# ВВЕДЕНИЕ

Музыкальное сопровождение в видеоиграх является крайне важным аспектом в разработке игр. Правильно выбранная или написанная музыка позволяет авторам передать больше эмоций, лучше раскрыть персонажей, сделать акцент на каком-то важном для игры аспекте, или же просто задавать настроение по мере прохождения этой игры. Но кроме эстетического и художественного аспекта, музыка может выступать основной составляющей, вокруг которой строятся все взаимодействия, атаки, реакции игрового мира на игрока, либо на внешние взаимодействия с ним.

Игры, в которых музыка лежит в основе геймплея, то есть отвечает или влияет на важнейшие для игры аспекты взаимодействия, управления или подходы к взаимодействию с игрой относят к жанру ритм игр. На рынке видеоигр, игры данного жанра могут выступать в форме тренировочной площадки для обучения игры на каком-либо инструменте, способ потренировать реакцию, заняться разминкой или же продуктом для простого получения удовольствия от игры.

Разработку всех ритм игр объединяет необходимость разработчиков в выборе готовых или написании собственных алгоритмов для анализа используемых аудиофайлов. Существуют разные подходы к решению проблемы интерпретации ударов в минуту, в дальнейшем bpm (от английского beats per minute), задаваемых ритмом аудиофайла, в интерактивные события, происходящие или создаваемые игроком. Данные подходы могут сильно отличатся от используемых технологий, а так же от количества используемых аудиофайлов в проекте и политики компании.

Целью данного дипломного проекта является разработка игрового прототипа, с использованием алгоритмов обработки цифровой спектрограммы аудиофайлов, позволяющих анализировать файл и вычислять bpm для создания событий и дальнейшего построения взаимодействий в игре. Такие алгоритмы позволяют сэкономить большое количество человеко-часов и ресурсов компании на поддержании игры в последующем в качестве сервиса.

1. **ОБЗОР ЛИТЕРАТУРЫ**
   1. **Обзор существующих аналогов игр**

В данном разделе рассмотрим ярких представителей жанра ритм игр и игр, использующих алгоритмы для анализа аудио сигналов.

* + 1. **Guitar Hero**

Одним из известнейших примеров жанра ритм игр является серия игр Guitar Hero (рисунок 1.1).



Рисунок 1.1 – Ритм игра «Guitar Hero» [[1](https://www.youtube.com/watch?v=slJA7xwDRWk)]

Первая игра серии вышла в 2005 году и известна тем, что управление в игре осуществлялось посредством специального контроллера в виде уменьшенной копии гитары. В качестве примера, первая часть серии включала в себя 47 композиций. Данная серия игр стала основой для множества подобного рода игр, в последующем занявших крупную долю рынка мобильных игр.

Основной целью игры являлось своевременное нажатие необходимых кнопок на специализированном контроллере (рисунок 1.2). Все кнопки контроллера были окрашены 5 различных цветов, означающих определённый лад. Кроме того на корпусе имелась педаль, отвечающая за активацию струн. Успешным считалось нажатие на данную педаль, с одновременно нажатыми кнопками ладов, отображаемых на экране игрока. Нажатие было необходимо производить именно в тот момент, когда окружность некоторого цвета, доходила до находящихся зон внизу экрана. По итогу успешных нажатий, игроку начислялись очки, влияющие на рейтинг данной попытки прохождения.

Основной целью игры являлось своевременное нажатие необходимых кнопок на специализированном контроллере (рисунок 1.2). Все кнопки контроллера были окрашены 5 различных цветов, означающих определённый лад. Кроме того на корпусе имелась педаль, отвечающая за активацию струн. Успешным считалось нажатие на данную педаль, с одновременно нажатыми кнопками ладов, отображаемых на экране игрока. Нажатие было необходимо производить именно в тот момент, когда окружность некоторого цвета, доходила до находящихся зон внизу экрана. По итогу успешных нажатий, игроку начислялись очки, влияющие на рейтинг данной попытки прохождения.



Рисунок 1.2 – Контроллер для Guitar Hero

При разработке данного продукта планировалось использовать лишь определенное количество лицензированных треков. В связи с этим, при создании набора проигрываемых нот, разработчики не использовали алгоритмы для анализа аудиофайлов. Вместо этого акцент был сделан на ручное воссоздание гитарных аккордов под специализированный контроллер, для создания ощущения игры на настоящей гитаре. Такое решение могло быть так же обосновано финансовыми возможностями компании и ограничениями технологий того времени.

Данный подход является более выгодным и точным по сравнению с использованием алгоритмов, при условии, что количество аудиофайлов не будет превышать некоторый порог. В своё время порог определяется штатом и финансовыми возможностями компании.

Минусом данного подхода можно выделить сложность к масштабируемости продукта в будущем. Алгоритмы для частотного анализа аудиофайлов позволили бы в дальнейшем сэкономить на разработке данного проекта, а так же удешевить выпуск дополнительного контента.

* + 1. **Osu!**

«Osu!» – это бесплатная ритм игра, в которой имеется четыре официальных игровых режима, отличающихся игровыми механиками (рисунок 1.3). Управление в игре осуществляется посредством различных периферийных устройств: компьютерной мыши, клавиатуры, графического планшета, сенсорного устройства.



Рисунок 1.3 – Ритм игра «Osu!»

Отличительной особенностью данного проекта является то, что «Osu!» является платформой, позволяющей игрокам добавлять свои композиции, предоставляя игрокам инструменты для добавления собственного фона, композиций, настройки времени появления, времени задержек и позиций интерактивных объектов.

При установке данного продукта, пользователь обнаружит лишь пару тестовых композиций. Это связано с тем, что библиотека доступных треков хранится на официальном сайте продукта и обладает отрытым доступом для добавления и скачивания, готовых к игре композиций (рисунок 1.4). Данные композиции хранятся в специальных файлах формата, хранящих в себе саму композицию, настройки, а так же порядок и положение интерактивных элементов.

Цель игры заключается в своевременном нажатии на появляющиеся в такт музыки ноты, удержание курсора в пределах подвижных слайдеров и раскручивание спиннеров на максимально возможное значение. За все вышеперечисленные действия игроку начисляются очки рейтинга. Чем точнее игрок попадает в такт или удерживает курсор в центре указанных зон, тем больше очков рейтинга ему начисляется. Полученные очки влияют на позицию игрока в мировом рейтинге игроков.

В данной игре частично используются алгоритмы для частотного анализа композиций, но остаётся необходимость вручную определять положение нот на экране, а так же других интерактивных элементов. Данных аспект является основным минусом такого подхода, потому как конечному пользователю хочется как можно быстрее получить желаемый результат в виде готовой интерактивной композиции, а не задумываться о расположении элементов по мере игры.

