Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Робототехника и комплексная автоматизация»

КАФЕДРА «Системы автоматизированного проектирования (РК-6)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Разработка системы сбора и исследования информации о фондовом рынке на Unreal Engine 4»

Студент РК6-81Б _____ **Афиногенов М.А.**
(Группа) (Подпись, дата) (Фамилия И.О.)

Руководитель ВКР _____ **Витюков Ф.А.**
(Подпись, дата) (Фамилия И.О.)

Нормоконтролёр _____ **Грошев С.В.**
(Подпись, дата) (Фамилия И.О.)

2024 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой РК6
(Индекс)
_____ Карпенко А.П.
(Фамилия И.О.)
«15» февраля 2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра

Студент группы РК6-81Б

Афиногенов Михаил Алексеевич
(фамилия, имя, отчество)

Тема квалификационной работы: «Разработка системы сбора и исследования информации о фондовом рынке на Unreal Engine 4»

Источник тематики (НИР кафедры, заказ организаций и т.п.): кафедра.

Тема квалификационной работы утверждена распоряжением по факультету РК №___ от «__» _____ 2024 г.

Техническое задание

Часть 1. *Аналитическая часть*

Изучить средства разработки, предоставляемые трехмерным движком Unreal Engine 4. Изучить методы оптимизации отрисовки большого колчества трехмерных объектов на сцене. Изучить методы сохранения объектов на сцене.

Часть 2. *Практическая часть 1. 3d-моделирование*

Создать функциональные интерактивные элементы и разработать систему их размещения на сцене. Осуществить корректный перенос финансовых графиков из формата CSV в графическое представление на сцене в Unreal Engine 4. Реализовать системы размещения финансовых графиков.

Часть 3. *Практическая часть 2. Разработка внутриигровых систем*

Реализовать возможность выбора, размещения, перемещения, редактирования и удаления функциональных интерактивных элементов на сцене в Unreal Engine 4. Реализовать возможность смены финансового графика с сохранением и дальнейшей загрузкой функциональных элементов.

Оформление выпускной квалификационной работы

Расчетно-пояснительная записка на 55 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

Работа содержит 8 графических листов формата А4.

Дата выдачи задания: «10» февраля 2024 г.

В соответствии с учебным планом выпускную квалификационную работу выполнить в полном объеме в срок до «21» июня 2024 г.

Руководитель квалификационной работы _____ **Витюков Ф.А.**
(Подпись, дата) (Фамилия И.О.)

Студент _____ **Афиногенов М.А.**
(Подпись, дата) (Фамилия И.О.)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ РК

КАФЕДРА РК6

ГРУППА РК6-81Б

УТВЕРЖДАЮ

Заведующий кафедрой РК6
(Индекс)

_____ Карпенко А.П.
(Фамилия И.О.)

«15» февраля 2024 г.

КАЛЕНДАРНЫЙ ПЛАН
выполнения выпускной квалификационной работы

Студент группы РК6-81Б

Афиногенов Михаил Алексеевич
(фамилия, имя, отчество)

Тема квалификационной работы: «Разработка системы сбора и исследования информации о фондовом рынке на Unreal Engine 4»

№ п/п	Наименование этапов выпускной квалификационной работы	Сроки выполнения этапов		Отметка о выполнении	
		план	факт	Должность	ФИО, подпись
1.	Задание на выполнение работы. Формулирование проблемы, цели и задач работы	10.02.2024	_____	Руководитель ВКР	Витюков Ф.А.
2.	1 часть. Аналитическая часть	24.02.2024	_____	Руководитель ВКР	Витюков Ф.А.
3.	Утверждение окончательных формулировок решаемой проблемы, цели работы и перечня задач	03.03.2024	_____	Заведующий кафедрой	Карпенко А.П.
4.	2 часть. Практическая часть 1	21.04.2024	_____	Руководитель ВКР	Витюков Ф.А.
5.	3 часть. Практическая часть 2	25.05.2024	_____	Руководитель ВКР	Витюков Ф.А.
6.	1-я редакция работы	30.05.2024	_____	Руководитель ВКР	Витюков Ф.А.
7.	Подготовка доклада и презентации	07.06.2023	_____	Студент	Котельникова Е.Ю.
8.	Заключение руководителя	10.06.2024	_____	Руководитель ВКР	Витюков Ф.А.
9.	Допуск работы к защите на ГЭК (нормоконтроль)	17.06.2024	_____	Нормоконтролер	Грошев С.В.
10.	Внешняя рецензия	17.06.2024	_____		
11.	Защита работы на ГЭК	26.06.2024	_____		

Студент _____ Афиногенов М.А. Руководитель ВКР _____ Витюков Ф.А.
(подпись, дата) (Фамилия И.О.) (подпись, дата) (Фамилия И.О.)

АННОТАЦИЯ

В данной работе рассмотрены подходы для создания некоторых элементов пользовательского интерфейса для сбора и редактирования информации о фондовых рынках. Описана разработка трехмерных интерактивных графических элементов. Рассмотрена реализация редактирования графических элементов. Также описаны этапы оптимизации 3D-моделей интерактивных элементов для дальнейшего использования в движке. Проведен анализ эффективности использования компонентной реализации, выполнено сравнение производительности реализующих его компонентов в движке Unreal Engine 4.

В расчётно-пояснительной записке 55 страниц, 7 рисунков, 8 графических листов.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы разработка интерактивных элементов пользовательского интерфейса (UI) стала неотъемлемой частью создания современных видеоигр и приложений виртуальной реальности. Популярность трехмерного движка Unreal Engine 4 значительно возросла, поскольку этот движок предоставляет широкий спектр инструментов и функций для создания высококачественных и интерактивных UI.

В данной научно-исследовательской работе будут рассмотрены методы разработки интерактивных элементов пользовательского интерфейса в Unreal Engine 4. В начале будет представлен обзор основных компонентов UE4, которые необходимы для создания UI, а затем будут приведены примеры различных методов разработки, включая использование скриптов, редактирование XML и создание пользовательских компонентов. Перед тем как начать разработку UI в UE4, необходимо ознакомиться с основными компонентами движка.

Один из ключевых компонентов — это Viewport, который представляет собой окно, отображающее игровой мир. В этом окне пользователь может взаимодействовать с объектами и элементами UI.

Другой важный компонент — это HUD (Head-Up Display), который отображает информацию о игре. HUD обычно располагается в верхней части экрана и содержит различные элементы, такие как кнопки и шкалы. Для создания интерактивных элементов UI в UE4 можно использовать несколько методов. Один из них — это использование скриптов. Скрипты позволяют программистам создавать сложные функции и поведения для элементов UI. Например, можно написать скрипт, который реагирует на нажатие кнопки и изменяет цвет фона или анимацию объекта.

Еще один метод разработки UI — это редактирование XML. В UE4 все элементы UI описываются в формате XML. Это позволяет разработчикам создавать сложные макеты и настраивать поведение элементов. Например, можно создать XML-файл, который определяет расположение и размер кнопок на экране.

Кроме того, в UE4 можно создавать пользовательские компоненты. Это позволяет разработчикам создавать уникальные элементы UI, которые соответствуют специфическим требованиям проекта. Например, можно создать пользовательский компонент, который отображает текстовые сообщения или анимации.

Выполнение данного курсового проекта подразумевает под собой разработку, проектировку и реализацию в виде кода части функционала сайта tradingview.com. Далее показано возможное расположение и отображение необходимых элементов на сайте tradingview.com [5] (рис. 1)

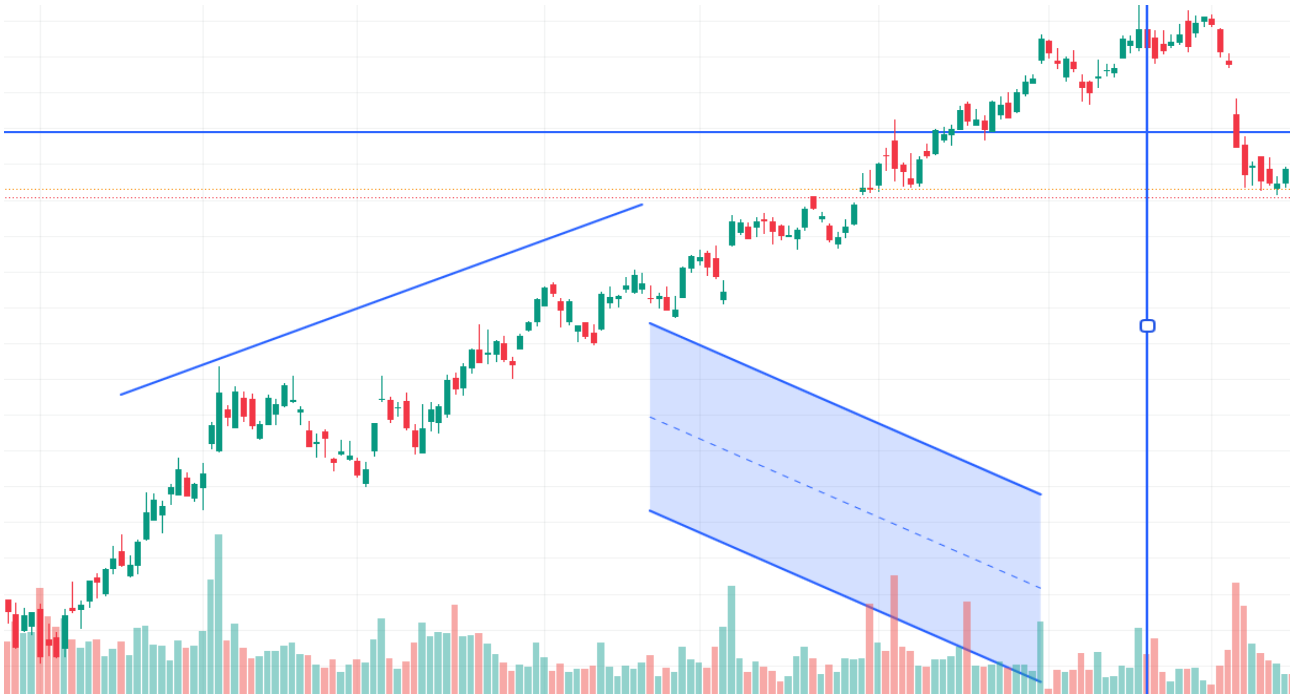

Рисунок 1. Расположение необходимых объектов на сайте tradingview.com [5]

В ходе данного курсового проекта те же самые функциональные элементы были реализованы на базе игрового движка Unreal Engine 4 (рис. 2)

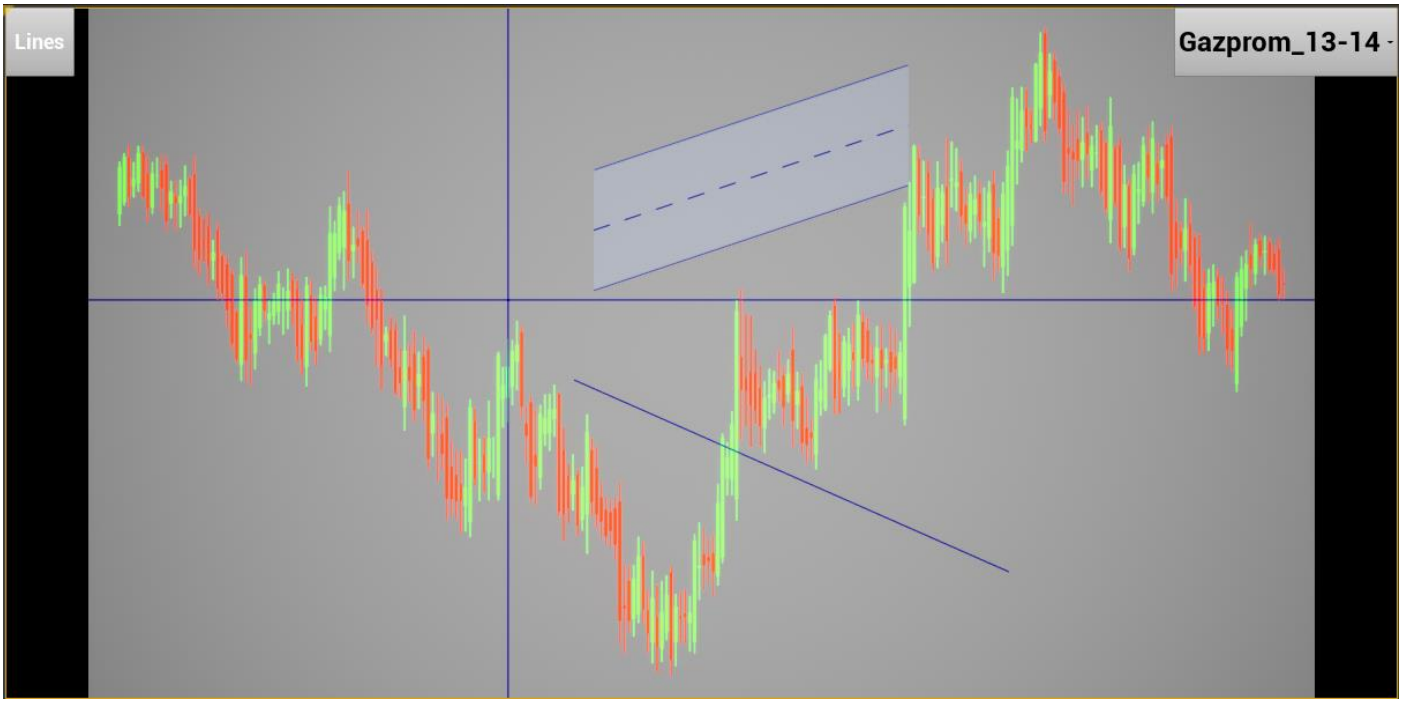

Рисунок 2. Расположение необходимых объектов на Unreal Engine 4

Следует отметить, что разработка интерактивных элементов пользовательского интерфейса в Unreal Engine 4 является достаточно простой благодаря широкому спектру инструментов и функций, предоставляемых движком. Использование скриптов, редактирование XML и создание пользовательских компонентов — это лишь некоторые из методов разработки UI в UE4. Каждый метод имеет свои преимущества и недостатки, поэтому выбор метода зависит от конкретных требований проекта.

