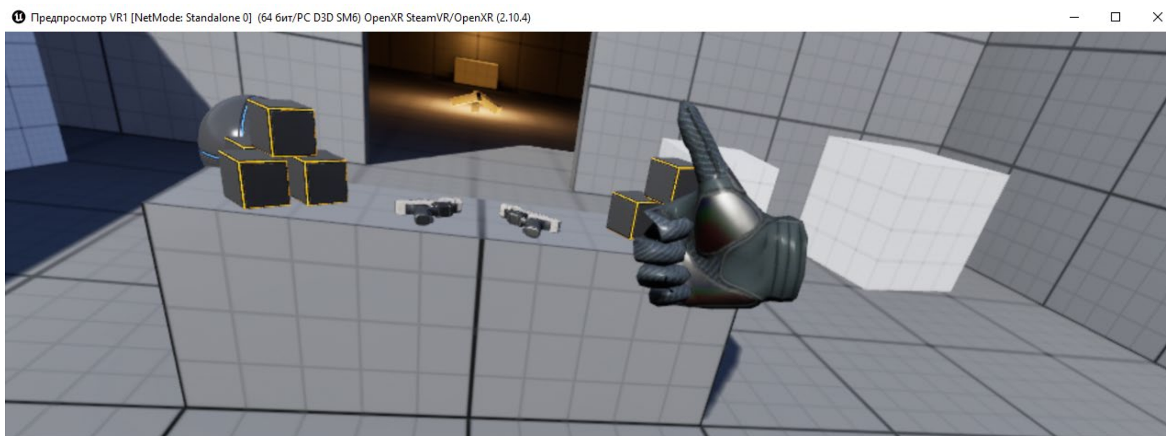

Рис. 4.12. Окно запущенного *VR*-проекта

Задание

Результат работы необходимо сохранить в виде проекта *Unreal Engine*, совместимого с версией 5.3 и выше. Название проекта должно соответствовать шаблону *Familiya_VR*.

Следует учитывать, что в названии проекта запрещено использование русских символов, пробелов и специальных знаков. Допускаются только латинские буквы, цифры и символ подчеркивания.

1. Подготовьте несколько собственных 3*D*-моделей в формате, совместимом с *Unreal Engine*, и импортируйте их в стандартный *VR*-проект, созданный ранее.
2. Запустите проект в режиме предпросмотра *VR* и проверьте отображение моделей, а также их восприятие в виртуальной среде с использованием шлема *Pico 4*.

Контрольные вопросы

1. Какие существуют способы подключения шлема виртуальной реальности к персональному компьютеру и для каких целей используется каждый из них?
2. Перечислите основные элементы управления на контроллерах *Pico 4* и укажите их назначение.
3. Что представляет собой *PICO Connect* и какова его роль при подключении шлема к персональному компьютеру?
4. Опишите порядок запуска проекта в режиме предпросмотра *VR* в *Unreal Engine* и условия, необходимые для его выполнения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Blender Foundation. Blender: Latest Reference Manual [Электронный ресурс]. – URL: https://docs.blender.org/manual/en/latest/ (дата обращения: 14.05.2025).

2. Cookson A., Dowlingsoka R., Crumpler C. Unreal Engine 4 Game Development in 24 Hours: Sams Teach Yourself. – Sams Publishing, 2016. – 528 с.

3. Epic Games. Unreal Engine 5.3 Documentation [Электронный ресурс]. – URL: https://dev.epicgames.com/documentation/en-us/unreal-engine (дата обращения: 14.05.2025).

4. Khan H. Virtual Filmmaking with Unreal Engine 5: Build High-end Short Films and Cinematics. – Packt Publishing, 2024. – 448 с.

5. Lospinoso J. C++ Crash Course: A Fast-Paced Introduction. – No Starch Press, 2019. – 816 с.

6. McCaffrey M. Unreal Engine VR Cookbook: Developing Virtual Reality with UE4. – Addison-Wesley Professional, 2017. – 432 с.

7. PICO XR Developer Center. PICO Unreal Integration [Электронный ресурс]. – URL: https://developer.picoxr.com/document/unreal/ (дата обращения: 14.05.2025).

8. Shannon T. Unreal Engine 4 for Design Visualization: Developing Stunning Interactive Visualizations, Animations, and Renderings. – Packt Publishing, 2017. – 400 с.

9. Sherif W. Learning C++ by Creating Games with Unreal Engine 4. – Packt Publishing, 2015. – 352 с.

Учебное издание

Головачев Никита Сергеевич
Бугаков Петр Юрьевич

ТЕХНОЛОГИИ ТРЕХМЕРНОГО МОДЕЛИРОВАНИЯ И ВИРТУАЛЬНОЙ РЕАЛЬНОСТИ

РАЗРАБОТКА ИНТЕРАКТИВНЫХ ТРЕХМЕРНЫХ СЦЕН

Редактор *О. В. Георгиевская*
Компьютерная верстка *А. П. Бочарниковой*

Изд. лиц. ЛР № 020461 от 04.03.1997.
Подписано в печать 03.06.2026. Формат 60 × 84 1/16.
Усл. печ. л. 5,98. Тираж 115 экз. Заказ 101.
Гигиеническое заключение
№ 54.НК.05.953.П.000147.12.02. от 10.12.2002.

Издательско-полиграфический центр СГУГиТ
630108, Новосибирск, ул. Плахотного, 10.

Отпечатано в издательско-полиграфическом центре СГУГиТ
630108, Новосибирск, ул. Плахотного, 8