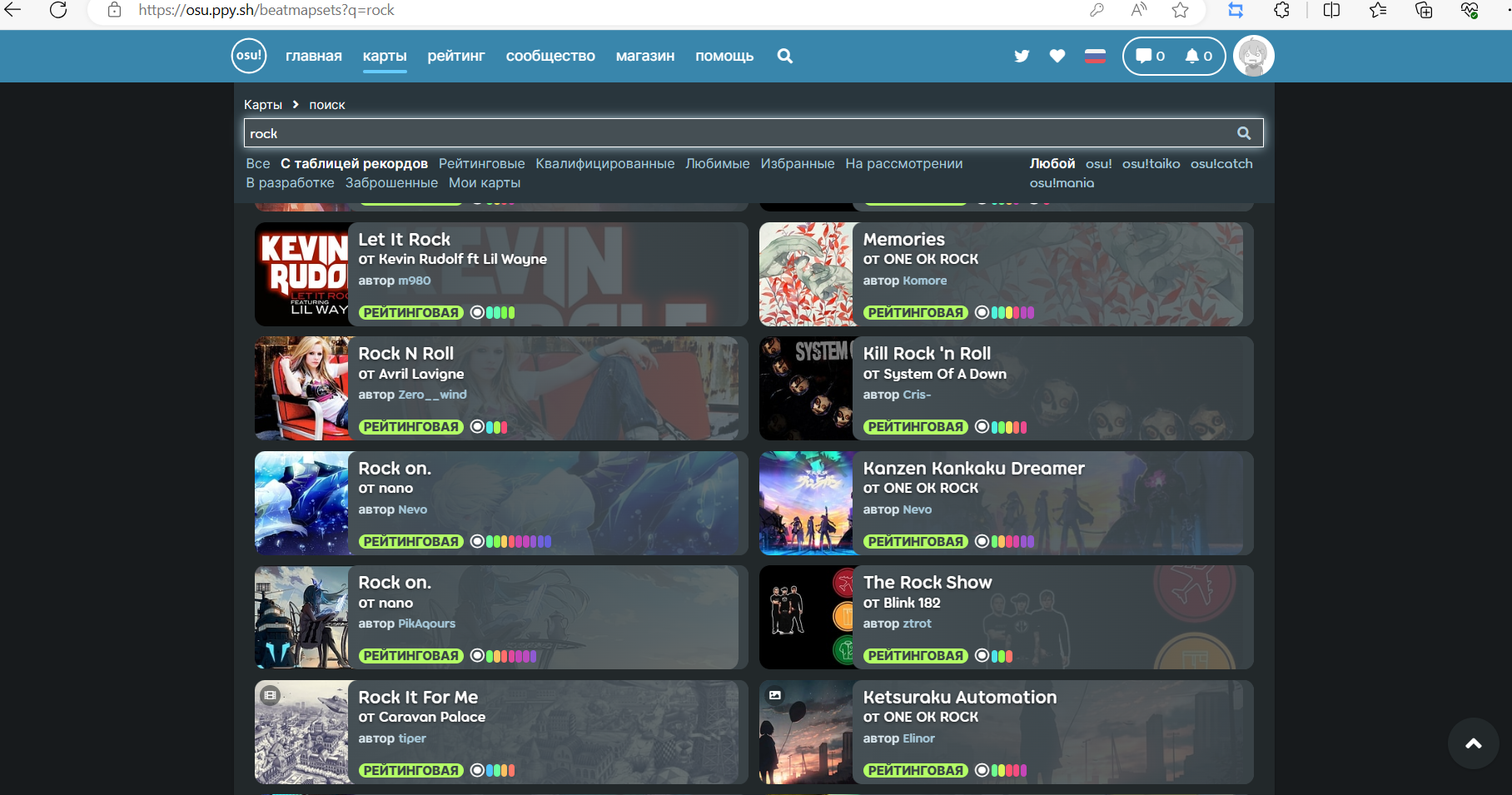


Рисунок 1.4 – Библиотека игры «Osu!» [2]

В связи со своей доступностью и наличием большой библиотеки композиций, данный проект стал крайне популярен как среди фанатов жанра игры и определённых прослоек пользователей, увлекающихся восточной музыкой, так и среди пользователей, использующих данную платформу для тренировки реакции и точности нажатия клавиш.

* + 1. **One Hand Clapping**

One Hand Clapping – это вокальный 2D платформер от небольшой инди студии (рисунок 1.5). Несмотря на небольшую известность проекта, хочется отметить нестандартный подход разработчиков к реализации взаимодействия игрока с внутри игровым миром.

В отличие от вышеперечисленных примеров, в данной игре отсутствует фактор борьбы с другими игроками за позицию в локальном или мировом рейтинге. Так же в данной игре не делается фокус на точности нажатия кнопок в соответствии с музыкальным сопровождением. One Hand Clapping предлагает игрокам использовать свой голос как инструмент для преодоления препятствий и решения головоломок. Игра погружает в фантастический мир, где правильные звуковые частоты помогают освещать путь, создавать платформы и управлять элементами окружения. Это не только новый способ взаимодействия с игровым миром, но и возможность для игроков тренировать свои вокальные навыки и чувство ритма.



Рисунок 1.5 – Игра One Hand Clapping

Управление в игре осуществляется посредством клавиатуры и микрофона. Все головоломки в игре построены так, что для их решения необходимо пропевать ноты определенной тональности. За счёт встроенных алгоритмов анализа аудио сигнала, поступающий в микрофон сигнал, преобразуется в частотно временной график, кривая которого, как пример, может послужить мостом для прохождения препятствий (рисунок 1.6). Все взаимодействия с персонажами и объектами в игре, так же зачастую осуществляют посредством голосового ввода.



Рисунок 1.6 – Пример использования голоса в One Hand Clapping [3]

* 1. **Обзор инструментов и технологий**

Данный подраздел включает в себя краткий обзор инструментов и технологий, которые использовались при разработке дипломного проекта.

* + 1. **Unreal Engine 4**

Unreal Engine 4 – является программной платформой, разработанной компанией Epic Games. Имеет в наличие широкий спектр инструментов, для создания визуальных эффектов, 3D графики, игр и симуляций. Четвертая версия данной платформы была разработана в 2012 году. Именно с четвертой версии данное программное обеспечение перешло на бесплатную форму распространения. С момента выпуска Unreal Engine 4 стала одной из самых популярных платформ для разработки игр по всему миру (рисунок 1.7).

Основные аспекты, выделяющие Unreal Engine 4 на фоне других платформ для разработки игр:

1. Большое сообщество. Unreal Engine 4 обладает активным сообществом разработчиков. Существует форум разработчиков Epic, где обсуждаются различные темы, связанные с разработкой на Unreal Engine 4. Специалисты компании часто отвечают на вопросы по разработке, а так же осуществляют консультацию и помощь с разработкой для компаний, работающих с продуктом компании. Кроме того, Unreal Engine предлагает магазин, где пользователи могут приобрести и продать различные плагины и контент, созданный другими пользователями.



Рисунок 1.7 – Рабочая среда Unreal Engine 4

**2.** **Языки программирования.** Unreal Engine 4 поддерживает C++ и проприетарного Blueprint Visual Scripting для создания игровых элементов. Blueprint Visual Scripting – это визуальная система скриптов, которая позволяет разработчикам значительно увеличить скорость прототипирования и создания полноценных игровых элементов, без написания кода.

**3.** **Качественная документация**. Документация Unreal Engine 4 доступна на многих языках и содержит подробные сведения по установке и работе с программным обеспечением. Она включает в себя введение в программирование на C++, обзоры предустановленных функций и технологий, руководства по работе с определёнными аспектами платформы, будь то графика или физические симуляции, и многое другое. Также имеется поддержка следующих инструментов программирования на С++:

3.1 C++ Class Wizard: Инструмент для быстрого создания новых классов и компонентов;

3.2 Console Variables:Инструмент для управления различными параметрами с использованием консоли;

3.3 Low-Level Memory Tracker: Инструмент для детального отслеживания используемой создаваемым продуктом памяти устройства.

**4.** **Поддержка современных технологий компьютерной графики**. Unreal Engine 4 поддерживает множество современных технологий компьютерной графики. Он включает в себя системы освещения и теней, материалы и текстуры, визуальные эффекты и постобработку. Кроме того, Unreal Engine 4 поддерживает технологии RTX, Dolby Atmos, VR, ARCore, Niagara, Houdini и многие другие.

**5.** **Поддержка всех известных платформ**. Unreal Engine 4 поддерживает разработку для множества платформ, включая Windows, macOS, Linux, iOS, Android, PlayStation, Xbox, Nintendo Switch и другие.

6. Интеграция со средой разработки. Для разработки на C++ в Unreal Engine 4 используется Microsoft Visual Studio или Apple Xcode. Для корректной работы данных сред разработки, Epic Games были разработаны специализированные плагины, которые находятся в открытом доступе.

7. Модульная архитектура. Unreal Engine 4, построен на модульной архитектуре. Имеется возможность добавления собственных модулей и расширений функциональности платформы.