1. Обзор существующих решений

В разработке интерактивных элементов пользовательского интерфейса (UI) для трехмерных видеоигр и приложений виртуальной реальности с помощью движка Unreal Engine 4 используются различные методы и подходы. Основным компонентом, используемым для создания интерактивных объектов, является Static Mesh Component, который позволяет прикреплять к объектам классов AActor (далее акторам) или объектам в игровом мире детальную информацию о геометрии и материалах объекта.

Подходы к созданию интерактивности с использованием Static Mesh Component варьируются. Один из них включает использование уже готовых моделей и анимаций. В Unreal Engine 4 существует обширный набор таких моделей и анимаций, которые можно адаптировать для создания интерактивных элементов UI. Например, можно использовать модель двери с анимацией открытия и закрытия, чтобы создать кнопку-переключатель.

Второй подход включает создание собственных моделей и анимаций. Unreal Engine 4 предлагает мощные инструменты для создания и редактирования геометрии объектов и анимаций. Это позволяет разработчикам создавать уникальные интерактивные элементы UI, полностью соответствующие требованиям их проектов.

Третий подход включает программирование интерактивности с использованием Blueprint — визуального языка программирования в Unreal Engine 4. С помощью Blueprint разработчики могут легко добавлять логику взаимодействия с интерактивными элементами UI. Например, можно программировать обработку нажатия на кнопку или изменение значения ползунка.

Кроме того, Unreal Engine 4 позволяет использовать физическую симуляцию для создания более реалистичной интерактивности. Например, при нажатии на кнопку она может отталкиваться от пальца пользователя или перемещаться по заданной траектории.

В целом, методы разработки трехмерных интерактивных элементов и моделей на движке движке Unreal Engine 4 обширны и позволяют достигать

высокого уровня реализма и функциональности. Они предоставляют возможность создания уникальных и привлекательных игровых интерфейсов, которые могут улучшить впечатление пользователей от игры.

Эти методы включают в себя не только технические аспекты создания интерактивных элементов, но и учет пользовательского опыта и интерфейсного дизайна. Понимание того, как пользователи взаимодействуют с трехмерным интерфейсом, и интеграция этого понимания в процесс разработки являются ключевыми для создания эффективных и интуитивно понятных пользовательских интерфейсов. Это требует тщательного анализа пользовательского поведения и предпочтений, а также тестирования и итеративного улучшения интерфейса на основе обратной связи от пользователей.

Разработка интерактивных элементов UI в трехмерном пространстве представляет собой сложную, но в то же время захватывающую задачу, которая открывает новые возможности для разработчиков и дизайнеров в области создания виртуальных миров и игровых интерфейсов. Благодаря инновационным подходам и технологиям, таким как Unreal Engine 4, сфера разработки видеоигр и виртуальной реальности продолжает развиваться, предоставляя все более мощные и эффективные инструменты для создания впечатляющих и вовлекающих пользовательских интерфейсов.

1.1. Оптимизация отрисовки трехмерных объектов

Одним из важных аспектов разработки UI в UE4 является оптимизация отображения трехмерных элементов. Для этого используются различные техники, включая Static Mesh Component, Instance Static Mesh и другие.

Static Mesh Component [3] - это компонент, который представляет собой статичный объект, состоящий из геометрии и материалов. Он используется для отображения трехмерных объектов в игровом мире. Static Mesh Component имеет ряд преимуществ, таких как более быстрая загрузка и меньшее количество ресурсов, необходимых для отображения объекта. Однако, его использование может привести к проблемам с производительностью, особенно при работе с большим количеством объектов.

Instance Static Mesh [3] - это техника, которая позволяет использовать один и тот же экземпляр Static Mesh Component для отображения нескольких объектов. Это позволяет сократить количество ресурсов, необходимых для отображения объектов, и улучшить производительность игры. Однако, использование Instance Static Mesh может быть ограничено в некоторых случаях, например, когда требуется отображение динамически изменяющихся объектов.

В целом, правильное использование Static Mesh Component и Instance Static Mesh может значительно повлиять на оптимизацию отображения трехмерных элементов в UE4. Однако, выбор метода зависит от конкретных требований проекта и требует тщательного анализа и тестирования.

Оптимизация отрисовки трехмерных объектов в контексте разработки интерактивных элементов пользовательского интерфейса в Unreal Engine 4 является ключевым аспектом для повышения эффективности и производительности игр и приложений. Основной компонент, используемый в этом процессе, - это Static Mesh Component, который позволяет добавлять статические 3D-объекты в игровой мир и контролировать их отображение и поведение.

Одним из методов оптимизации является упрощение геометрии объектов. Использование моделей с меньшим количеством полигонов снижает нагрузку на процессор и видеокарту, что приводит к улучшению производительности. Такой подход

позволяет сохранить визуальное качество при одновременном уменьшении ресурсоемкости.

Другой важный метод оптимизации – это снижение количества отдельных отображаемых объектов на сцене. Важность использования этого метода можно проиллюстрировать следующим примером – в том случае, когда один объект, созданный программистом, состоит из множества разных менее масштабных объектов, которые представляют собой объекты разных классов, их совместная отрисовка очень сильно усложняется как с точки зрения производительности, так и с точки зрения решений по редактированию этих объектов и компонентов. Оптимальным решением в данной ситуации будет создать один «родительский» объект класса, а остальные «дочерние» объекты интерпретировать как компоненты этого объекта, что позволяет упростить создание, дальнейшее изменение и редактирование данного объекта, а также существенно снижает нагрузку на вычислительные элементы компьютера.

Также важно использование текстурных атласов, которые позволяют объединять несколько текстур в одну. Это снижает количество вызовов к видеопамяти и ускоряет процесс отрисовки. Unreal Engine 4 обеспечивает возможности для создания и эффективного использования текстурных атласов, что способствует оптимизации работы с текстурами.

В целом, использование Static Mesh Component в Unreal Engine 4 позволяет проводить эффективную оптимизацию отрисовки трехмерных объектов. Это включает в себя не только технические аспекты, но и учет визуального качества и пользовательского опыта. Правильное использование данного компонента способствует созданию плавной и реалистичной игровой среды, улучшая общее впечатление от игры.

Кроме того, оптимизация в Unreal Engine 4 включает в себя работу с освещением и тенями. Эффективное использование источников света и теней может значительно повышать реалистичность сцены, одновременно снижая нагрузку на систему. Методы, такие как динамическое освещение и отложенное рендеринг, обеспечивают гибкость и производительность при создании сложных освещенных сцен.

Наконец, важным аспектом оптимизации является профилирование и мониторинг производительности. Unreal Engine 4 предоставляет мощные инструменты для

мониторинга и анализа производительности, позволяя разработчикам точно определять узкие места и оптимизировать их. Профилирование помогает в выявлении проблем с производительностью на ранних этапах разработки, что способствует более эффективному и целенаправленному процессу оптимизации.

1.2 Преимущества использования Static Mesh Component

Static Mesh Component – это мощный инструмент, который позволяет разработчикам создавать сложные сцены, объединяя множество отдельных объектов в единое целое. Этот компонент является одним из ключевых элементов Unreal Engine 4 и предоставляет разработчикам множество возможностей для оптимизации производительности и упрощения процесса создания контента. Одной из главных причин использования Static Mesh Component является возможность быстрого и простого создания сложных сцен, которые могут содержать большое количество объектов. Вместо того чтобы создавать каждый объект отдельно и добавлять его на сцену, можно создать один Static Mesh и использовать его для представления всей группы объектов. Это позволяет существенно сократить время, затрачиваемое на создание и управление объектами, а также упростить процесс оптимизации производительности.

Кроме того, Static Mesh Component может использовать различные оптимизации, такие как отсечение и раннее прекращение отрисовки, чтобы уменьшить количество работы, которую нужно выполнить при отрисовке сцены. Это позволяет значительно повысить производительность при работе со сложными сценами и обеспечить плавную и реалистичную визуализацию. В целом, использование Static Mesh Component является более эффективным способом работы со сложными сценами в Unreal Engine 4. Он позволяет сократить время, затрачиваемое на создание и управление объектами, а также повысить производительность при работе со сложными сценами. Благодаря этому компоненту разработчики могут создавать более качественные игры и приложения, которые работают быстро и плавно даже на ограниченных ресурсах.

Static Mesh Component также предоставляет возможность для более точного контроля над визуальными эффектами и освещением объектов. Разработчики могут настроить различные параметры, такие как текстуры, материалы и освещение, для каждого объекта в сцене. Это позволяет создавать более реалистичные и привлекательные визуальные эффекты, которые улучшают пользовательский опыт и делают игру или приложение более привлекательными для игроков.

Также Static Mesh Component [4] обеспечивает более эффективное использование ресурсов системы. Вместо того чтобы обрабатывать каждый объект отдельно, система

может обрабатывать только один Static Mesh, что позволяет сократить нагрузку на процессор и графический процессор. Это особенно важно при работе с большими сценами или в условиях ограниченной аппаратной поддержки.

1.3 Необходимость использования Instance Static Mesh

В современных игровых движках, таких как Unreal Engine, оптимизация отрисовки графики является ключевым аспектом, определяющим производительность и качество визуального представления. Instance Static Mesh Component (ISM) предоставляет мощный механизм для оптимизации путем инстанцирования (instancing), что позволяет значительно снизить количество вызовов отрисовки (draw calls) и уменьшить нагрузку на центральный процессор (CPU) и графический процессор (GPU). В этом документе рассматриваются принципы работы ISM, механизмы его функционирования и степень, до которой можно оптимизировать отрисовку.

В рамках выпускной квалификационной работы (ВКР) необходимо реализовать отрисовку большого количества однотипных трехмерных объектов – в данном случае финансовых свечей. Для их грамотного представления реализованы 2 подтипа объектов: тело свечи и тени свечи. В данной конкретной работе используются подготовленные графики в формате CSV, данные из которых используются для отрисовки свечей. В свою очередь графики представлены в виде таблиц, где одна строка – одна свеча. Из таблицы получаются значения минимального и максимального значения свечи, точки открытия и закрытия свечи, ее время открытия и закрытия. В таблице 250 строк – соответственно, на выходе будет реализовано 250 свечей в рамках одного графика. Каждая свеча представлена 2 конструктивными элементами, и суммарно получается, что для того, чтобы отрисовать один график, необходимо обработать информацию по 250-и записям в таблице, после чего быстро отрисовать 500 элементов графика.

Основные проблемы с которыми можно столкнуться при решении данной задачи – это большая нагрузка на компьютер как при разработке, так и при использовании готовой работы, так как имеется необходимость в быстрой отрисовке большого количества элементов, а также необходимость быстрой и четкой обработки данных с полученных графиков. Обработка графиков была решена посредством перевода графиков, полученных с сайта, в формат CSV - Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми. – а затем при помощи специальной

структуры данных преобразована в таблицу с данными, где одна строка соответствует одной свече на графике. Большую ресурсоемкость задачи было решено оптимизировать с помощью функционала игрового движка Unreal Engine 4, в частности с использованием технологии Instance Static Mesh Component.

