**Принципы разработки игр на Unreal Engine**

**Бартов И.С РИС-24-2Б  
 Введение**

Разработка видеоигр в настоящее время представляет собой один из самых быстроразвивающихся сегментов IT. Unreal Engine — это мощный игровой движок, позволяющий реализовать высококачественные графические эффекты и поддерживать гибкие игровые механики. Он активно используется как в инди-разработках, так и в крупных коммерческих проектах благодаря своей масштабируемости и мощным инструментам.

Цель данного доклада — проанализировать возможности Unreal Engine как платформы для разработки игр, выявить основные принципы его использования и оценить его удобство и эффективность. Основными задачами являются изучение объектно-ориентированного программирования, системы Blueprint, BehaviorTree, программирования на C++ и сетевого взаимодействия, а также определение их влияния на процесс разработки и конечное качество проектов.

**1. Объектно-ориентированное программирование в Unreal Engine**

**Объектно-ориентированное программирование (ООП) - парадигма, в центре которой данные и объекты, а не функции.** Unreal Engine поддерживает ООП, что облегчает разработку сложных и масштабируемых проектов. Класс AActor является основным для всех игровых объектов, и позволяет описывать свойства и поведение объектов. Класс APawn наследуется от AActor и используется для объектов, которыми может управлять игрок, а ACharacter наследуется от APawn и добавляет функциональность, необходимую для персонажей, таких как передвижение и взаимодействие с окружением.

ООП в Unreal Engine позволяет создавать сложные иерархии классов, которые легко расширять и поддерживать. Например, создание базового класса врага, от которого наследуются более специализированные классы, добавляющие уникальные способности или поведение, помогает структурировать проект и сделать его более управляемым.

Переходя к следующему разделу, стоит отметить, что одной из уникальных возможностей Unreal Engine является система визуального программирования Blueprint, которая значительно упрощает работу с логикой игры.

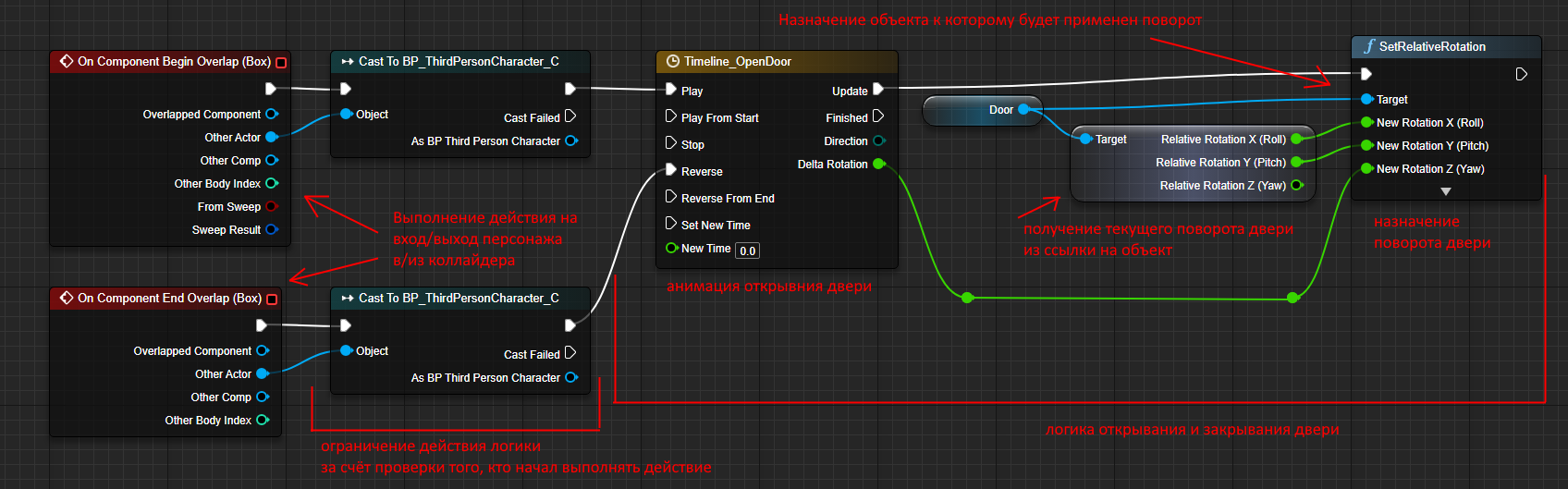
**2. Blueprint как визуальное программирование**

Blueprint или **чертежи**, предоставляет удобный интерфейс для проектирования логики игры без необходимости писать код на языке C++. Это упрощает работу над прототипами и ускоряет их итерацию. Blueprint — это одна из ключевых функций Unreal Engine, которая позволяет разработчикам всех уровней навыков быстро создавать игровые механики и взаимодействия.