* + 1. **Microsoft Visual Studio**

Microsoft Visual Studio – это интегрированная среда разработки (IDE), разработанная компанией Microsoft (рисунок 1.8). Является одним из важнейших инструментов, используемых при разработке данного дипломного проекта и для разработки программного обеспечения в целом.

Данная интегрированная среда разработки поддерживает широкий спектр языков программирования, таких как C++, Node.js, Python и R, что позволяет разработчикам создавать приложения для различных платформ, включая Windows, Azure и многие другие. Одной из выдающихся особенностей Visual Studio является её способность улучшать продуктивность разработчиков за счёт таких функций, как IntelliSense, которая предлагает авто дополнения кода, и CodeLens, которая предоставляет полезную информацию прямо в редакторе кода.

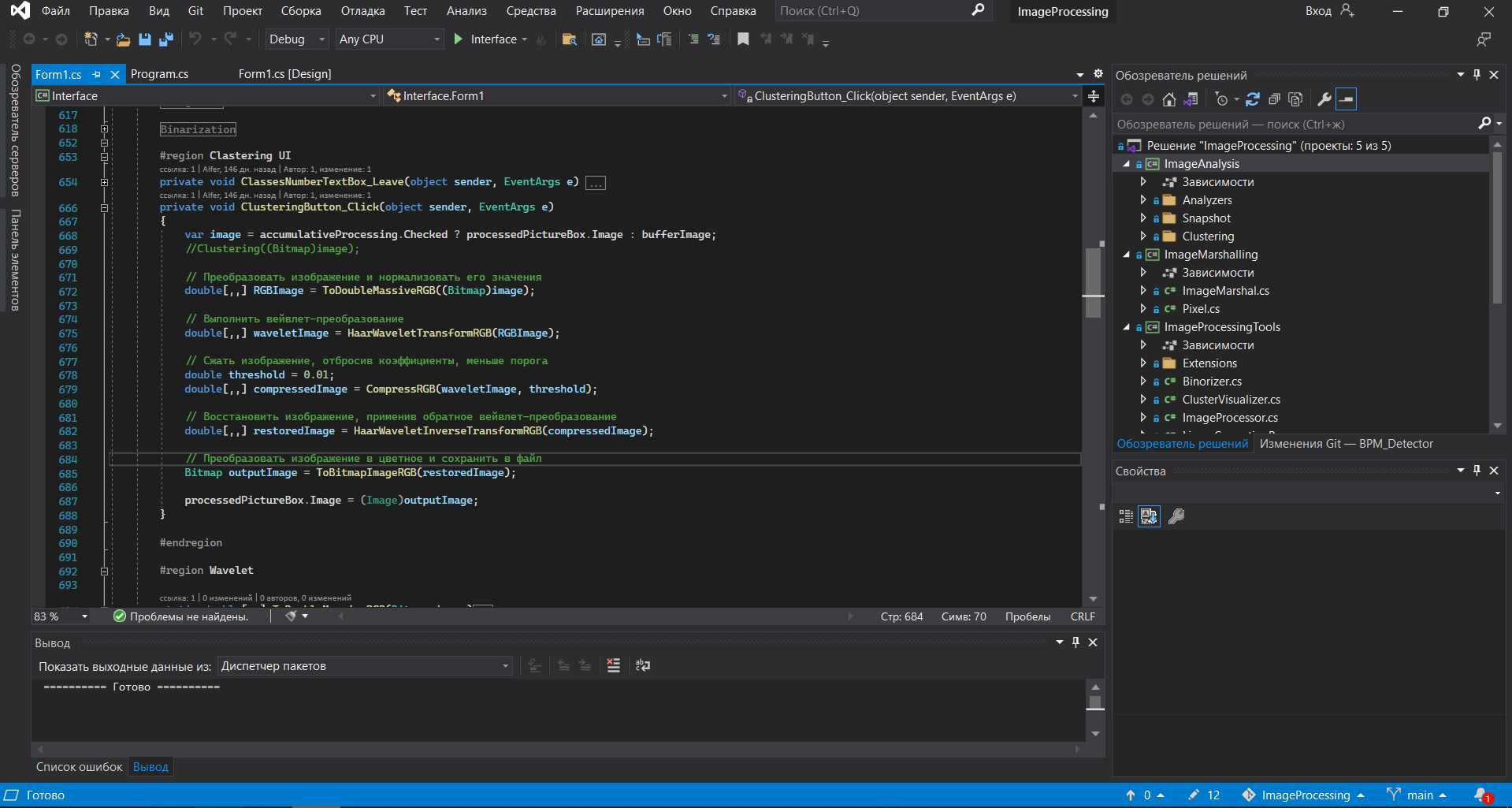


Рисунок 1.8 – Интегрированная среда разработки Microsoft Visual Studio

Visual Studio интегрируется с GitHub и Unreal Engine 4, что упрощает управление версиями и работу над проектом. Кроме этого, данное ПО обладает детальными настройками, как самой среды разработки, так и разрабатываемого проекта. Для более тонкой настройки можно использовать доступные к установке плагины и дополнительные инструменты от других пользователей. Эти инструменты и возможности интеграции делают Visual Studio идеальным выбором для командных и одиночных проектов.

Visual Studio также выделяется на фоне конкурентов благодаря своим возможностям для отладки и диагностики, предоставляя разработчикам инструменты для эффективного поиска и устранения ошибок в коде. Это включает в себя визуализации для асинхронных операций и автоматические анализаторы, которые помогают оптимизировать производительность и качество кода. Такие возможности делают Visual Studio не только инструментом для написания кода, но и платформой для обеспечения высокого качества разработки программного обеспечения.

Использование Visual Studio в дипломном проекте позволяет значительно ускорить процесс разработки и повысить его качество, благодаря широкому спектру инструментов и функций, которые она предлагает.

* + 1. **FL Studio 21**

FL Studio является мощным и гибким инструментом, который предоставляет пользователям широкий спектр возможностей для создания музыки. С момента своего первого выпуска в 1998 году, FL Studio претерпела множество изменений и улучшений, став одной из самых популярных программ для музыкальных продюсеров.