В данном случае Instance Static Mesh Component является наиболее оптимальным решением, так как эта технология позволяет объединить некоторое количество вызовов отрисовки (Draw Calls) в один, что снижает нагрузку на видеокарту и повышает скорость исполнения необходимых вызовов отрисовки.

Instance Static Mesh Component - это компонент, который используется для отрисовки повторяющихся объектов в игровом движке. Он позволяет создавать копии одного и того же статического меша на сцене без дополнительного расхода памяти на каждую копию. Это особенно важно при отрисовке большого количества объектов, так как позволяет существенно уменьшить нагрузку на процессор и GPU. Instance Static Mesh Component повышает производительность игры и позволяет создавать более крупные и детализированные сцены без потери в качестве графики.

Среди принципов работы Instance Static Mesh Component можно отметить следующее:

1. Группировка объектов и их отрисовка - в традиционной системе отрисовки каждый объект на сцене требует отдельного draw call, что означает отправку команды на GPU для отрисовки этого объекта. При большом количестве объектов, особенно идентичных, этот процесс создает значительную нагрузку на CPU, так как каждый draw call требует обработки и подготовки данных. Instance Static Mesh Component позволяет объединить все экземпляры одного и того же меша в один draw call. Это достигается путем создания одного набора данных меша, который используется для всех экземпляров, а также таблицы трансформаций, определяющей позицию, вращение и масштаб каждого экземпляра.
2. Снижение числа draw calls - Draw call — это команда, отправляемая от CPU к GPU для рендеринга конкретного объекта или группы объектов. При большом количестве объектов draw calls становятся узким местом в производительности, так как каждая команда требует значительных ресурсов

для обработки. Использование ISM позволяет сократить количество draw calls с тысяч до единиц. Например, вместо 1000 draw calls для 1000 объектов, ISM позволяет отправить всего 1 draw call, что значительно уменьшает нагрузку на CPU и увеличивает общую производительность.

3. Индивидуальные параметры трансформации - Одним из преимуществ ISM является возможность задания индивидуальных параметров трансформации для каждого экземпляра объекта. Это включает в себя позицию, вращение и масштаб, что позволяет создавать визуально разнообразные сцены с использованием одного и того же меша. Эти трансформации хранятся в виде таблицы, которая используется GPU для применения соответствующих изменений к каждому экземпляру во время отрисовки.

Использование Instance Static Mesh Component (Instanced Static Mesh или ISM) в Unreal Engine позволяет значительно оптимизировать отрисовку множества одинаковых объектов на сцене. Это достигается за счет использования техники instancing, которая уменьшает нагрузку на процессор и видеокарту. Вот основные аспекты, как это работает и какие выгоды это дает:

1. Группировка объектов: Вместо того чтобы обрабатывать каждый объект отдельно, Instance Static Mesh Component позволяет объединить все экземпляры одного и того же меша в один draw call. Draw call — это команда, отправляемая на GPU для отрисовки объектов. С уменьшением числа draw call, уменьшается и нагрузка на процессор. Чем больше количество экземпляров одного и того же меша на сцене, тем больше выгода от использования ISM. В сценах с тысячами идентичных объектов (например, леса, толпы, здания) уменьшение количества draw calls может быть значительным, приводя к существенному увеличению производительности.
2. Сокращение Draw calls: Если у вас есть 1000 одинаковых деревьев, которые нужно отрисовать, использование Instance Static Mesh вместо стандартного Static Mesh сократит количество draw call с 1000 до 1 (в идеальном случае).
3. Параметры трансформации: Instance Static Mesh позволяет каждому экземпляру объекта иметь свою трансформацию (позиция, вращение,

масштаб), что делает каждое дерево уникальным, но все они обрабатываются одной командой на GPU.

4. Сложные меши и материалы требуют больше ресурсов для обработки и рендеринга. При использовании ISM выигрыш в производительности будет более заметен для сложных объектов, так как каждый draw call требует значительных ресурсов для подготовки и обработки данных. Для простых мешей выигрыш может быть менее заметен, но все равно ощутим.
5. В сценах, содержащих множество мелких деталей и объектов, таких как леса, городские ландшафты или крупные массовки, использование ISM может значительно улучшить производительность. Такие сцены обычно требуют большого количества draw calls, что значительно нагружает CPU и GPU. Использование ISM позволяет уменьшить количество draw calls и повысить эффективность обработки.

Оптимизация отрисовки с использованием Instance Static Mesh зависит от многих факторов, но вот некоторые примеры и ориентировочные оценки:

1. Количество экземпляров: Чем больше количество экземпляров, тем больше выгода от использования Instance Static Mesh. Например, для сцены с 1000 и более одинаковых объектов, снижение количества draw calls может быть весьма значительным.
2. Сложность объектов: Если меши простые, то выигрыш может быть менее заметным, так как процессор и так быстро обрабатывает команды. Для более сложных объектов и сложных материалов выгода будет более ощутимой.
3. Тип сцены: В сценах, где множество мелких деталей (например, леса, городские сцены), использование Instance Static Mesh позволит существенно улучшить производительность.
4. GPU и CPU: Современные GPU отлично справляются с instancing, и в большинстве случаев производительность возрастет на порядок, если использовать Instance Static Mesh Component для повторяющихся объектов.

Также использование Instance Static Mesh Component обеспечивает положительное влияние на компоненты рабочей машины пользователя и разработчика, так как:

1. Происходит снижение времени на CPU - Оптимизация в плане снижения нагрузки на CPU может достигать 70-90%, так как уменьшается количество draw call и пересчетов трансформаций. В традиционной системе рендеринга каждый draw call требует значительных ресурсов для подготовки и отправки данных на GPU. При большом количестве объектов это становится узким местом, ограничивающим производительность. ISM позволяет сократить количество draw calls, что снижает нагрузку на CPU и освобождает ресурсы для других задач, таких как обработка логики игры и физики.
2. Происходит снижение времени на GPU: На GPU можно ожидать улучшения производительности в диапазоне от 20% до 80%, в зависимости от сложности сцены и количества объектов. Современные GPU оптимизированы для работы с большими объемами данных и параллельной обработкой. Техника инстанцирования использует эти возможности, позволяя GPU обрабатывать множество объектов за один цикл отрисовки. Это достигается путем передачи одного набора данных меша и таблицы трансформаций, что уменьшает объем данных, передаваемых между CPU и GPU, и снижает количество draw calls.
3. Инстанцирование позволяет более эффективно использовать кэш и память GPU, так как данные меша хранятся в едином экземпляре и повторно используются для всех экземпляров. Это уменьшает количество операций чтения-записи и повышает общую производительность системы.

Instance Static Mesh Component в Unreal Engine является мощным инструментом для оптимизации отрисовки, особенно в сценах с большим количеством идентичных объектов. Техника инстанцирования позволяет значительно уменьшить количество draw calls, снизить нагрузку на CPU и улучшить общую производительность системы. Оптимизация отрисовки с использованием ISM зависит от множества факторов, включая количество экземпляров, сложность объектов, тип сцены и характеристики оборудования. В большинстве случаев использование ISM приводит к значительному

улучшению производительности, что делает его ключевым инструментом для создания высокоэффективных и визуально насыщенных сцен в современных играх.

Instance Static Mesh Component также обладает удобным интерфейсом для управления и настройки каждой копии меша на сцене. Это позволяет программистам и художникам быстро и эффективно создавать сложные сцены с большим количеством деталей. Кроме того, использование данного компонента позволяет легко изменять параметры каждой копии меша, такие как масштаб, поворот, положение, что делает процесс настройки сцены более гибким и удобным.

Еще одним преимуществом Instance Static Mesh Component является его эффективное управление памятью и ресурсами. Поскольку компонент использует один и тот же меш для всех копий, это существенно снижает объем занимаемой памяти и ускоряет процесс загрузки игры. Благодаря этому игровые сцены могут быть более крупными и детализированными, не теряя при этом в производительности.

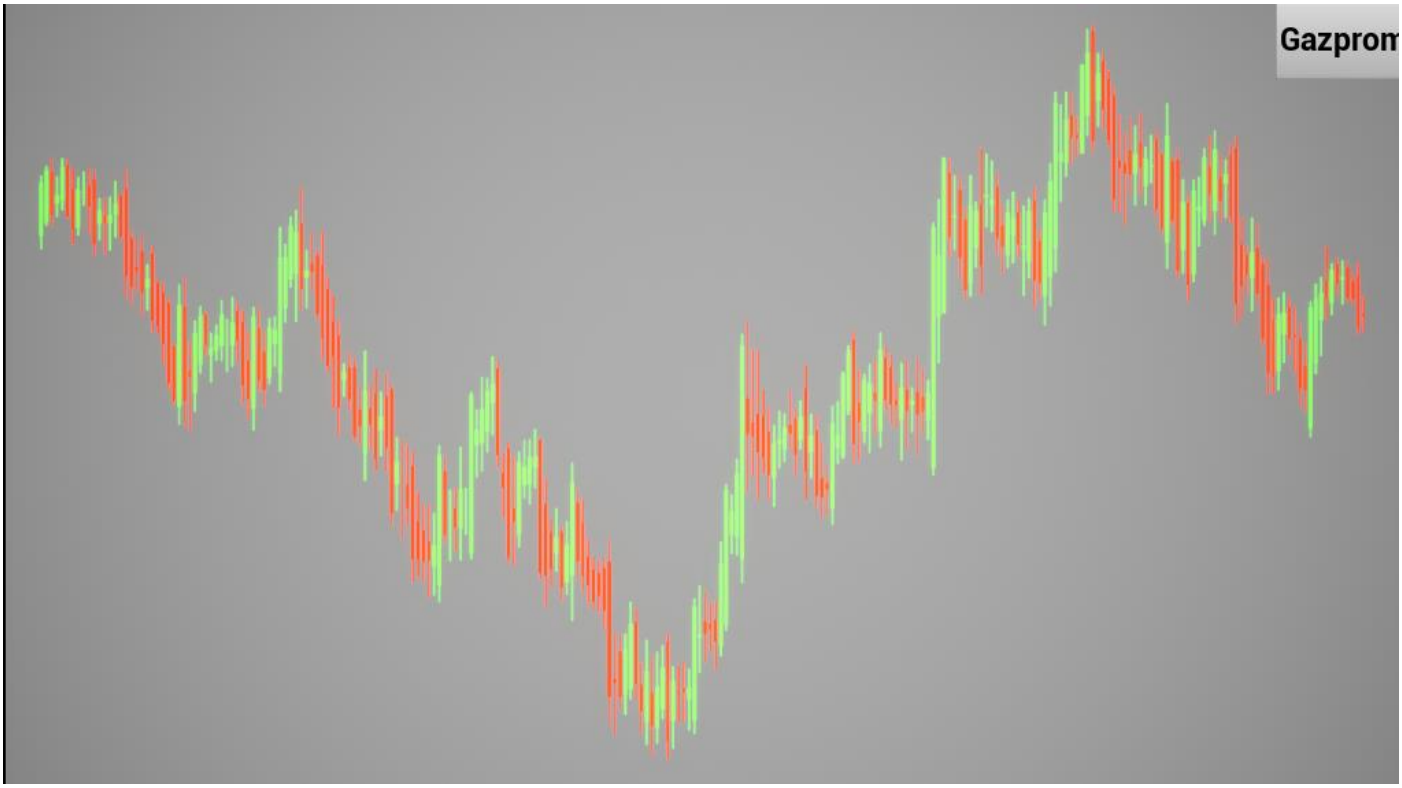

Рисунок 3. Пример использования Instance Static Mesh Component

Также технология Instance Static Mesh Component позволяет работать со Static Mesh Component-ами, входящими в множество одинаковых элементов отрисовки, по отдельности, что позволяет присваивать каждому элементу Instance Static Mesh Component свой цвет, положение на сцене, размер, масштаб, прозрачность и так далее.

Еще одно удобство технологии Instance Static Mesh Component заключается в том, что общий набор одинаковых элементов можно представить в виде массива, в котором можно добавлять, удалять и редактировать объекты по своему усмотрению. На этой возможности основана реализация смены отображаемых свечных графиков в рамках данной работы.