Основные шаги при создании игровой механики через Blueprint:

1. **Создание класса Blueprint**: Разработчик создает новый класс Blueprint, наследующийся от AActor или другого базового класса. Этот процесс начинается с выбора нужного типа чертежа, что позволяет адаптировать функциональность под конкретные нужды.
2. **Добавление компонентов**: В Blueprint добавляются компоненты, такие как StaticMesh[[1]](#footnote-1) для визуализации объекта, Box Collider для определения границ столкновений, и Particle System для создания визуальных эффектов.
3. **Настройка обработчиков событий**: Blueprint предоставляет возможность использовать события, такие как OnClicked (нажатие на объект), BeginOverlap (пересечение границы коллайдера[[2]](#footnote-2)) и другие, что делает объект интерактивным и отзывчивым на действия игрока.
4. **Определение действия**: с помощью узлов в графе событий можно определить, что произойдет при выполнении того или иного действия, будь то воспроизведение анимации, изменение состояния объекта, отправка сигнала в другие объекты, изменение материалов или даже запуск физического симулятора.

Одним из примеров может быть создание системы дверей. Разработчик создает Blueprint для двери, добавляет StaticMesh для визуального отображения и настраивает коллайдеры. Затем, с помощью Blueprint, можно настроить логику открывания и закрывания двери по нажатию кнопки или при подходе персонажа. Это позволяет легко тестировать и изменять поведение двери, не переписывая код.



*Рис.1 Пример кода открывания и закрывания Двери на Blueprint*

Использование системы Blueprint значительно упрощает разработку игровых механик и ускоряет процесс создания прототипов. Это делает Unreal Engine удобным инструментом для как опытных разработчиков, так и начинающих специалистов, позволяя сосредоточиться на творческих аспектах создания игры*.*

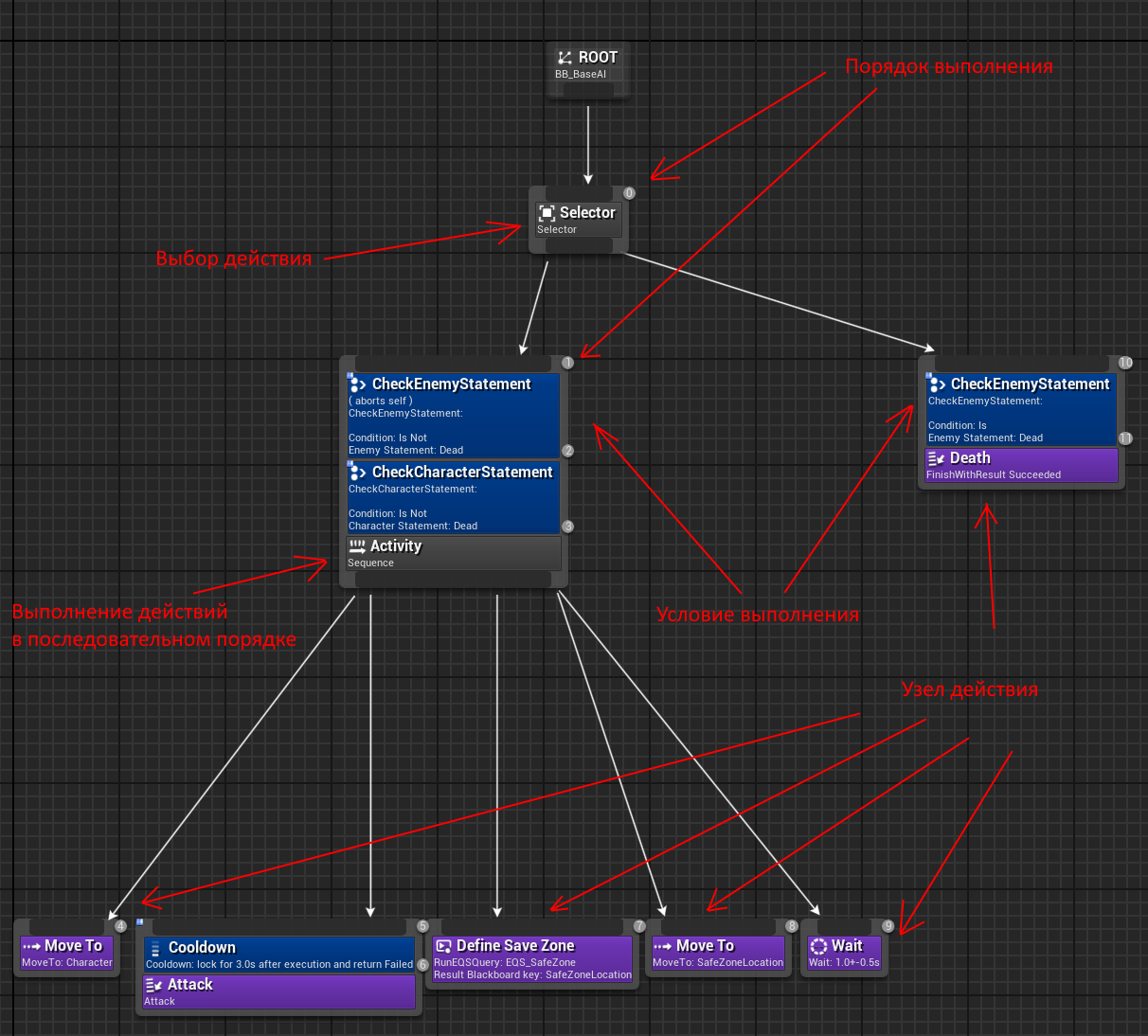
Следующей важной составляющей является BehaviorTree, который используется для управления поведением персонажей с помощью алгоритмов искусственного интеллекта.