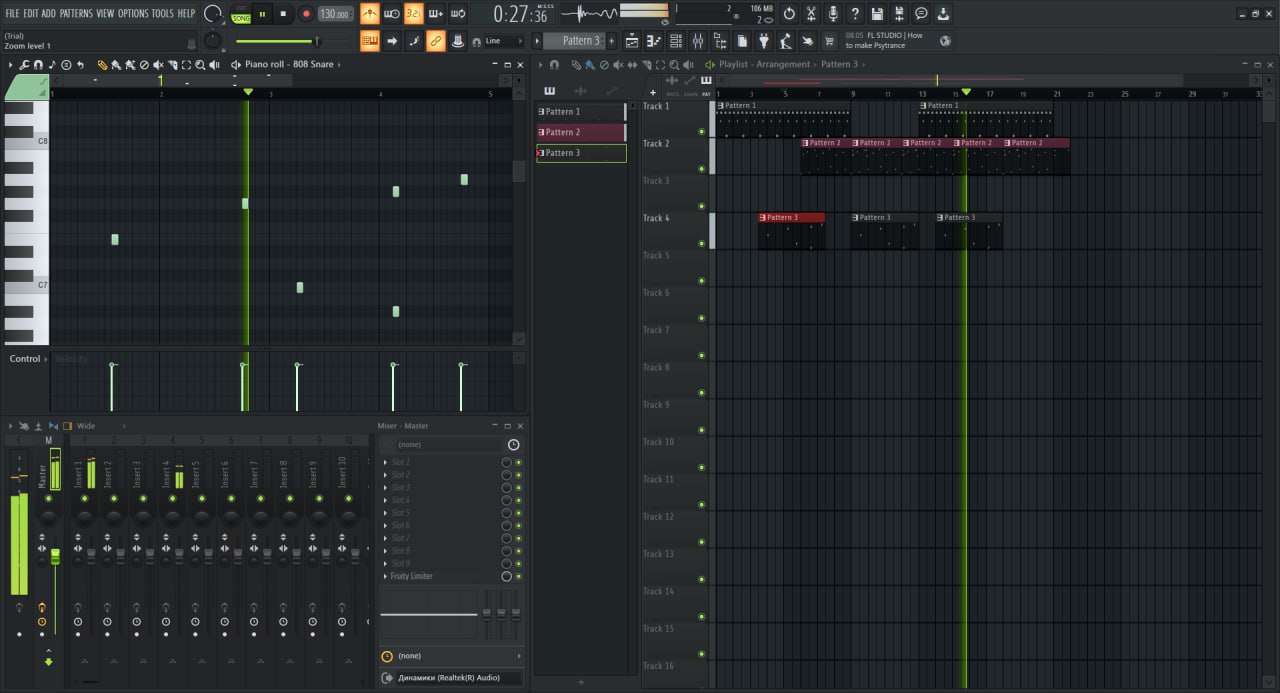


Рисунок 1.9 – Пример создания музыки в FL Studio 21

Программа предлагает интуитивно понятный интерфейс, который позволяет пользователям легко навигировать по различным функциям и инструментам, обеспечивая эффективный и гладкий рабочий процесс (рисунок 1.9). В FL Studio включен обширный набор инструментов, таких как синтезаторы, сэмплеры и барабанные машины, а также библиотека звуков, которая предоставляет пользователям все необходимое для создания музыки. Программа также предлагает возможности для живых выступлений и поддерживает обновления, что делает ее идеальным выбором для музыкантов и продюсеров всех уровней. FL Studio подходит как для начинающих, так и для опытных продюсеров, и включает в себя все необходимое для превращения музыкальных идей в реальность.

Кроме того, FL Studio 21 предлагает уникальные функции, такие как многоканальная запись, автоматизация клипов и встроенные плагины, которые обеспечивают пользователям большую гибкость при создании музыки. Все вышеперечисленные возможности позволят создать несколько музыкальных композиций с различными параметрами.

FL Studio так же поддерживает широкий спектр аудиоформатов, что позволяет пользователям импортировать и экспортировать музыку в любом формате.

Использование FL Studio 21 в дипломном проекте позволяет значительно ускорить процесс разработки и повысить его качество, благодаря широкому спектру инструментов и функций, которые она предлагает.

1. **СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

После анализа всех требований к разрабатываемому проекту, мы переходим к разбиению системы на функциональные блоки. Этот метод позволяет создавать гибкую архитектуру приложения, что в свою очередь позволяет изменять существующие и добавлять новые функциональные блоки, не затрагивая общую работу системы.

Структура проекта состоит из следующих блоков:

* блок игровой логики;
* блок игрового контроллера;
* блок игрового интерфейса;
* блок игровых объектов;
* блок игрового меню;
* блок чтения аудиофайла;
* алгоритм анализа аудиосэмплов;
* алгоритм вычисления музыкального ритма.

Взаимосвязь основных блоков проекта отображена на структурной схеме ГУИР.400201.304 С1.

* 1. **Блок игровой логики**

Этот раздел включает в себя фундаментальные принципы и правила, которые определяют структуру и механизмы взаимодействия между игроком и элементами игры. В нем детально разъясняются ключевые логические составляющие, которые являются основой для игрового процесса и включают в себя следующие аспекты:

– критерии, при которых игрок считается проигравшим партию;

– методика начисления игровых баллов в зависимости от действий, предпринимаемых игроком во время игры;

– определение системы отношений и взаимодействий между игровой фигурой и другими элементами игрового процесса;

– механизм переключения между различными игровыми уровнями и интерфейсом игрового меню;

– разработка параметров физического поведения как неподвижных, так и подвижных объектов в игре.

Кроме того, данный блок является связующим для всех других блоков в структуре данного проекта. Необходимо рассмотреть возможность добавления системы триггеров, которая будет активировать определенные события при выполнении игроком или игровыми объектами конкретных условий.

Использование визуального языка программирования Blueprints в Unreal Engine 4 позволяет быстро и эффективно реализовывать сложные игровые логики, делая процесс разработки более доступным для дизайнеров и разработчиков всех уровней.

* 1. **Блок игрового контроллера**

Блок игрового контроллера создан для того, чтобы обеспечить широкий спектр возможностей для взаимодействия игрока с виртуальным миром посредством управления игровым персонажем. В рамках этого блока подробно разъясняются ключевые аспекты, такие как:

– основы перемещения персонажа по игровому пространству

– взаимодействие персонажа с различными объектами внутри этого пространства, а также реакции, которые персонаж проявляет в ответ на разнообразные события, происходящие в игре.

– установление взаимосвязей между взаимодействием игрока с устройствами управления, а так же происходящими игровыми событиями, в соответствии с конкретными нажатиями пользователя.

Блок игрового контроллера является ключевым элементом для обеспечения интерактивности в игровом процессе, так как он отвечает за преобразование действий игрока в конкретные команды для игровой пешки. В процессе управления, когда игрок нажимает клавиши на клавиатуре или кнопки на геймпаде, система должна регистрировать эти нажатия и генерировать соответствующие события. Важно отметить, что блок управления персонажем должен быть спроектирован таким образом, чтобы он мог распознавать только те нажатия клавиш или кнопок, которые были заранее определены разработчиками как ответственные за выполнение определенных действий игровой пешкой.

Дополнительно, данный блок контроллера игры предоставляет возможность отстраниться от конкретного персонажа или пешки, которыми игрок будет управлять через контроллер, что позволяет более гибко подходить к процессу разработки. Специфические функции, предоставляемые движком Unreal Engine 4, дают осуществлять смену управления между разными пешками и персонажами, что в свою очередь способствует поддержанию гибкости и масштабируемости создаваемых модулей кода.

* 1. **Блок игрового интерфейса**

Блок игрового интерфейса является ключевым элементом для передачи информации пользователю. Данный блок отвечает за визуализацию самого игрового процесса, отображение различных параметров или индикаторов, необходимых для отображения пользователю в реальном времени. В данном случае данный блок осуществляет контроль и вывод счётчика очков.

Для эффективной реализации и оптимизации данного блока, разработчики используют продвинутый набор инструментов для работы с графическими интерфейсами, предоставляемый Unreal Engine 4. Данные инструменты позволяют создавать как высококачественные, так и простые графические пользовательские интерфейсы, обеспечивая при этом гибкость и широкие возможности для кастомизации. Язык визуального программирования Blueprints, используемый для описания логики, облегчает процесс разработки, поскольку он позволяет быстро прототипировать и тестировать идеи без необходимости писать сложный код. Это делает его идеальным инструментом для создания кастомизированных и гибких игровых интерфейсов.

* 1. **Блок игровых объектов**

Блок игровых объектов в контексте разработки игр на Unreal Engine 4 представляет собой критически важный компонент, обеспечивающий функциональность и интерактивность игрового мира. Этот блок служит основой для создания и управления различными элементами игры, которые взаимодействуют с игроком и окружающей средой.

В контексте Unreal Engine 4, игровые объекты представляют собой экземпляры классов, которые взаимодействуют друг с другом и с окружающим миром во время игрового процесса. Важно отметить, что блок игровых объектов не ограничивается только персонажами или видимыми элементами игры. Он также включает в себя невидимые элементы, которые влияют на игровой процесс, такие как системы физики, искусственного интеллекта, звуковые эффекты и другие.

Блок игровых объектов также включает в себя материалы, которые определяют внешний вид объектов, а также как они взаимодействуют друг с другом и с игровой средой. Материалы могут варьироваться от простых цветов и текстур до сложных шейдеров, создающих реалистичные эффекты, такие как отражение, преломление, освещение и тени.

В целом, блок игровых объектов играет ключевую роль в создании динамичного, интерактивного и визуально привлекательного игрового мира. Он служит основой для создания разнообразных игровых сценариев и обеспечивает гибкость и масштабируемость, необходимые для реализации как сложных, так и более простых игровых механик и интерактивных элементов. Этот блок является неотъемлемой частью любого проекта на Unreal Engine 4 и требует глубокого понимания и внимательного подхода при его реализации.