1.4. Базовый класс USaveGame для сохранений в Unreal Engine 4

Существует несколько способов сохранения объектов в Unreal Engine 4. Один из них - использование системы сохранения уровня, которая позволяет сохранять все объекты и настройки уровня в один файл. Другой способ - использование системы сохранения игрока, которая сохраняет только информацию о конкретном игроке, его инвентаре, положении и прогрессе в игре. Также можно использовать собственные методы сохранения, создавая собственную систему сохранения данных объектов. Каждый из этих способов имеет свои преимущества и недостатки, и выбор подходящего зависит от конкретных потребностей и целей разработки проекта.

USaveGame - это класс в Unreal Engine 4, который используется для сохранения и загрузки игровых данных. Этот класс позволяет разработчикам сохранять состояние игры, такие как прогресс игрока, настройки игры, инвентарь и многое другое, чтобы игрок мог продолжить игру с того же места, где он остановился.

USaveGame - это базовый класс, который позволяет создавать пользовательские классы для сохранения данных игры. Он предназначен для хранения информации, которая должна быть сохранена между игровыми сессиями, чтобы игрок мог вернуться к игре позже и продолжить с сохраненным прогрессом.

Структура проекта

Все файлы с исходным кодом расположены в папке «Source», файлы конфигурации – в папке «Config», основной контент проекта – в папке «Content».

Далее изображена блок-схема дерева файлов с исходным кодом (рис. 3):

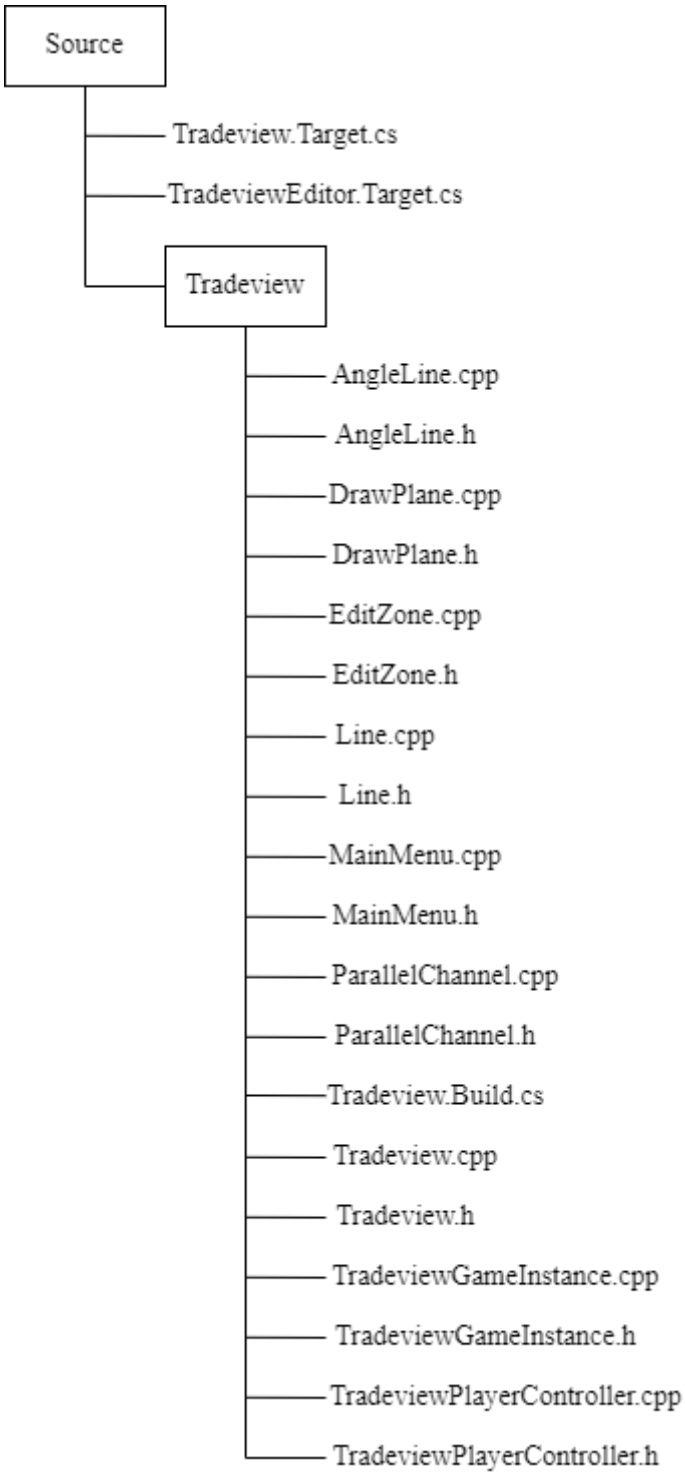

Рисунок 3. Структура папки Source

2. Описание классов

В данном проекте были реализованы следующие классы:

1. AAngleLine – базовый класс, наследуемый от класса AActor, являет собой линию с произвольным углом наклона – динию тренда.

   Листинг 1 – файл AngleLine.h

```
1. UCLASS()
2. class TRADEVIEW_API AAngleLine : public AActor
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   AAngleLine();
8.
9. public:
10.     UPROPERTY(VisibleAnywhere)
11.       UStaticMeshComponent* AngleLine;
12.     UPROPERTY(VisibleAnywhere)
13.       UStaticMeshComponent* DummyMesh;
14.     UPROPERTY(VisibleAnywhere)
15.       UStaticMeshComponent* EditZone1;
16.     UPROPERTY(VisibleAnywhere)
17.       UStaticMeshComponent* EditZone2;
18.     UPROPERTY(VisibleAnywhere)
19.       TArray<USceneComponent*> EditZoneArray;
20.
21.     FVector FirstPointAngle;
22.     FVector SecondPointAngle;
23.
24.     void CreateAngle(FTransform);
25.
26.     UFUNCTION()
27.       void ChangeCursorInTouchZone(UPrimitiveComponent*
   TouchedComponent);
28.     UFUNCTION()
29.       void ChangeCursorToDefault(UPrimitiveComponent*
   TouchedComponent);
30.
31.     virtual void Tick(float DeltaSeconds) override;
32.     virtual void BeginPlay() override;
33.   };
```

2. ALine – базовый класс, наследуемый от класса AActor, являет собой основной элемент отображения функциональных элементов – линию. Отображение элементов происходит после выбора необходимого элемента по нажатию кнопки на виджете, реализованном в классе UMainMenu, и клика в произвольном месте по объекту класса ADrawPlane, отвечающего за плоскость для размещения объектов.

   Листинг 2 – файл Line.h

```
1. UCLASS()
2. class TRADEVIEW_API ALine : public AActor
```

```
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   ALine();
8.
9. protected:
10.     virtual void BeginPlay() override;
11.
12.   public:
13.     virtual void Tick(float DeltaSeconds) override;
14.
15.     UPROPERTY(VisibleAnywhere)
16.       UStaticMeshComponent* Line;
17.     UPROPERTY(VisibleAnywhere)
18.       UStaticMeshComponent* DummyMesh;
19.
20.     UPROPERTY(VisibleAnywhere)
21.       TArray<FVector> PointPosition;
22.
23.     UPROPERTY(VisibleAnywhere)
24.       bool Angle;
25.
26.     UPROPERTY(VisibleAnywhere)
27.       FVector StartTrace;
28.
29.     UFUNCTION()
30.       void ChangeCursorInTouchZone(UPrimitiveComponent*
    TouchedComponent);
31.     UFUNCTION()
32.       void ChangeCursorToDefault(UPrimitiveComponent*
    TouchedComponent);
33.     void CreateHorizontal(FVector);
34.     void CreateVertical(FVector);
35.   };
```

3. ADrawPlane – класс, наследуемый от класса AActor, необходим для размещения на его плоскости функциональных элементов.

Листинг 3 – файл DrawPlane.h

```
1. UCLASS()
2. class TRADEVIEW_API ADrawPlane : public AActor
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   ADrawPlane();
8.
9. protected:
10.   virtual void BeginPlay() override;
11.
12.   public:
13.     virtual void Tick(float DeltaTime) override;
14.
15.     UPROPERTY(VisibleAnywhere)
16.       UStaticMeshComponent* DrawPlane;
17.   };
```

4. AEditZone – класс, наследуемый от класса AActor, являет собой зону редактирования размещенных объектов.

   Листинг 4 – файл EditZone.h

```
1. UCLASS()
2. class TRADEVIEW_API AEditZone : public AActor
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   AEditZone();
8.
9. protected:
10.     virtual void BeginPlay() override;
11.
12.   public:
13.     virtual void Tick(float DeltaTime) override;
14.
15.     UPROPERTY(VisibleAnywhere)
16.       UStaticMeshComponent* EditZone;
17.   };
```

5. UMainMenu – класс, наследуемый от класса UUserWidget, необходимый для отображения и функционирования виджетов на экране. С помощью него реализованы кнопки перехода к меню выбора размещаемого элемента (кнопка Lines) и непосредственного выбора размещаемого элемента из предложенных вариантов (предложенные варианты: Horizontal – горизонтальная линия, Vertical – вертикальная линия, Angle – “линия тренда”, или линия с произвольным углом наклона, задаваемым пользователем, Channel – параллельный канал, являющий собой три параллельные линии и зону внутри канала).

   Листинг 5 – файл UMainMenu.h

```
1. UCLASS()
2. class TRADEVIEW_API UMainMenu : public UUserWidget
3. {
4.   GENERATED_BODY()
5.
6. private:
7.
8.   UPROPERTY(meta = (BindWidget))
9.     class UButton* Lines_Button;
10.     UPROPERTY(meta = (BindWidget))
11.       class UButton* Horizontal_Button;
12.     UPROPERTY(meta = (BindWidget))
13.       class UButton* Vertical_Button;
14.     UPROPERTY(meta = (BindWidget))
15.       class UButton* Angle_Button;
16.     UPROPERTY(meta = (BindWidget))
17.       class UButton* Channel_Button;
18.
```

```
19.     UPROPERTY(meta = (BindWidget))
20.       class UWidgetSwitcher* MenuSwitch;
21.
22.     UPROPERTY(meta = (BindWidget))
23.       class UWidget* LineButtons;
24.
25.     UPROPERTY(meta = (BindWidget))
26.       class UWidget* BlankScreen;
27.
28.     UFUNCTION()
29.       void OpenLinesMenu();
30.     UFUNCTION()
31.       void HorizontalLine();
32.     UFUNCTION()
33.       void VerticalLine();
34.     UFUNCTION()
35.       void AngleLine();
36.     UFUNCTION()
37.       void ChannelLine();
38.
39.   protected:
40.     virtual bool Initialize();
41.
42.     UPROPERTY(VisibleAnywhere)
43.       class APlayerController* PlayerController;
44.
45.     UPROPERTY(VisibleAnywhere)
46.       class ATradeviewPlayerController* PC;
47.
48.     bool LMOpen = false;
49.
50.   public:
51.     TSubclassOf<UUserWidget> LinesMenuClass;
52.   };
```

6. AParallelChannel – класс, наследуемый от класса AActor, являет собой параллельный канал, ограниченный тремя параллельными линиями, которые перемещаются друг относительно друга только по вертикали, и тач-зоной, обозначающей зону покрытия канала.