**3. BehaviorTree для алгоритмов искусственного интеллекта**

BehaviorTree используется для описания поведения персонажей, таких как, например, патрулирование точек на карте. BehaviorTree состоит из узлов, которые представляют задачи, условия и действия. Основные элементы включают:

* **Selector**: Узел, который выбирает одно из нескольких действий для выполнения в зависимости от успеха или провала предыдущих задач.
* **Sequence**: Узел, выполняющий действия по порядку, останавливаясь на первом неуспешном.
* **Decorator**: Специальные условия, которые определяют, может ли узел быть выполнен в данный момент.
* **Service**: Сервисы позволяют обновлять данные на протяжении выполнения определенного действия, что полезно для реактивного поведения.

**Пример создания патруля с BehaviorTree:**

****

*Рис.2 Пример алгоритма поведения врага, сделанный с помощью Behavior Tree*

В этом примере враг будет патрулировать между двумя точками и, если увидит игрока, переключится на атаку. Для этого создается узел Sequence, который включает в себя два действия: MoveTo (перемещение к следующей точке патруля) и Wait (ожидание на точке). Если персонаж встречает игрока (через узел Blackboard), система переключается на узел Attack, который инициирует атаку на игрока.

Это поведение можно легко адаптировать для разных типов персонажей и использовать для создания более сложных паттернов действий. Например, можно добавить дополнительные условия, чтобы враг начал искать укрытия или вызвать подкрепление, если его здоровье низкое. [1]

Продвигаясь к следующему разделу, рассмотрим, как использование языка C++ в Unreal Engine позволяет разрабатывать высокопроизводительные решения для сложных игровых механик.

**4. Программирование на C++**

C++ остается неотъемлемой частью разработки на Unreal Engine. Он позволяет реализовать классы с высокой производительностью и точным контролем за игровым процессом. Через C++ можно добавлять новые функции и возможности, создавать специализированные игровые механики, а также взаимодействовать с низкоуровневыми компонентами движка.

Одним из самых больших преимуществ Unreal Engine является возможность комбинировать C++ и Blueprint. Это позволяет использовать C++ для создания высокопроизводительных компонентов и сложной логики, а затем использовать Blueprint для быстрого создания взаимодействий и тестирования идей. Например, можно создать новый компонент на C++, который обрабатывает физику автомобиля, и затем использовать Blueprint для настройки поведения автомобиля в зависимости от условий окружающей среды.