Для создания и настройки данного блока игровых объектов будет использоваться язык визуального программирования Blueprints, который является частью инструментария Unreal Engine 4. Этот выбор обусловлен тем, что Blueprints позволяет разработчикам быстро и эффективно создавать логические системы для создаваемых объектов без необходимости писать обширный код, а также предоставляет возможность легко настраивать взаимодействия объектов, материалов и компонент дипломного проекта.

* 1. **Блок игрового меню**

Блок игрового меню, описывает комплекс элементов интерфейса, предназначенных для обеспечения взаимодействия между игроком и программным обеспечением. При разработке данных элементов использовались специализированные инструменты, встроенные в игровой движок Unreal Engine 4, которые позволяет создавать высококачественные графические пользовательские интерфейсы. В состав данного блока входит несколько ключевых компонентов:

– главное меню игры, которое является первым экраном, с которым сталкивается пользователь при запуске программы. Оно предоставляет доступ ко всем основным функциям игры.

– меню настроек, где игроки могут изменять графические параметры игры в соответствии со своими предпочтениями, что позволяет им настроить игровой процесс наиболее комфортно для себя.

– меню выбора аудиофайла, предоставляющее возможность выбрать музыкальное сопровождение, которое будут воспроизводиться во время игры.

– меню помощи, отображающее пользователям возможные способы управления в данной игре, а так же привязку определённых клавиш к определённым действиям игрового персонажа.

Для эффективной реализации и оптимизации данного блока, используется продвинутый набор инструментов, предоставляемый Unreal Engine 4. Данные инструменты позволяют создавать как простые так и более продвинутые игровые меню, с возможностью широкой кастомизации. Реализация логики взаимодействия с интерфейсами реализована на языке визуального программирования Blueprints.

* 1. **Блок чтения аудиофайла**

Блок алгоритма осуществляющего чтение заголовка и блока данных аудиофайла. Первоначальной задачей алгоритма является определение формата файла, валидация событий, приводящих к возникновению ошибок при чтении и открытие файла в режиме чтения. Аудиофайл должен соответствовать формату .wav, так как данный формат файла используется для хранения несжатого аудиосигнала в импульсно-кодовой модуляции.

После открытия файла, главными задачами блока являются, чтение заголовка, инициализация структуры для сохранения данных из заголовка аудиофайла, таких как количество каналов, частоту дискретизации, аудиоформат, наличие и тип кодировки, количество байт для хранения одного сэмпла, а так же общий размер файла без учета первых 16 байт. После получения информации из заголовка файла, файл должен быть закрыт, и открыт снова, с учетом имеющейся информации о размере заголовка, и точки начала блока данных. Информация, полученная из заголовка, необходима для корректного чтения блока данных конкретного аудиофайла. Так же полученная информация будет использована для анализа композиции в блоке анализа аудиофайла.

При повторном открытии аудиофайла, указатель переносится на начало блока данных. Необходимо иметь в виду тот факт, что в зависимости от количества каналов, а так же частоты дискретизации, меняются параметры чтения аудиофайла. При чтении данных стерео аудиосигнала. Блок данных представляет собой пары значений для левого и правого канала. От частоты дискретизации аудиосигнала зависит размер набора значений, отвечающих за хранение информации об одной секунде аудиосигнала. Большая часть композиций используемых в данном дипломном проекте имеют частоты дискретизации 44100 Гц и 48000 Гц. В связи с тем, что для последующего анализа музыкального ритма, необходимо брать малый временной интервал, равный одной десятой секунды, размер анализируемого набора значений так же должен ровняться одной десятой от частоты дискретизации. Именно такой набор будет соответствовать данному временному промежутку.

Прочитанные наборы данных обрабатываются с помощью алгоритма быстрого преобразования Фурье. Такой подход позволяет сэкономить время и вычислительные ресурсы устройства, но приводят к потере верхних частот аудиосигнала. Наборы значений, полученные в результате операции быстрого преобразования Фурье, являются результатами работы данного блока и направляются для последующего анализа и вычисления музыкального ритма.

Данный алгоритм реализован с использованием языка программирования С++.

* 1. **Алгоритм анализа аудиосэмплов**

Блок алгоритма анализа аудиосэмплов, на основе информации, полученной из заголовка, а так же в результате чтения аудиофайла, анализирует полученные блоки данных. После анализа алгоритм подготавливает данные к передаче в следующий алгоритм вычисления музыкального ритма. Входными значениями алгоритма является массив комплексных чисел с амплитудно-частотными значениями, где модуль комплексного числа равняется амплитуде сигнала на определённой частоте, а индекс является частотным диапазоном, с учётом масштабирования частотных диапазонов к размеру полученного блока.

Основной задачей алгоритма анализа аудиосигнала является анализ амплитудных значений в басовом частотном диапазоне до 350 Гц, определение минимального и максимального амплитудного значения, для определения анализируемого амплитудного диапазона, а так же вычисление граничного значения, для последующего вычисления тех амплитудных значений, которые могут являться пиковыми и указывать на ритм музыкальной композиции.

Выходным значением является массив с максимальными амплитудными значениями, отображающими перепады амплитуд в басовом частотном диапазоне с частотой обновления 0.1 секунды. Так же выходными значениями алгоритма являются граничное значение и среднее арифметическое значение суммы всех максимальных значений амплитуд, которые в дальнейшем отправляется в алгоритм вычисления музыкального ритма.

Данный алгоритм реализован с использованием языка программирования С++.

* 1. **Алгоритм вычисления музыкального ритма**

В данном алгоритме, исходя из полученных данных алгоритмов чтения аудиофайла, а также из алгоритма анализа аудиосэмплов, происходит вычисление музыкального ритма композиции.

Входными значениями алгоритма являются массив максимальных амплитудных значений из басового частотного диапазона, граничное значение, для вычисления пиковых амплитуд, а так же среднее арифметическое суммы всех максимальных значений амплитуд.

Данный алгоритм вычисляет момент первого и последнего пикового значения, а так же сравнивает все значения из массива с граничным значением. В случае если амплитудное значение выше чем граничное значение, то производится сравнение конкретного значения, с четырьмя соседними значениями массива. Так же берётся учет того, что если среднее арифметическое суммы максимальных значений больше граничного значения, следовательно, музыкальный ритм превышает значение, приблизительно равное 150 ударом в минуту. В связи с этим, анализируемый диапазон соседних значений массива сужается до двух значений.

Итоговый ритм аудиозаписи равняется количеству пиковых значений в диапазоне делённых на разницу последнего и первого пикового амплитудного значения, и умноженного на 600. Таким образом, мы получаем количество ударов в минуту, для конкретного произведения.

Результатом данного алгоритма являются данные, на основе которых создаются игровые события, взаимодействующие с игроком через игрового персонажа.

1. **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

Функциональная структура проекта во многом аналогична структурной. Взаимосвязь между основными компонентами представлена на диаграмме классов ГУИР.400201.304 РР.1.

* 1. **Блок игровой логики**

В данном блоке находятся описания базовых логических законов и правил игры. Определение этих базовых элементов осуществляется в классах GM\_MenuMode и GM\_GameMode, которые являются наследуемыми классами от базового класса GameMode. Так же в этом блоке будут описаны классы, содержащие логику игровых уровней: BP\_MainMenu и BP\_GameMap. Ниже будут рассмотрены отдельные поля, функции, события и методы, определённые в данных классах и используемые в данном дипломном проекте.

**3.1.1 GameMode**

Базовый класс GameMode содержит в себе следующие поля:

Поле GameSessionClass хранит в себе объект класса GameSession, предназначенный для реализации сетевых игровых сессий. По умолчанию инициализируется объектом базового класса GameSession, что соответствует одиночному режиму игры.