Листинг 6 – файл ParallelChannel.h

```
1. UCLASS()
2. class TRADEVIEW_API AParallelChannel : public AActor
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   AParallelChannel();
8.
9. protected:
10.     virtual void BeginPlay() override;
11.
12.   public:
13.     virtual void Tick(float DeltaTime) override;
14.
15.     UPROPERTY(EditAnywhere)
16.       UMaterialInterface* DummyMaterial;
17.     UPROPERTY(EditAnywhere)
18.       UMaterialInterface* MLMaterial;
```

```
19.     UPROPERTY(EditAnywhere)
20.       UMaterialInterface* LMaterial;
21.     UPROPERTY(VisibleAnywhere)
22.       UMaterialInterface* EMaterial;
23.
24.     UPROPERTY(VisibleAnywhere)
25.       UStaticMeshComponent* Channel;
26.     UPROPERTY(VisibleAnywhere)
27.       UStaticMeshComponent* BCLine;
28.     UPROPERTY(VisibleAnywhere)
29.       UStaticMeshComponent* MCLine;
30.     UPROPERTY(VisibleAnywhere)
31.       UStaticMeshComponent* TCLine;
32.
33.     UPROPERTY(VisibleAnywhere)
34.       UStaticMeshComponent* EditZone0;
35.     UPROPERTY(VisibleAnywhere)
36.       UStaticMeshComponent* EditZone1;
37.     UPROPERTY(VisibleAnywhere)
38.       UStaticMeshComponent* EditZone2;
39.     UPROPERTY(VisibleAnywhere)
40.       UStaticMeshComponent* EditZone3;
41.     UPROPERTY(VisibleAnywhere)
42.       UStaticMeshComponent* EditZone4;
43.     UPROPERTY(VisibleAnywhere)
44.       UStaticMeshComponent* EditZone5;
45.
46.     UPROPERTY(VisibleAnywhere)
47.       UProceduralMeshComponent* ChannelDummy;
48.
49.     TArray<FVector> Vertices;
50.     TArray<int32> Triangles;
51.     TArray<FVector2D> UVs;
52.
53.     UPROPERTY(VisibleAnywhere)
54.       FVector FirstPoint;
55.     UPROPERTY(VisibleAnywhere)
56.       FVector SecondPoint;
57.     UPROPERTY(VisibleAnywhere)
58.       FVector ThirdPoint;
59.     UPROPERTY(VisibleAnywhere)
60.       FVector ForthPoint;
61.
62.     UPROPERTY(VisibleAnywhere)
63.       TArray<USceneComponent*> EditZoneArray;
64.
65.     void CreateParallelChannel(ALine*, ALine*);
66.
67.     UFUNCTION()
68.       void ChangeCursorInTouchZone(UPrimitiveComponent*
   TouchedComponent);
69.
70.     UFUNCTION()
71.       void ChangeCursorToDefault(UPrimitiveComponent*
   TouchedComponent);
72.   };
```

7. UTradeviewGameInstance – класс, наследуемый от класса UGameInstance, являющий собой необходимые настройки и положения элементов при запуске и обработке работы проекта. В нем реализовано начальное размещение и

отображение виджета класса UMainMenu, необходимого для работы выбора размещаемого элемента.

Листинг 7 – файл TradeviewGameInstance.h

```
1. UCLASS()
2. class TRADEVIEW_API UTradeviewGameInstance : public UGameInstance
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   UTradeviewGameInstance();
8.
9. public:
10.     UFUNCTION(BlueprintCallable)
11.       void LoadMainMenu();
12.
13.     TSubclassOf<UUserWidget> MenuClass;
14.   };
```

8. ATradeviewPlayerController – класс, наследуемый от класса APlayerController, являет собой основной элемент управления объектов на экране и их логикой. В нем реализовано размещение функциональных элементов в соответствии с произвольными нажатиями пользователя, их изменение и поведение во время изменения и перемещения.

Листинг 8 – файл TradeviewPlayerController.h

```
1. UCLASS()
2. class TRADEVIEW_API ATradeviewPlayerController : public
   APlayerController
3. {
4.   GENERATED_BODY()
5.
6. private:
7.   ATradeviewPlayerController();
8.
9. protected:
10.     void SetupInputComponent();
11.
12.   public:
13.     UPROPERTY(VisibleAnywhere)
14.       AAngleLine* AngleLine;
15.     UPROPERTY(VisibleAnywhere)
16.       FVector FirstPointAngle;
17.     UPROPERTY(VisibleAnywhere)
18.       FVector SecondPointAngle;
19.
20.     UPROPERTY(VisibleAnywhere)
21.       AParallelChannel* Channel;
22.     UPROPERTY(VisibleAnywhere)
23.       ALine* BottomChannelLine;
24.     UPROPERTY(VisibleAnywhere)
25.       ALine* TopChannelLine;
26.     UPROPERTY(VisibleAnywhere)
27.       FVector FirstPointChannel;
28.     UPROPERTY(VisibleAnywhere)
29.       FVector SecondPointChannel;
```

```
30.     UPROPERTY(VisibleAnywhere)
31.       FVector ThirdPointChannel;
32.
33.     UPROPERTY(VisibleAnywhere)
34.       FTransform LineTransform;
35.
36.     UPROPERTY(VisibleAnywhere)
37.       UPrimitiveComponent* GrabbedComponent;
38.     UPROPERTY(VisibleAnywhere)
39.       FVector RelativeGrabLocation;
40.     UPROPERTY(VisibleAnywhere)
41.       float OriginalDistance;
42.     UPROPERTY(VisibleAnywhere)
43.       FVector AncorPoint;
44.     UPROPERTY(VisibleAnywhere)
45.       TArray<USceneComponent*> EditZoneArray;
46.
47.     UPROPERTY(VisibleAnywhere)
48.       float Height;
49.     UPROPERTY(VisibleAnywhere)
50.       bool LineEditing = false;
51.     UPROPERTY(VisibleAnywhere)
52.       bool ChannelEditing = false;
53.     UPROPERTY(VisibleAnywhere)
54.       int ChannelLine = 0;
55.     UPROPERTY(VisibleAnywhere)
56.       bool HeightEditing = false;
57.
58.
59.
60.     virtual void Tick(float DeltaSeconds) override;
61.
62.     void OnMouseClick();
63.     void OnMouseRelease();
64.
65.     void SetHorizontal();
66.     void SetVertical();
67.     void SetAngle();
68.     void SetChannel();
69.
70.     bool GrabMode = true;
71.     bool Horizontal = false;
72.     bool Vertical = false;
73.     bool Angle = false;
74.     bool PlaceAngle = false;
75.     bool ParallelChannel = false;
76.     bool PlaceParallelChannel = false;
77.     bool PlaceParallelTop = false;
78.   };
```

3. Реализация размещения и поведения классов

В меню выбора размещаемых элементов есть 4 опции:

Кнопка Horizontal – отвечает за задание логики размещения горизонтальной линии без возможности ее дальнейшего редактирования, но оставляя возможность ее перемещения по плоскости отображения объектов. При нажатии на кнопку Horizontal запускается логика задания размеров будущей горизонтальной линии и программа переключается в режим ожидания нажатия пользователем в произвольном месте на плоскости отображения. После регистрации нажатия программа получает координаты нажатия на плоскость, после чего создает объект класса Aline, задает ему координаты размещения на плоскости для размещения объектов, затем создает компонент DummyMeshComponent, реализующий тач-зону размещенной горизонтальной линии и присоединяет этот объект к уже размещенной горизонтальной линии. Далее размещенный объект имеет возможность быть перемещенным по вертикали посредством нажатия на его тач-зону и перетягивания его в произвольную точку на рабочей плоскости.

Кнопка Vertical – отвечает за задание логики размещения вертикальной линии без возможности ее дальнейшего редактирования, но оставляя возможность ее перемещения по плоскости отображения объектов. При нажатии на кнопку Vertical запускается логика задания размеров будущей вертикальной линии и программа переключается в режим ожидания нажатия пользователем в произвольном месте на плоскости отображения. После регистрации нажатия программа получает координаты нажатия на плоскость, после чего создает объект класса Aline, задает ему координаты размещения на плоскости для размещения объектов, затем создает компонент DummyMeshComponent, реализующий тач-зону размещенной вертикальной линии и присоединяет этот объект к уже размещенной вертикальной линии. Далее размещенный объект имеет возможность быть перемещенным по горизонтали посредством нажатия на его тач-зону и перетягивания его в произвольную точку на рабочей плоскости.

Кнопка Angle – отвечает за задание логики размещения линии с произвольным углом наклона с возможностью ее дальнейшего перемещения и редактирования. После нажатия кнопки Angle программа ожидает, пока пользователь нажмет в произвольной точке рабочей плоскости, после чего происходит определение координат точки касания

и эти координаты записываются в поле первой точки построения линии (FirstPointAngle). Далее программа на каждый тик считывает положение курсора указателя мыши пользователя и обновляет положение линии так, чтобы концами этой линии являлись уже известные координаты первой точки и проекция положения курсора указателя мыши на рабочую плоскость. Когда пользователь хочет завершить построение линии, ему необходимо еще раз нажать левую кнопку мыши в произвольном месте на рабочей плоскости для считывания координат положения курсора указателя мыши и записи соответствующих координат в поле второй точки построения линии (SecondPointAngle). Построение линии с произвольным углом наклона завершается и строится линия, концы которой имеют координаты, записанные в поля первой и второй точек построения линии, после чего создаются вспомогательные элементы: компонент DummyMeshComponent, являющий собой тач-зону построенной линии с произвольным углом наклона, который после создания присоединяется к уже построенной линии; Два объекта класса AEditZone, являющих собой визуальные поля, обозначающие поля редактирования построенной линии с произвольным углом наклона, которые располагаются на концах построенной линии.

Перемещение линии с произвольным углом наклона осуществляется посредством наведения курсора указателя мыши на тач-зону этой линии, нажатия левой кнопкой мыши по тач-зоне и перетягивания ее в произвольную точку на рабочей плоскости.

Редактирование линии с произвольным углом наклона осуществляется посредством наведения курсора указателя мыши на тач-зону редактируемой линии, после которого на концах линии станут видны поля для редактирования. Для редактирования необходимо переместить курсор в область поля для редактирования, зажать левую кнопку мыши и перетащить поле для редактирования в необходимую пользователю точку.

При редактировании после нажатия левой кнопки мыши в области поля для редактирования линия перестраивается следующим образом:

После нажатия левой кнопкой мыши по одному из двух полей для редактирования координаты противоположного поля для редактирования (то есть соответствующие координаты конца линии) записываются в поле якорной точки (AncorPoint) и остаются неизменными до конца редактирования.

Линия перестраивается относительно двух точек: якорной точки и положения курсора указателя мыши пользователя с необходимыми изменениями положения и масштаба.

После того, как пользователь отпустит левую кнопку мыши, линия выйдет из режима редактирования и будет построена по двум точкам: точке с координатами якорной точки и точки, в которой была отпущена левая кнопка мыши. Положение вспомогательных объектов (тач-зоны и полей для редактирования) изменится соответственно изменениям линии с произвольным углом наклона.

Кнопка Channel – отвечает за задание логики размещения объекта класса AParallelChannel, являющим собой параллельный канал, с возможностью его дальнейшего перемещения и редактирования. После нажатия кнопки Channel программа ожидает, пока пользователь нажмет в произвольной точке рабочей плоскости, после чего происходит определение координат точки касания и эти координаты записываются в поле первой точки построения параллельного канала (FirstPointChannel). Далее программа на каждый тик считывает положение курсора указателя мыши пользователя и обновляет положение первой линии канала так, чтобы концами этой линии являлись уже известные координаты первой точки и проекция положения курсора указателя мыши на рабочую плоскость. Когда пользователь хочет завершить построение первой линии канала, ему необходимо еще раз нажать левую кнопку мыши в произвольном месте на рабочей плоскости для считывания координат положения курсора указателя мыши и записи соответствующих координат в поле второй точки построения первой линии канала (SecondPointChannel). Построение первой линии канала завершается и строится линия, концы которой имеют координаты, записанные в поля первой и второй точек построения канала, после чего создается вторая линия канала, имеющая идентичные вектора положения, вращения и масштаба. Вторая линия канала перемещается строго по вертикали относительно первой линии канала, для чего необходимо было ввести следующие ограничения перемещения второй линии канала:

При перемещении курсора указателя мыши по вертикали вверх и вниз вторая линия канала перемещается на такое же расстояние вверх и вниз соответственно.

При перемещении курсора указателя мыши по горизонтали вправо и влево вторая линия канала перемещается так, чтобы прямая, содержащая вторую линию канала, следовала за курсором указателя мыши.

Координаты положения второй линии канала определяются из отношения разницы координат концов первой линии канала по вертикали и разницы координат концов первой линии канала по горизонтали, умноженного на перемещение курсора указателя мыши по горизонтали.

После нажатия левой кнопки мыши устанавливается положение второй линии канала, после чего строится параллельный канала следующим образом:

В функцию построения параллельного канала передаются обе линии параллельного канала.

Определяются ключевые (крайние) точки параллельного канала: первая и вторая точки параллельного канала, расположенные на концах первой линии канала, определяются из ранее полученных при построении первой линии канала точек и записываются в соответствующие поля; Третья и четвертая точки параллельного канала определяются как сумма второй точки параллельного канала и разницы координат по вертикали линий параллельного канала и сумма первой точки параллельного канала и разницы координат по вертикали линий параллельного канала соответственно.

Строится зона покрытия канала, она же тач-зона канала – процедурно генерируемый меш из четырех треугольных секций, которые задаются с учетом расположения нормалей секций процедурно генерируемого меша: две секции – лицевая сторона – должны иметь нормали, направленные в сторону пользователя, оставшиеся две – тыльная сторона процедурно генерируемого меша – соответственно, в противоположную сторону, от пользователя. Это достигается путем грамотного задания топологии построения треугольных секций процедурно генерируемого меша – по ”правилу правой руки”, то есть при обходе вершин так, как они были расположены при задании топологии процедурно генерируемого меша, необходимо направить пальцы правой руки через вершины треугольной секции и большой палец правой руки, отведенный под 90 градусов, будет показывать направление нормали треугольной секции процедурно генерируемого меша. Это продемонстрировано цифрами красного и синего цвета (рис. 4).