Создание игрового персонажа с помощью C++ включает в себя наследование от класса ACharacter и реализацию функций для управления передвижением. В этом процессе также можно использовать системы ввода (Input System) для назначения клавиш управления и реализации функций прыжка, атаки и взаимодействия с окружающим миром.

C++ также позволяет оптимизировать производительность проекта. Например, при разработке игры с большим количеством врагов, обработка их поведения на Blueprint может стать менее эффективной. Использование C++ для основных расчетов и алгоритмов значительно снижает нагрузку на систему и обеспечивает плавную работу. [3]

**5. Сетевое взаимодействие и репликация в Unreal Engine**

Для создания многопользовательских игр Unreal Engine предлагает мощные инструменты для сетевого программирования и репликации объектов. С помощью Blueprint и C++ можно создавать и настраивать сетевые взаимодействия, используя классы, такие как APlayerController и AGameMode.

**Репликация** используется для синхронизации состояния объектов между сервером и клиентами. Unreal Engine автоматически обрабатывает репликацию переменных, что позволяет разработчикам легко создавать многопользовательский геймплей. Например, перемещение персонажа может быть реплицировано, чтобы все игроки видели одинаковую позицию друг друга. Blueprint также поддерживает репликацию, что делает его удобным для создания простых многопользовательских механик без необходимости глубокого знания сетевого программирования.

Unreal Engine поддерживает интеграцию с внешними библиотеками и плагинами, что позволяет расширять возможности движка. Разработчики могут использовать C++ для интеграции различных SDK[[3]](#footnote-3), таких как системы аналитики, рекламные сети или специфические API[[4]](#footnote-4) для работы с устройствами. Это делает Unreal Engine универсальной платформой для создания как игровых, так и неигровых приложений. [3]

**Вывод**

Анализ принципов показывает, что Unreal Engine предоставляет разработчикам широкий спектр инструментов и возможностей для создания высококачественных видеоигр. Объектно-ориентированное программирование обеспечивает структурированность и масштабируемость проектов, позволяя эффективно управлять сложными игровыми механиками. Система визуального скриптинга Blueprint ускоряет процесс разработки и делает его доступным для специалистов с разным уровнем программных навыков. Интеграция системы BehaviorTree предоставляет мощные инструменты для создания интеллектуального и адаптивного искусственного интеллекта, а использование языка C++ позволяет реализовать высокопроизводительные и оптимизированные игровые решения. В совокупности эти принципы делают Unreal Engine незаменимым инструментом для современных разработчиков игр.

**Заключение**

Unreal Engine представляет собой один из самых мощных и гибких игровых движков на современном рынке, предлагая разработчикам обширные возможности для создания как небольших инди-проектов, так и масштабных коммерческих игр. Принципы разработки на Unreal Engine, основанные на объектно-ориентированном программировании, системе визуального скриптинга Blueprint, интеграции системы BehaviorTree и использовании языка C++, обеспечивают высокую производительность, гибкость и удобство в реализации игровых механик. Практическая значимость Unreal Engine подтверждается его широким применением в индустрии видеоигр и успешными проектами, созданными на его основе. Теоретическая значимость заключается в предоставлении структурированных и проверенных методологий разработки, которые могут быть использованы для создания сложных и масштабируемых игровых проектов. Таким образом, Unreal Engine является незаменимым инструментом для современных разработчиков игр, обеспечивая все необходимые средства для реализации самых амбициозных игровых идей.

**Список используемых источников**

1. Introduction to Blueprints:  
   https://dev.epicgames.com/documentation/en-us/unreal-engine/introduction-to-blueprints-visual-scripting-in-unreal-engine
2. Unreal Engine 4 Blueprint Программирование: <https://www.youtube.com/playlist?list=PL2suyruNHd0iZA-hThptrbmeDN2qbIlSe>
3. Programming with C++: https://dev.epicgames.com/documentation/en-us/unreal-engine/programming-with-cplusplus-in-unreal-engine

1. Static Mesh - основной тип отображения геометрии в Unreal Engine [↑](#footnote-ref-1)
2. Коллайдер - невидимая упрощённая форма объекта, привязанная к нему, для расчёта столкновений с другими объектами [↑](#footnote-ref-2)
3. SDK - Standard Development Kit (Стандартный пакет инструментов разработки) [↑](#footnote-ref-3)
4. API - Application Programming Interface (Программный интерфейс приложения) [↑](#footnote-ref-4)