Поле GameStateClass хранит в себе объект класса GameState, предназначенный для хранения специфичных аспектов игрового мира, а так же некоторых данных каждого отдельного пользователя в сетевых игровых сессиях. По умолчанию инициализируется объектом базового класса GameState, что соответствует одиночному режиму игры.

Поле PlayerControllerClass хранит в себе объект класса PlayerController, предназначенный для реализации всевозможных взаимодействий и передвижений подконтрольного персонажа или пешки. По умолчанию инициализируется объектом базового класса PlayerController, что приравнивается к отсутствию возможности взаимодействия управляемым персонажем или некоторой пешкой с игровым миром.

Поле PlayerStateClass хранит в себе объект класса PlayerState, предназначенный для хранения некоторых аспектов состояний и параметров подконтрольного персонажа или пешки. По умолчанию инициализируется объектом базового класса PlayerState, что соответствует отсутствию начальных состояний персонажа.

Поле HUDClass хранит в себе объект класса HUD, предназначенный для реализации пользовательского интерфейса игрока. По умолчанию инициализируется объектом базового класса HUD, что соответствует отсутствию как такового пользовательского интерфейса.

Поле DefaulPawnClass хранит в себе объект класса Pawn, предназначенный для хранения ссылки на базовый управляемый игроком объект. По умолчанию инициализируется объектом базового класса Pawn, что соответствует отсутствию подконтрольного объекта для PlayerController.

Поле SpectatorClass хранит в себе объект класса Pawn, предназначенный для реализации возможности наблюдения за игровым процессом со стороны. Чаще всего используется при разработке сетевых игр. По умолчанию инициализируется объектом базового класса Pawn, что равняется отсутствию наблюдателя.

Поле ReplaySpectatorPCClass хранит в себе объект класса PlayerController, предназначенный для реализации возможности управления для наблюдателя. По умолчанию инициализируется объектом базового класса PlayerController, что приравнивается к отсутствию возможности взаимодействия в роли наблюдателя.

Поле ServerStatReplicatorClass хранит в себе объект класса ServerStatReplicator, предназначенный для реализации хранения сетевых данных для репликации другим игрокам в сетевой игре. По умолчанию инициализируется объектом базового класса ServerStatReplicator, что соответствует одиночному режиму текущей игры.

Метод ConstructionScript() является базовым методом всех создаваемых классов в Unreal Engine 4, и представляет собой инициализацию параметров при создании объекта класса.

**3.1.2** **GM\_MenuMode**

Класс GM\_MenuMode, является производным от базового класса GameMode и хранит в себе основные игровые настройки для главного меню приложения.

Данный класс содержит следующие поля:

Поле DefaultPawnClass инициализируется значением None. Это необходимо для того, чтобы обеспечить фиксированную позицию камеры, а так же для корректного считывания управления при работе с главным меню игры, и другими окнами.

Метод GetGameMode() возвращает ссылку на объект данного класса GM\_GameMode.

Метод ChangeGameMode(GM\_GameMode) осуществляет замену текущего игрового режима GameMode на передаваемый объект класса GM\_GameMode.

**3.1.3 GM\_GameMode**

Класс GM\_GameMode, является производным от базового класса GameMode и хранит в себе основные игровые настройки для игрового процесса приложения.

Данный класс содержит следующие поля:

Поле GameSessionClass по умолчанию инициализируется значением None, в связи с тем, что разрабатываемый проект является одиночной игрой без создания сетевых сессий.

Поле PlayerControllerClass по умолчанию инициализируется ссылкой на объект класса BP\_PlayerController. Таким образом, к подконтрольному игроку объекту, указанному в поле DefaultPawnClass, будет привязано управление из инициализированного класса.

Поле HUDClass по умолчанию инициализируется ссылкой на объект класса BP\_PlayerHUD. Данный класс будет осуществлять контроль и вывод на экран количества очков, набранных игроком.

Поле DefaultPawnClass по умолчанию инициализируется ссылкой на объект класса BP\_PlayerCharacter. Объект указанного класса будет являться подконтрольным игроку, и появляться на игровом уровне благодаря классу PlayerStart.

Поле PlayerStart содержит в себе ссылку на объект класса PlayerStart. Данный объект представляет собой точку в пространстве XYZ для порождения базовой пешки игрока.

Метод GetGameMode() возвращает ссылку на объект данного класса GM\_GameMode.

Метод ChangeGameMode(GM\_GameMode) осуществляет замену текущего игрового режима GameMode на передаваемый объект класса GM\_GameMode.

**3.1.4 BP\_MainMenu**

Класс BP\_MainMenu описывает логику и параметры уровня игрового меню. Данный класс содержит следующие поля:

Поле GameMode хранит в себе указатель на объект класса GM\_MenuMode, содержащий предустановленные настройки данного уровня.

Событие EventBeginPlay() вызывается при создании уровня и осуществляет создание экземпляра класса игрового меню MainMenu, посредством функции CreateWidget(UserWidget). Результат функции выводиться на экран пользователя.

Событие EventConstruct() инициализирует базовые параметры класса. Данное событие вызывается при создании экземпляра класса.

**3.1.5 BP\_GameMap**

Класс BP\_GameMap описывает логику и параметры игрового уровня приложения. Данный класс содержит следующие поля:

Поле FloorPlatformsArray является массивом указателей, предназначенным для хранения указателей на все имеющиеся на данном уровне объекты класса BP\_FloorPlatform.

Событие EventBeginPlay() вызывается при создании уровня и осуществляет привязку игровой камеры к используемому объекту игрового контроллера BP\_PlayerController, посредством функции SetViewTargerWithBlend(PlayerController, CameraActor). Далее производится вызов вызов функции SetInputModeGameOnly (PlayerController, Widget, EMouseLockMode) для установления считывания принимаемых сигналов контроллера в режим «только для игры». Последним действием данного события, является вызов метода GetAllActorsOfClass(ActorsClass) и поиска всех существующих объектов относящихся к классу BP\_FloorPlatform. Результат метода записывается в поле FloorPlatformsArray.

Событие EventTick() вызываемое на каждом кадре игры. Данное событие осуществляет выбор случайного указателя из массива FloorPlatformsArray, посредством метода RandomIntegerInRange(). После получения указателя на объект BP\_FloorPlatform, производится проверка булевого поля IsActivate? и в случае отрицательного значения false, производится вызов пользовательского события IsActivateEvent().

* 1. **Блок управления**

Блок управления отвечает за аспекты управления во время игрового процесса, а так же за фиксацию нажатия тех или иных клавиш в игре. Данный блок представлен в виде классов PlayerController, BP\_PlayerController и Input.

* + 1. **PlayerController**

Класс PlayerController является базовым классом для реализации управления игровыми объектами, персонажами и любыми акторами находящимися на игровой сцене.

Базовый класс PlayerController содержит следующие поля:

Поле InputPitchScale отвечает за масштабирование ввода, который влияет на поворот камеры или персонажа по оси X.

Поле InputYawScale отвечает за масштабирование ввода, который влияет на поворот камеры или персонажа по оси Y.

Поле InputRollScale отвечает за масштабирование ввода, который влияет на поворот камеры или персонажа по оси Z.

Метод SetViewTargetWithBlend(PlayerController, CameraActor) осуществляет привязку принимаемого объекта CameraActor, к текущему, или принимаемому игровому контроллеру.

Метод GetInputPitchScale() возвращает значение переменной InputPitchScale.

Метод GetInputYawScale() возвращает значение переменной InputYawScale.

Метод GetInputRollScale() возвращает значение переменной InputRollScale.

Метод SetInputPitchScale(Float) присваивает передаваемое значение переменной InputPitchScale.

Метод SetInputYawScale(Float) присваивает передаваемое значение переменной InputYawScale.

Метод SetInputRollScale(Float) присваивает передаваемое значение переменной InputRollScale.

Событие EventConstruct() инициализирует базовые параметры класса. Данное событие вызывается при создании экземпляра класса.