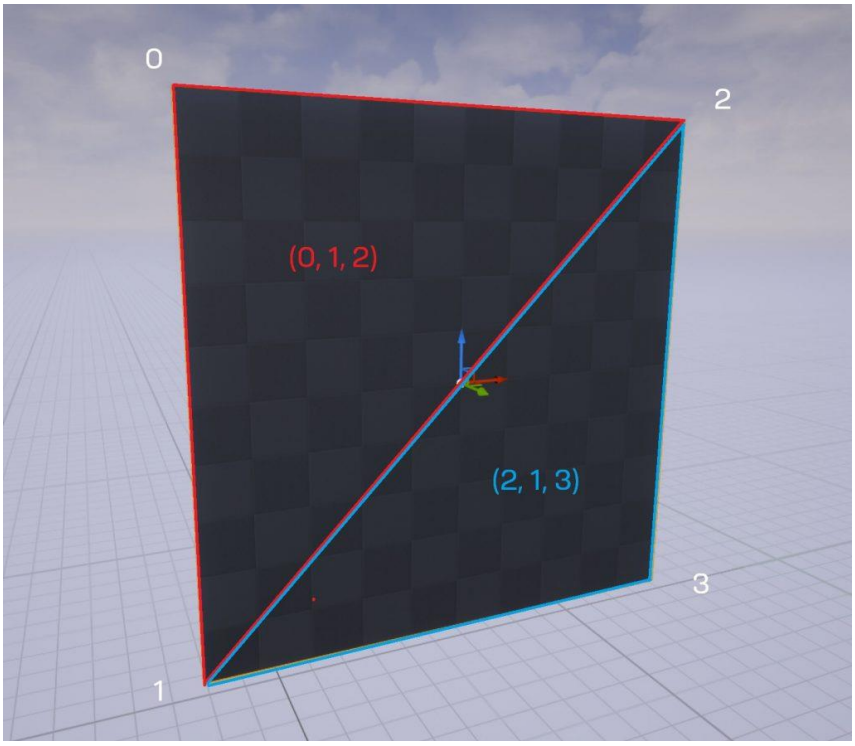

Рисунок 4. Пример построения процедурно генерируемого меша [6]

Создаются и добавляются к параллельному каналу 6 объектов класса AEditZone, являющих собой области полей для редактирования параллельного канала. Они располагаются соответственно концам обеих линий, образующих параллельный канал, по 2 области на каждую линию, и по одной области полей редактирования на центр каждой линии параллельного канала.

После построения параллельного канала пользователь может перемещать и редактировать параллельный канал по своему усмотрению.

Перемещение параллельного канала осуществляется посредством наведения курсора указателя мыши на тач-зону параллельного канала, нажатия левой кнопкой мыши по тач-зоне и перетягивания ее в произвольную точку на рабочей плоскости.

Редактирование параллельного канала осуществляется посредством наведения курсора указателя мыши на тач-зону редактируемого параллельного канала, после которого станут видны все поля для редактирования. Для редактирования необходимо переместить курсор в область поля для редактирования, зажать левую кнопку мыши и перетащить поле для редактирования в необходимую пользователю точку.

При редактировании после нажатия левой кнопки мыши в области поля для редактирования параллельный канал перестраивается следующими образами:

Если курсор указателя мыши находится в области одного из полей редактирования, расположенных на концах линий параллельного канала, то по нажатию левой кнопки мыши в области поля редактирования на одной из линий параллельного канала противоположный конец этой линии становится “якорной точкой” и остается неизменной до конца редактирования.

Линия параллельного канала перестраивается относительно двух точек: якорной точки и положения курсора указателя мыши пользователя с необходимыми изменениями положения и масштаба.

Вторая линия параллельного канала повторяет изменения и перемещения редактируемой линии параллельного канала, но сохраняет параллельность и разницу между линиями параллельного канала по вертикали.

Соответственно перестраивается и тач-зона параллельного канала, сохраняя свою топологию.

Если курсор указателя мыши находится в области одного из полей редактирования, расположенных на центрах линий параллельного канала, то по нажатию левой кнопки мыши в области поля редактирования на одной из линий параллельного канала пользователь может переместить редактируемую линию вертикально вверх или вниз, по следующим принципам:

При перемещении курсора указателя мыши по вертикали вверх и вниз редактируемая линия канала перемещается на такое же расстояние вверх и вниз соответственно.

При перемещении курсора указателя мыши по горизонтали вправо и влево редактируемая линия канала перемещается так, чтобы прямая, содержащая редактируемую линию канала, следовала за курсором указателя мыши.

Координаты положения редактируемой линии канала определяются из отношения разницы координат концов линии канала по вертикали и разницы координат концов линии канала по горизонтали, умноженного на перемещение курсора указателя мыши по горизонтали.

Тач-зона параллельного канала перестраивается соответствующе изменениям параллельного канала.

После отпускания левой кнопки мыши программа выйдет из режима редактирования.

4. Реализация размещения и отрисовки свечных графиков

В ходе данной работы необходимо реализовать грамотное отображение и размещение финансовых свечных графиков с использованием интерактивных элементов и средств отображения игрового движка Unreal Engine 4.

В первую очередь необходимо было подобрать и отформатировать данные по финансовому состоянию акций некоторых крупных компаний, в данном случае APPLE, Камаз и ГазПром. Для этого были скачаны из общего доступа графики акций вышеперечисленных компаний и переведены в формат CSV – формат файла, в котором данные разделяются запятой либо точкой с запятой. Данные отформатированные файлы необходимо было загрузить в Unreal Engine 4, для чего были созданы соответствующие таблицы для размещения там информации по графикам в понятной и легкодоступной форме, а также специальная структура, благодаря которой CSV-файлы были успешно разделены на блоки данных и размещены в соответствующие строки таблиц.

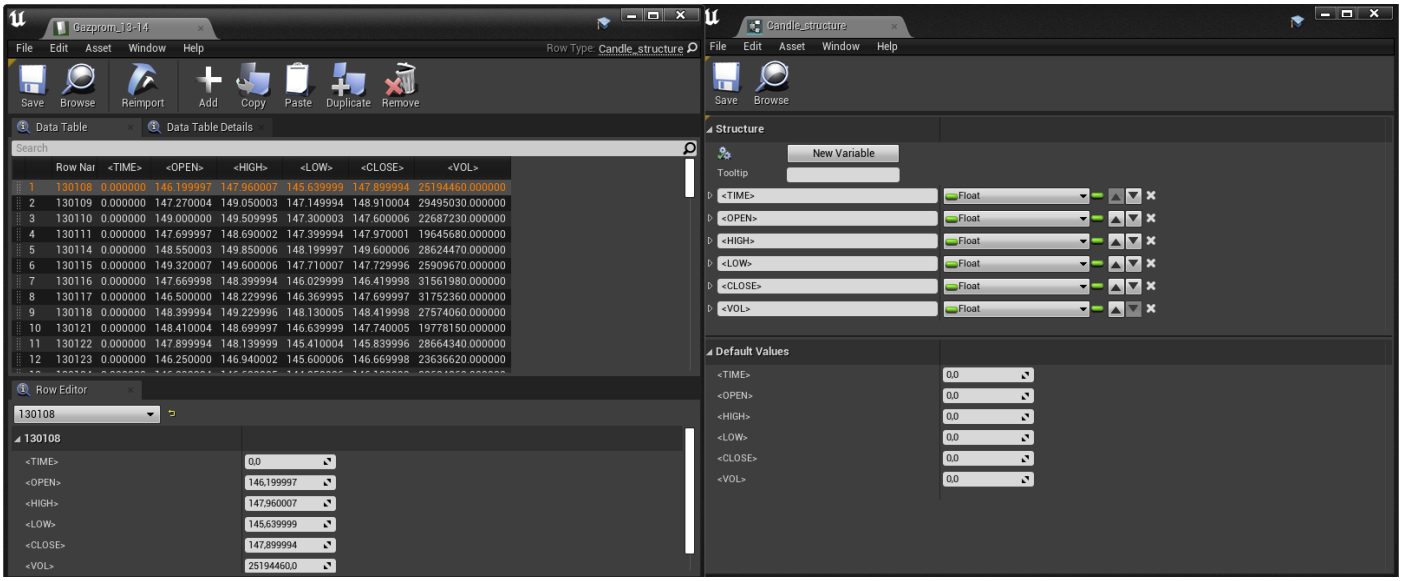

Рисунок 5. Пример размещения данных в таблице (слева) и специальная структура для размещения данных (справа)

Для отрисовки данных из таблицы на сцене необходимо было реализовать следующее:

1. Выбор необходимой таблицы в зависимости от того, какой график выберет пользователь в выпадающем меню выбора графика. Реализовано данное решение было с использованием интерактивных виджетов, в частности с

использованием ComboBox, являющим собой настраиваемый выпадающий список предложенных элементов.

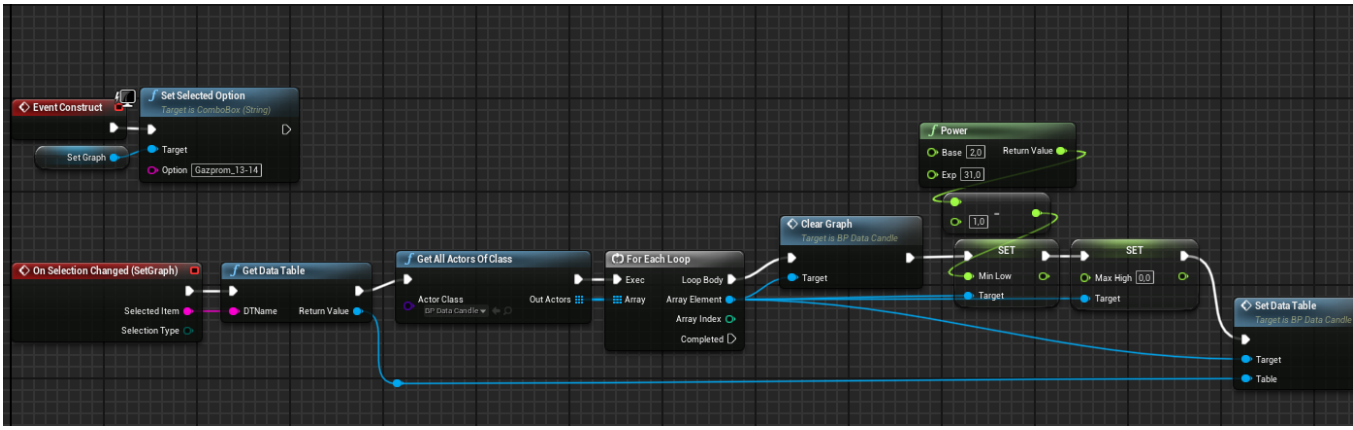

Рисунок 6. Графическое представление алгоритма действий при выборе пользователем отображаемого графика

Данный алгоритм реализует следующее:

1. При создании виджета (то есть при запуске программы) автоматически выбирается предложенный вариант – в данном случае будет отрисован график акций ГазПром.
2. При смене пользователем желаемого отображаемого графика запускается событие смены графика (On Selected Change (SetGraph)), в котором происходит чтение из элемента виджета ComboBox значения, которое было в нем установлено пользователем, на основании полученного значения по имени таблицы выбирается таблица, из которой будет происходить чтение данных, после чего сама таблица передается на вход функции SetDataTable, которая, в свою очередь, была разработана для отрисовки свечного графика из таблицы, поступающей на вход как аргумент функции. До вызова функции SetDataTable алгоритм вызывает функцию, подготавливающую сцену для отрисовки нового свечного графика – функцию ClearGraph, которая очищает массив Instance Static Mesh Component, тем самым удаляя более не использующиеся данные со сцены, а также очищая саму сцену. Также в функцию SetDataTable передаются константы, необходимые для просчета значений для дальнейшего определения масштаба и размещения свечей графика.