* + 1. **BP\_PlayerController**

Класс BP\_PlayerController наследуется от базового класса PlayerController, и отвечает за управление объектом BP\_PlayerCharacter, посредством использования следующих пользовательских асинхронных событий: InputAxisMove\_Forward, InputAxisMove\_Left, InputActionInteract. Производный класс BP\_PlayerController включает в себя следующие поля:

Поле DistanceToMove определяет фиксированную дистанцию, на которую объект BP\_PlayerCharacter будет совершать перемещение.

Поле LastMovement хранит в себе время последнего совершённого передвижения игрока.

Поле LastInteract хранит в себе время последнего совершённого взаимодействия игрока.

Поле AsBP\_PlayerCharacter хранит в себе ссылку на объект BP\_PlayerCharacter, над которым игрок осуществляет управление.

Метод GetNewVectorForward() осуществляет вычисление точки в координатной плоскости XYZ, на которую управляемый объект BP\_PlayerCharacter должен произвести перемещение. Данный метод использует функции GetPlayerPawn() и GetActorLocation(), для получения текущей позиции объекта. После чего осуществляется прибавление значения DistanceToMove к полю X исходного вектора и объединение результатов в новый вектор. Полученное значение вектора возвращается результатом метода.

Метод GetNewVectorBack() осуществляет вычисление точки в координатной плоскости XYZ, на которую управляемый объект BP\_PlayerCharacter должен произвести перемещение. Данный метод использует функции GetPlayerPawn() и GetActorLocation(), для получения текущей позиции объекта. После чего осуществляется вычитание значения DistanceToMove от значения X исходного вектора и объединение результатов в новый вектор. Полученное значение вектора возвращается результатом метода.

Метод GetNewVectorLeft() осуществляет вычисление точки в координатной плоскости XYZ, на которую управляемый объект BP\_PlayerCharacter должен произвести перемещение. Данный метод использует функции GetPlayerPawn() и GetActorLocation(), для получения текущей позиции объекта. После чего осуществляется прибавление значения DistanceToMove к полю Y исходного вектора и объединение результатов в новый вектор. Полученное значение вектора возвращается результатом метода.

Метод GetNewVectorRight() осуществляет вычисление точки в координатной плоскости XYZ, на которую управляемый объект BP\_PlayerCharacter должен произвести перемещение. Данный метод использует функции GetPlayerPawn() и GetActorLocation(), для получения текущей позиции объекта. После чего осуществляется вычитание значения DistanceToMove от значения Y исходного вектора и объединение результатов в новый вектор. Полученное значение вектора возвращается результатом метода.

Метод GetPlayerController() возвращает ссылку на объект используемого контроллера класса BP\_PlayerController.

Метод CheckDeadZone(Float) принимает входное значение устройства ввода и сверяет его с диапазоном от -0.2 до 0.2. Результатом функции является булево значение.

Метод CheckLastMovementTime() сверяет поле LastMovement с текущим значением GetGameTimeInSeconds(). Возвращает true если разница значений больше 0.2 секунд.

Метод CheckLastInteractTime() сверяет поле LastInteract с текущим значением GetGameTimeInSeconds(). Возвращает true если разница значений больше 0.2 секунд.

Событие EventBeginPlay() вызывается после создания экземпляра объекта класса BP\_PlayerController. Задачей данного события является инициализация поля AsBP\_PlayerCharacter. Данное событие вызывает функцию GetPlayerPawn() и проверяет полученный результат на соответствие классу BP\_PlayerCharacter. После этого результат проверки записывается в поле AsBP\_PlayerCharacter.

Событие EventInputActionInteract() вызывается при нажатии игроком кнопки взаимодействия. Данное событие осуществляет проверку того, что CheckLastInteractTime() и canBeActivate объекта AsBP\_PlayerCharacter возвращают значение true. Если условие истинно, осуществляется вызов пользовательского события DeActivateEvent() для объекта класса BP\_FloorPlatform. После этого выполняется получение пользовательского интерфейса посредством функций GetPlayerController() и GetHUD(). Полученный объект сверяется на соответствие классу BP\_PlayerHUD. В случае успешного выполнения проверки, для данного объекта вызывается пользовательское событие IncrementScore. Значение поля LastInteract перезаписывается текущим временем с помощью функции GetGameTimeInSeconds().

Событие EventInputAxisMove\_Forward(Float) принимает значение текущего направления движения, поступающего с устройства управления пользователя по оси X. Данное событие вызывается при изменении входного значения, отличного от нуля. При вызове данного события осуществляется проверка входного значения методом CheckDeadZone(Float), а так же проверка времени последнего передвижения CheckLastMovementTime(). Если результаты проверок равняются true, осуществляется проверка направления движения по оси X.

При движении персонажа вперёд осуществляется вызов метода GetNewVectorForward(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию.

При движении персонажа назад осуществляется вызов метода GetNewVectorBack(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию.

По окончанию обработки события производится перезапись значения поля LastMovement.

Событие EventInputAxisMove\_Left(Float) принимает значение текущего направления движения, поступающего с устройства управления пользователя по оси Y. Данное событие вызывается при изменении входного значения, отличного от нуля. При вызове данного события осуществляется проверка входного значения методом CheckDeadZone(Float), а так же проверка времени последнего передвижения CheckLastMovementTime(). Если результаты проверок равняются true, осуществляется проверка направления движения по оси Y.

При движении персонажа влево осуществляется вызов метода GetNewVectorLeft(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию.

При движении персонажа вправо осуществляется вызов метода GetNewVectorRight(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию.

По окончанию обработки события производится перезапись значения поля LastMovement.

Событие EventInputActionGamePadMove\_Forward() вызывается при нажатии верхней кнопки крестовины геймпада. Осуществляется проверка на возможность осуществления движения CheckLastMovementTime(), а после вызов метода GetNewVectorForward(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector< Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию. По окончанию обработки события производится перезапись значения поля LastMovement.

Событие EventInputActionGamePadMove\_Back() вызывается при нажатии нижней кнопки крестовины геймпада. Осуществляется проверка на возможность осуществления движения CheckLastMovementTime(), а после вызов метода GetNewVectorBack(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию. По окончанию обработки события производится перезапись значения поля LastMovement.

Событие EventInputActionGamePadMove\_Left() вызывается при нажатии левой кнопки крестовины геймпада. Осуществляется проверка на возможность осуществления движения CheckLastMovementTime(), а после вызов метода GetNewVectorLeft(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию. По окончанию обработки события производится перезапись значения поля LastMovement.

Событие EventInputActionGamePadMove\_Right() вызывается при нажатии правой кнопки крестовины геймпада. Осуществляется проверка на возможность осуществления движения CheckLastMovementTime(), а после вызов метода GetNewVectorRight(). Результат метода передается в функцию проверки возможности передвижения на новую позицию – LineTraceByChannel(Vector<Double>, Vector<Double>, Vector<Actor>). Если результат метода равен false, управляемый объект BP\_PlayerCharacter осуществляет передвижение на новую позицию. По окончанию обработки события производится перезапись значения поля LastMovement.

* + 1. **Input**

Класс Input является так же ключевым базовым классом, отвечающим за переключение состояний считывания принимаемых сигналов. Основными методами данного класса, используемыми в проекте, являются:

Метод SetInputModeUIOnly(PlayerController, Widget, EMouseLockMode) принимает объект класса, наследуемого от базового класса PlayerController, и устанавливает считывание принимаемых сигналов контроллера в режим «только для пользовательского интерфейса». Данный метод используется для корректной работы с любого рода окнами графического интерфейса, наследуемыми от базового класса Widget.

Метод SetInputModeGameOnly(PlayerController, Widget, EMouseLockMode) также принимает объект класса, наследуемого от базового класса PlayerController, но устанавливает считывание принимаемых сигналов контроллера в режим «только для игры».

**3.3 Блок игрового интерфейса**

Блок игрового интерфейса отвечает за отображение игровой информации пользователю в реальном времени непосредственно во время игрового процесса. Данный блок представлен в виде базового класса HUD и дочернего класса BP\_PlayerHUD.