3. Чтение данных из таблицы и заполнение необходимых для построения констант – общая высота свечей, необходимая для того, чтобы масштаб свечей по вертикали соответствовал соотношению масштабов самих свечей; Ширина свечей, необходимая для правильного отображения свечей как отдельных друг от друга элементов, с учетом отступов и количества самих свечей. Для этого была было разработано следующее:
    1. При вызове функции SetDataTable происходит чтение полученной как аргумент функции таблицы с данными о свечном графике, затем каждая строка разбивается на отдельные значения переменных в соответствии со структурой строки таблицы данных, после чего происходит вызов функции просчета горизонтального масштаба свечей графика и их размещения (SetCandlePositions).

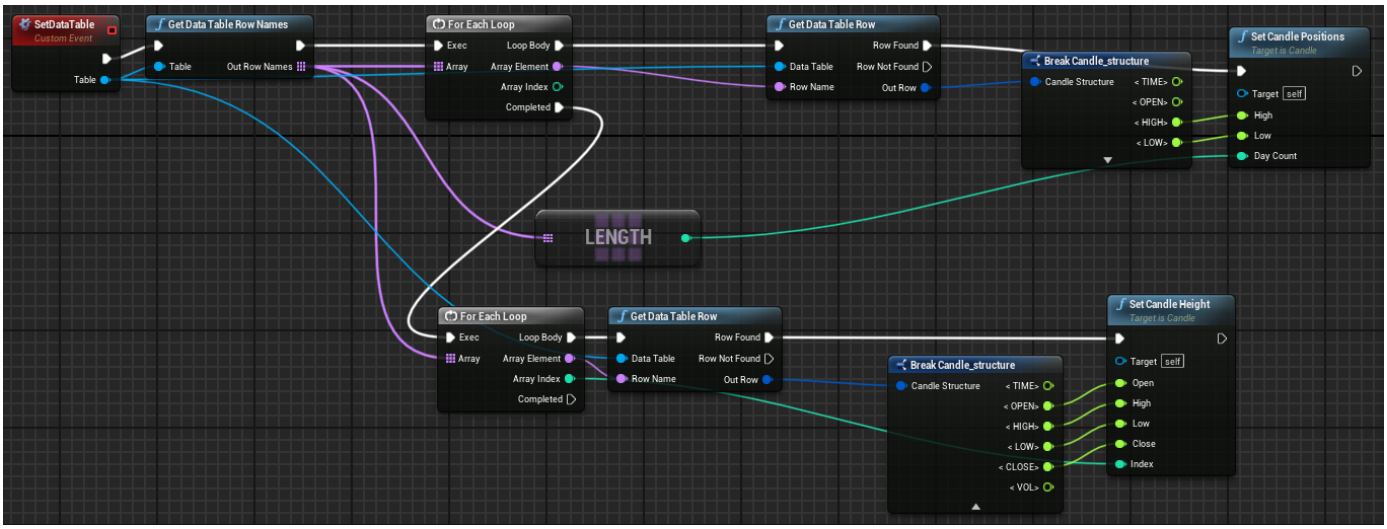

Рисунок 7. Графическое представление функции SetDataTable

2. При вызове функции SetCandlePositions происходит следующее – функция принимает как аргументы крайние точки теней свечей и количество свечей, затем происходит расчет констант, необходимых для соблюдения масштаба свечей и их взаимного расположения, после чего свечи размещаются на одной высоте с соблюдением своих масштабов и отступов свечей друг от друга.

Листинг 9 – файл Candle.cpp – функция SetCandlePositions

```
1. void ACandle::SetCandlePositions(float _high, float _low, int _dayCount)
2. {
3.     //time = _time;
4.     //open = _open;
5.     high = _high;
6.     low = _low;
7.     //close = _close;
8.     //volume = _volume;
9.     dayCount = _dayCount;
10.
11.    if (high > maxHigh)
12.    {
13.        maxHigh = high;
14.    }
15.    if (low < minLow)
16.    {
17.        minLow = low;
18.    }
19.
20.    // candle placement here
21.    candleWidth = 100 * 0.8f / dayCount;
22.    FVector StartPos = FVector(-100*0.4f/2.f, -100*0.8f/2.f, 0.f);
23.    FTransform newCandle = FTransform(FRotator(0.f, 0.f, 0.f),
   FVector(StartPos.X + 0.f, StartPos.Y + CandleBody->GetInstanceCount() *
   candleWidth + candleWidth/2.f, 0.03f), FVector(candleWidth/100.f,
   candleWidth/100.f - 0.001f, 1.f));
24.    FTransform newCandleLine = FTransform(newCandle.GetRotation(),
   newCandle.GetLocation() + /*FVector(-20.f, -50.f, 0.f)*/FVector(0.f, 0.f,
   0.f), FVector(candleWidth / 100.f, candleWidth / 200.f - 0.0005f, 1.f));
25.
26.    CandleBody->AddInstance(newCandle);
27.    CandleLine->AddInstance(newCandleLine);
28. }
```

После завершения цикла горизонтального размещения свечей графика для каждой строки таблицы с данными о графике вызывается функция SetCandleHeight, отвечающая за размещение свечей во вертикали с соблюдением масштабов сцены и взаимного расположения и масштаба свечей графика.

Листинг 10 – файл Candle.cpp – функция SetCandleHeight

```
1. void ACandle::SetCandleHeight(float _open, float _high, float _low, float
   _close, int index)
2. {
3. // counting candle body's placement
4. float XCandleCenter = (_open + _close) / 2.f;

5. float XCandleDataPercent = (XCandleCenter - minLow) * 100.f / (maxHigh -
   minLow);
6. float XCandleCoordinates = (100.f * 0.4f * XCandleDataPercent) / 100.f;

7. float XCandleDataScale;
8. if (_open > _close)
9. {
10.XCandleDataScale = (_open - _close) * 100.f / (maxHigh - minLow);

11.CandleBody->SetCustomDataValue(index, 0, 1.f);
```

```
12.CandleBody->SetCustomDataValue(index, 1, 0.f);

13.CandleLine->SetCustomDataValue(index, 0, 1.f);
14.CandleLine->SetCustomDataValue(index, 1, 0.f);
15.}
16.else
17.{
18.XCandleDataScale = (_close - _open) * 100.f / (maxHigh - minLow);

19.CandleBody->SetCustomDataValue(index, 0, 0.f);
20.CandleBody->SetCustomDataValue(index, 1, 1.f);

21.CandleLine->SetCustomDataValue(index, 0, 0.f);
22.CandleLine->SetCustomDataValue(index, 1, 1.f);
23.}
24.float XCandleCoordinatesScale = ((100.f * 0.4f * XCandleDataScale) /
   100.f) / (100.f * 0.4f);

25.// counting candle line's placement
26.float XCandleLineCenter = (_high + _low) / 2.f;

27.float XCandleLineDataPercent = (XCandleLineCenter - minLow) * 100.f /
   (maxHigh - minLow);
28.float XCandleLineCoordinates = (100.f * 0.4f * XCandleLineDataPercent) /
   100.f;

29.float XCandleLineDataScale = (_high - _low) * 100.f / (maxHigh - minLow);
30.float XCandleLineCoordinatesScale = ((100.f * 0.4f *
   XCandleLineDataScale) / 100.f) / (100.f * 0.4f);

31.// setting everything in their places
32.FTransform oldBodyTransform;
33.CandleBody->GetInstanceTransform(index, oldBodyTransform, false);
34.CandleBody->UpdateInstanceTransform(index,
   FTransform(oldBodyTransform.GetRotation(), oldBodyTransform.GetLocation()
   + FVector(XCandleCoordinates, 0.f, 0.f), FVector(XCandleCoordinatesScale,
   oldBodyTransform.GetScale3D().Y, oldBodyTransform.GetScale3D().Z)));

35.FTransform oldLineTransform;
36.CandleLine->GetInstanceTransform(index, oldLineTransform, false);
37.CandleLine->UpdateInstanceTransform(index, FTransform(FRotator(0.f, 0.f,
   0.f), oldLineTransform.GetLocation() + FVector(XCandleLineCoordinates,
   0.f, 0.f), FVector(XCandleLineCoordinatesScale,
   oldLineTransform.GetScale3D().Y, oldLineTransform.GetScale3D().Z)));
38.}
```

В данной функции происходит определение цвета свечи посредством сравнения значений точек открытия и закрытия торгов – если значение точки открытия торгов за определенный промежуток времени было меньше значения точки закрытия торгов, то такая свеча будет иметь зеленый цвет, что является символом положительной тенденции, в противном случае свеча будет окрашена в красный цвет. Далее происходит расчет вертикального положения каждой свечи, для чего необходимо рассчитать процентное замощение тела и теней свечи пространства необходимого графика, затем определяется общее расстояние сцены по вертикали для

того, чтобы через пропорцию преобразовать процентное замощение пространства свечой в ее величину в координатах Unreal Engine. Подобным образом вычисляется масштаб тела свечи, длина и масштаб теней свечи и их взаимное положение на сцене.

4. Отрисовка свечей с соблюдением их взаимного размещения, масштаба, расчеты высоты теней свечей и тел свечей, присвоение свечам определенного цвета в зависимости от данных, полученных из таблицы. Это происходит посредством использования технологии Instance Static Mesh Component, заполнение которого идет подобно массиву – в данном случае свеча, заранее помещенная в массив Instance Static Mesh Component (CandleBody предназначен для тел свечей, CandleLine – для теней свечей, соответственно), может быть изменена посредством обращения к ней по ее индексу в структуре Instance Static Mesh Component и непосредственному внесению изменений через специальные команды. В ходе выполнения функций SetCandlePositions и SetCandleHeight поэлементно вносятся изменения в каждое тело и тени свечи графика, вследствие чего все представленные в структуре массива элементов Instance Static Mesh Component свечи будут отрисованы за меньшее количество DrawCall-ов, являя собой структуру идентичных друг другу объектов, но, в то же время, каждая свеча будет самостоятельной единицей представления графической информации о финансовом графике.

5. Реализация системы сохранений отрисованных элементов

В ходе данной работы был разработан специальный класс UMySaveGame, наследуемый от базового класса USaveGame, необходимого для сохранения состояния игры, объектов и информации об объектах на сцене.

Для сохранения данных с помощью USaveGame можно использовать функции SaveGameToSlot и LoadGameFromSlot, которые позволяют сохранить данные в файл и загрузить их обратно из файла. Также можно использовать другие методы, такие как SaveGameToMemory и LoadGameFromMemory, для сохранения данных в память и загрузки их оттуда.

Для того, чтобы сохранить функциональные элементы на сцене, был разработан следующий функционал класса UMySaveGame:

1. Структура FSaveLine – необходима для сохранения горизонтальных и вертикальных линий, содержит в себе поля Position – положение линии на сцене, и Horizontal – булевая переменная, определяющая какой будет размещенная линия – горизонтальной и вертикальной. Для размещения линий на сцене используются уже разработанные методы класса ALine – CreateHorizontal и CreateVertical.
2. Структура FSaveAngleLine – необходима для сохранения линий тренда – линий с произвольным углом наклона, - содержит в себе поля FirstPoint – положение первой точки линии тренда (одного конца отрезка), SecondPoint – положение второй точки линии тренда (другого конца отрезка), LineTransform – структура, содержащая вектор вращения, вектор положения и вектор масштаба объекта. Для размещения линий тренда на сцене используется уже разработанный метод класса AAngleLine – CreateAngle.
3. Структура FSaveChannel – необходима для сохранения параллельных каналов, содержит в себе поля FirstPoint – положение первой точки параллельного канала, SecondPoint – положение второй точки параллельного канала, ThirdPoint – положение третьей точки параллельного канала, ForthPoint – положение четвертой точки параллельного канала, BotLine – объект класса ALine, необходимый для сохранения нижней линии параллельного канала,

TopLine – объект класса ALine, необходимый для сохранения верхней линии параллельного канала. Для размещения параллельного канала на сцене используется уже разработанный метод класса AParallelChannel – CreateChannel.

Листинг 11 – файл MySaveGame.h

```
1. USTRUCT()
2. struct FSaveLine
3. {
4. GENERATED_BODY()
5.
6. public:
7. UPROPERTY(EditAnywhere)
8. FVector Position;
9.
10.    UPROPERTY(EditAnywhere)
11.    bool Horizontal;
12.    };
13.
14.    USTRUCT()
15.    struct FSaveAngleLine
16.    {
17.    GENERATED_BODY()
18.
19.    public:
20.    UPROPERTY(EditAnywhere)
21.    FVector FirstPoint;
22.
23.    UPROPERTY(EditAnywhere)
24.    FVector SecondPoint;
25.
26.    UPROPERTY(EditAnywhere)
27.    FTransform LineTransform;
28.    };
29.
30.    USTRUCT()
31.    struct FSaveChannel
32.    {
33.    GENERATED_BODY()
34.
35.    public:
36.    UPROPERTY(EditAnywhere)
37.    FVector FirstPoint;
38.    UPROPERTY(EditAnywhere)
39.    FVector SecondPoint;
40.    UPROPERTY(EditAnywhere)
41.    FVector ThirdPoint;
42.    UPROPERTY(EditAnywhere)
43.    FVector ForthPoint;
44.
45.    UPROPERTY(EditAnywhere)
46.    ALine* BotLine;
47.    UPROPERTY(EditAnywhere)
48.    ALine* TopLine;
49.    };
50.
51.    UCLASS()
52.    class TRADEVIEW_API UMySaveGame : public USaveGame
```

```
53.    {
54.    GENERATED_BODY()
55.
56.    public:
57.    UPROPERTY(EditAnywhere)
58.    TArray<FSaveLine> SaveLines;
59.
60.    UPROPERTY(EditAnywhere)
61.    TArray<FSaveAngleLine> SaveAngleLines;
62.
63.    UPROPERTY(EditAnywhere)
64.    TArray<FSaveChannel> SaveChannel;
65.
66.    UMySaveGame();
67.    };
```

Сохранение объектов на сцене происходит следующим образом:

1. Создается объект класса UMySaveGame, благодаря которому осуществимо сохранение расположенных на сцене объектов.
2. Проводится поиск по объектам класса ALine, который описывает горизонтальные и вертикальные линии.
3. Создается подходящая структура, входящая в состав объекта класса UMySaveGame, в поля которой поочередно записываются все необходимые для сохранения и дальнейшей загрузки данные.
4. Полученная структура сохраняется в поле объекта класса UMySaveGame.
5. п.п 2-4 повторяются для следующих элементов: линий тренда (класс AnglrLine) и параллельных каналов (класс AParallelChannel).
6. Полученный на выходе объект класса UMySaveGame сохраняется в специальный слот, отведенный автоматически Unreal Engine 4, и создается файл, описывающий сохраненный элементы.

Листинг 12 – файл TradeviewPlayerController.cpp – функция SaveGame

```
1. void ATradeviewPlayerController::SaveGame(FString FileName)
2. {
3. int Index;
4. if (FileName == "Gazprom_13-14") Index = 0;
5. else if (FileName == "KMAZ_13-14") Index = 1;
6. else Index = 2;
7.
8. GI = GetGameInstance();
9. UMySaveGame* SavedData =
   Cast<UMySaveGame>(UGameplayStatics::CreateSaveGameObject(UMySaveGame::Static
   Class()));
```

```
10.    TArray<AActor*> Out;
11.
12.    UGameplayStatics::GetAllActorsOfClass(GetWorld(), ALine::StaticClass(),
       Out);
13.    for (AActor* FActor : Out)
14.    {
15.    FSaveLine FSL;
16.    FSL.Position = FActor->GetActorLocation();
17.    FSL.Horizontal = Cast<ALine>(FActor)->Horizontal;
18.    SavedData->SaveLines.Add(FSL);
19.    }
20.    Out.Empty();
21.
22.    UGameplayStatics::GetAllActorsOfClass(GetWorld(),
       AAngleLine::StaticClass(), Out);
23.    for (AActor* FActor : Out)
24.    {
25.    FSaveAngleLine FSAL;
26.    FSAL.FirstPoint = Cast<AAngleLine>(FActor)->FirstPointAngle;
27.    FSAL.SecondPoint = Cast<AAngleLine>(FActor)->SecondPointAngle;
28.    FSAL.LineTransform = FActor->GetActorTransform();
29.    SavedData->SaveAngleLines.Add(FSAL);
30.    }
31.    Out.Empty();
32.
33.    UGameplayStatics::GetAllActorsOfClass(GetWorld(),
       AParallelChannel::StaticClass(), Out);
34.    for (AActor* FActor : Out)
35.    {
36.    FSaveChannel FSC;
37.    FSC.FirstPoint = Cast<AParallelChannel>(FActor)->FirstPoint;
38.    FSC.SecondPoint = Cast<AParallelChannel>(FActor)->SecondPoint;
39.    FSC.ThirdPoint = Cast<AParallelChannel>(FActor)->ThirdPoint;
40.    FSC.ForthPoint = Cast<AParallelChannel>(FActor)->ForthPoint;
41.    FSC.BotLine = Cast<AParallelChannel>(FActor)->BCLineSave;
42.    FSC.TopLine = Cast<AParallelChannel>(FActor)->TCLineSave;
43.    SavedData->SaveChannel.Add(FSC);
44.    }
45.    Out.Empty();

46.    UGameplayStatics::SaveGameToSlot(SavedData, FileName, 0 /*Index*/);
47.}
```

Перед загрузкой и размещением сохраненных объектов на сцену производится очистка сцены.

Листинг 13 – файл TradeviewPlayerController.cpp – функция ClearActors

```
1. void ATradeviewPlayerController::ClearActors()
2. {
3. TArray<AActor*> Out;
4.
5. UGameplayStatics::GetAllActorsOfClass(GetWorld(),
   ALine::StaticClass(), Out);
6. for (AActor* ActorFound : Out) { ActorFound->Destroy(); }
```

```
7. Out.Empty();
8. UGameplayStatics::GetAllActorsOfClass(GetWorld(),
   AAngleLine::StaticClass(), Out);
9. for (AActor* ActorFound : Out) { ActorFound->Destroy(); }
10.    Out.Empty();
11.
12.    UGameplayStatics::GetAllActorsOfClass(GetWorld(),
   AParallelChannel::StaticClass(), Out);
13.    for (AActor* ActorFound : Out) { ActorFound->Destroy(); }
14.    Out.Empty();
15.}
```

Загрузка и размещение объектов на сцене происходит следующим образом:

1. Создается объект класса UMySaveGame, необходимый для загрузки и расшифровки сохраненных ранее данных.
2. В полученном объекте класса поочередно расшифровываются структуры данных, содержащие сохраненные элементы.
3. В зависимости от того, с какой структурой данных проводится работа в данный момент, вызываются соответствующие функции отрисовки функциональных элементов.

Листинг 14 – файл TradeviewPlayerController.cpp – функция LoadGame

```
1. void ATradeviewPlayerController::LoadGame(FString FileName)
2. {
3. int Index;
4. if (FileName == "Gazprom_13-14") Index = 0;
5. else if (FileName == "KMAZ_13-14") Index = 1;
6. else Index = 2;
7.
8. if (UGameplayStatics::DoesSaveGameExist(FileName, 0 /*Index*/))
9. {
10.    GI = GetGameInstance();
11.    UMySaveGame* LoadedData =
   Cast<UMySaveGame>(UGameplayStatics::LoadGameFromSlot(FileName, 0
   /*Index*/));
12.
13.    if (LoadedData->SaveLines.Num() > 0)
14.    for (FSaveLine ActorToSpawn : LoadedData->SaveLines)
15.    {
16.    ALine* Line = GetWorld()->SpawnActor<ALine>();
17.    ActorToSpawn.Horizontal ? Line-
   >CreateHorizontal(ActorToSpawn.Position) : Line-
   >CreateVertical(ActorToSpawn.Position);
18.    }
19.    if (LoadedData->SaveAngleLines.Num() > 0)
20.    for (FSaveAngleLine ActorToSpawn : LoadedData->SaveAngleLines)
21.    {
22.    AAngleLine* Line = GetWorld()->SpawnActor<AAngleLine>();
```

```
23.    Line->CreateAngle(ActorToSpawn.LineTransform,
   ActorToSpawn.FirstPoint, ActorToSpawn.SecondPoint);
24.    }
25.    if (LoadedData->SaveChannel.Num() > 0)
26.    for (FSaveChannel ActorToSpawn : LoadedData->SaveChannel)
27.    {
28.    AParallelChannel* LoadChannel = GetWorld()-
   >SpawnActor<AParallelChannel>();
29.    LoadChannel->CreateParallelChannel(ActorToSpawn.BotLine,
   ActorToSpawn.TopLine, ActorToSpawn.FirstPoint,
   ActorToSpawn.SecondPoint, ActorToSpawn.ThirdPoint,
   ActorToSpawn.ForthPoint);
30.    }
31.    }
32.    }
```

ЗАКЛЮЧЕНИЕ

В данной работе были рассмотрены методы разработки интерактивных элементов пользовательского интерфейса в трехмерном движке Unreal Engine 4, с акцентом на использование Static Mesh Component. Этот компонент представляет собой мощный инструмент для создания статичных 3D-моделей в игровом мире, обладающий широким спектром возможностей для настройки и адаптации под конкретные нужды проекта.

Static Mesh Component позволяет разработчикам создавать разнообразные интерактивные элементы пользовательского интерфейса, такие как кнопки, переключатели и слайдеры, с высокой степенью настраиваемости в плане формы, размера, текстур и анимаций. Это обеспечивает гибкость и уникальность в создании пользовательских интерфейсов, что способствует созданию более привлекательного и интуитивно понятного опыта для пользователей.

Кроме того, Static Mesh Component взаимодействует с другими компонентами и функциями Unreal Engine 4, позволяя разработчикам интегрировать различные виды интерактивности и эффектов. Такая функциональность расширяет границы традиционных пользовательских интерфейсов и позволяет создавать более сложные и динамичные взаимодействия в игровом мире.

Также была подчеркнута важность оптимизации при работе с трехмерными объектами, которая является ключевым фактором для обеспечения плавной работы и высокой производительности игр. Static Mesh Component поддерживает различные оптимизации, включая упрощение геометрии, использование текстурных атласов и оптимизацию использования вычислительных мощностей компьютера, что позволяет создавать эффективные и быстро загружаемые интерактивные элементы.

В ходе данной работы также был исследован класс USaveGame в Unreal Engine 4, который отвечает за сохранение и загрузку игровых данных. Были рассмотрены его структура, функционал и применение. USaveGame представляет собой важный компонент любой игры, построенной на движке Unreal Engine 4, так как он позволяет сохранять и восстанавливать состояние игры, что улучшает игровой процесс и повышает удобство использования. Кроме того, этот класс предоставляет разработчикам гибкие возможности для управления сохранениями и загрузками, что

позволяет создавать индивидуальные решения для каждого конкретного проекта. В рамках преддипломной практики были изучены основные методы класса USaveGame, такие как SaveGame, LoadGame, а также рассмотрены способы их использования в коде. Было продемонстрировано, как создать собственный класс сохранений, наследующийся от USaveGame, и реализовать в нем необходимый функционал.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Unreal Engine 4 Documentation // Unreal Engine Documentation URL: https://docs.unrealengine.com/. Дата обращения: 11.10.2023.
2. Display aspect ratio // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Display_aspect_ratio/. Дата обращения: 09.10.2023.
3. Geometry instancing // Wikipedia, the free encyclopedia URL: https://en.wikipedia.org/wiki/Geometry_instancing. Дата обращения: 13.11.2023.
4. Working with Static Mesh Actors and Components // Epic Games Development Documentation URL: https://dev.epicgames.com/community/learning/tutorials/qzoY/unreal-engine-working-with-static-mesh-actors-and-components. Дата обращения: 17.11.2023.
5. Tools referencing // Tradingview site URL: https://ru.tradingview.com/ Дата обращения: 18.11.2023.
6. Procedural Mesh Component Documentation // Unreal Engine Documentation URL: https://docs.unrealengine.com/4.27/en-US/API/Plugins/ProceduralMeshComponent/UProceduralMeshComponent/. Дата обращения: 23.11.2023
7. Instanced Static Mesh Component // Unreal Documentation URL : https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Runtime/Engine/Elements/SMInstance?application_version=5.1. Дата обращения : 17.04.2024
8. Экспорт финансовых графиков в формате CSV // Tradingview site URL: https://www.tradingview.com/blog/ru/export-chart-data-in-csv-14395/. Дата обращения: 13.04.2024
9. USaveGame base class // Unreal Documentation URL: https://docs.unrealengine.com/4.27/en-US/API/Runtime/Engine/GameFramework/USaveGame/ . Дата обращения: 16.05.2024
10. USTRUCT base class // Unreal Documentation URL: https://docs.unrealengine.com/4.26/en-US/API/Runtime/CoreUObject/UObject/UStruct/ . Дата обращения: 19.05.2024

11. *UGameplayStatics class in Unreal Engine // VREALMATIC Site URL: https://vrealmatic.com/unreal-engine/classes/ugameplaystatics . Дата обращения: 18.05.2024*