**3.3.1 HUD**

Базовый класс HUD предоставляет следующие базовые поля и методы для реализации игрового интерфейса:

Поле ShowHUD хранит в себе булево значение о состоянии отображения интерфейса пользователя.

Метод GetShowHUD() возвращает значение булевого поля ShowHUD.

Метод SetShowHUD(Boolean) осуществляет присвоение принимаемого значения полю ShowHUD.

Метод GetOwningPawn() возвращает ссылку на объект класса Pawn, управляемый игроком.

Метод GetOwningPlayerController() возвращает ссылку на объект класса PlayerController игрового контроллера, управляющего пешкой игрока.

**3.3.2 BP\_PlayerHUD**

Класс BP\_PlayerHUD является дочерним классом HUD. Данный класс содержит основные поля и методы для вывода пользователького интерфейса игроку.

Класс BP\_PlayerHUD содержит следующие поля:

Поле HUDWidget хранит ссылку на объект класса Widget, осуществляющий функцию пользовательского интерфейса для игрока.

Метод GetHUDWidget() возвращает ссылку на объект класса Widget, хранящуюся в поле HUDWidget.

Метод SetHUDWidget(Widget) осуществляет присвоение принимаемого значения полю HUDWidget.

Событие EventBeginPlay() вызывается при запуске игрового уровня GameMap и осуществляет создание объекта класса PlayerUI, выполняющего функцию пользовательского интерфейса. После создания, поле ScoreInt данного объекта, инициализируется значением 0. В поле HUDWidget класса BP\_PlayerHUD сохраняется ссылка на созданный объект класса PlayerUI, посредством метода SetHUDWidget(Widget). AddToViewport(Widget) производит вывод созданного объекта на экран пользователя.

Событие EventIncrementScore() вызывается в коде логики игрового уровня. Данное событие получает ссылку на объект пользовательского интерфейса посредством метода GetHUDWidget(). Из полученного результата вызывается значение поля ScoreInt и осуществляется инкрементация текущего значения. Полученный результат записывается по той же ссылке на объект HUDWidget.

**3.4 Блок игровых объектов**

Блок игровых объектов хранит параметры, события и методы объектов, используемый в данном проекте. Данными объектами могут быть как игровой персонаж, так и другие объекты и материалы используемые в игре. Данный блок содержит классы Actor, Pawn, Character, BP\_PlayerCharacter, BP\_FloorPlatform, M\_Floor, M\_Wall.

**3.4.1 Actor**

Класс Actor является базовым классом для всех объектов в системе Unreal Engine 4. К данному классу могут относиться как статические, так и динамические объекты. Так же объекты класса Actor, могут не иметь никаких графических элементов, и представлять собой элементы кода, взаимодействующие с игровым окружением.

Класс Actor содержи следующие поля:

Поле canBeDamaged является булевой переменной, хранящей возможность данного объекта получать урон.

Поле AutoDestroyWhenFinished является булевой переменной, хранящей возможность данного объекта к самоуничтожению, по окончанию жизни.

Метод GetAllChildActors(Actor, Boolean) принимает указатель на объект класса, наследуемого от класса Actor. Осуществляет поиск всех существующих объектов всех классов, наследуемых от передаваемого указателя. Возвращает массив всех найденных объектов.

Метод GetParentActor(Actor) принимает объект и возвращает указатель на базовый объект класса Actor, от которого наследуется передаваемый.

Метод IsChildActor(Actor) принимает указатель на объект класса Actor, и осуществляет проверку, является ли данный объект дочерним. В качестве результата возвращает булево значение.

Метод GetComponentsByClass(Actor, ComponentClass) принимает указатель на объект класса Actor, а так же указатель на класс компоненты. Осуществляет поиск всех компонент, соответствующих классу компоненты образца. Результатом метода является массив всех найденных компонент.

Метод GetAllActorsOfClass(ActorsClass) осуществляет поиск всех существующих объектов относящихся к принимаемому классу. Поиск осуществляется в пределах иерархии активного игрового уровня. Результатом метода является массив всех найденных объектов.

**3.4.2 Pawn**

Класс Pawn является дочерним классом Actor и представляет собой любой объект с возможностью взятия контроля над ним посредством PlayerController.

Класс Pawn содержит следующие поля:

Поле PlayerState хранит в себе указатель на текущий объект класса PlayerState. Данный класс хранит в себе параметры состояния объекта.

Метод AddInputVector(Vector<Double>, Boolean) принимает направление относительно текущей позиции объекта и осуществляет толчок данного объекта в этом направлении. Значения вектора накапливаются в течение кадра и применяются в качестве ускорения при обновлении движения.

Метод GetPlayerState() возвращает указатель на текущий объект в поле PlayerState.

Метод SetPlayerState(PlayerState) принимает указатель на объект PlayerState и записывает это значение в поле класса.

Метод GetPawnOwner(PawnMovement) принимает компонент PawnMovement и возвращает указатель на объект класса Pawn, содержащий данную компоненту.

**3.4.3 Character**

Класс Character является дочерним классом Pawn. Данный класс представляет собой базовый класс с расширенным функционалом для управления объектом.

Класс Character содержит следующие поля:

Поле CharacterMovement содержит в себе компоненту, используемую для реализации передвижения объектов класса Character.

Поле CapsuleComponent сдержит в себе компоненту, выполняющую роль коллизии, и используемую для определения пересечения данной компоненты с другими объектами в игровом мире.

Поле Mesh содержит компоненту, которая является объектом класса StaticMesh, и представляет собой визуальное отображение объекта класса. Так же данная компонента содержит в себе данные о материале используемым данным объектом.

Метод GetBaseRotationOffset(Character) принимает указатель на объект класса Character, и возвращает базовое смещение компоненты вращения данного персонажа.

Метод GetCharacterOwner(CharacterMovement) принимает указатель на компоненту CharaterMovement, отвечающую за передвижение Charater. Результатом функции является объект типа Character, к которому данная компонента относится.

Метод CanJump(Character)принимает указатель на объект класса Character, осуществляет проверку возможности осуществления прыжка персонажа из данной позиции и возвращает результат проверки в формате булевого значения.

Метод AddImpulse(Character, Vector<Double>) принимает указатель на объект класса Character, а также направление относительно текущей позиции и осуществляет толчок данного персонажа в этом направлении. Значения вектора накапливаются в течение кадра и применяются в качестве ускорения при обновлении движения.

**3.4.4 BP\_PlayerCharacter**

Класс BP\_PlayerCharacter является дочерним классом Character. Данный класс описывает подконтрольного игроку персонажа и имеет функционал, посредством которого игрок взаимодействует с игровым миром.

Класс BP\_PlayerCharacter содержит следующие поля:

Поле Sphere хранит в себе объект класса StaticMeshComponent. Данная компонента хранит в себе сферическую модель, её параметры, а так же материал.

Поле CapsuleComponent сдержит в себе компоненту, выполняющую роль коллизии, и используемую для определения пересечения данной компоненты с другими объектами в игровом мире.

Поле canBeActivate является булевой переменной, отображающей возможность данного объекта взаимодействовать с другими объектами. Данное поле используется для валидации события InputActionInteract в классе BP\_PlayerController.

Поле OverlapedObject предназначено для хранения ссылки на текущий объект класса BP\_FloorPlatform, взаимодействующий с данным объектом класса BP\_PlayerCharacter.

Событие OnComponentBeginOverlap(CapsuleComponent) вызывается при условии, что компонента CapsuleComponent произвела пересечение с некоторым объектом в текущий момент времени. Данное событие производит проверку объекта, с которым произошло данное событие, на его соответствие классу BP\_FloorPlatform. В случае если результат данной операции является положительным, тогда полю OverlapedObject присваивается указатель на данный объект. Полю CanBeActivate присваивается значение true.

Событие OnComponentEndOverlap(CapsuleComponent) вызывается при условии, что компонента CapsuleComponent прекратила пересечение с объектом в текущий момент времени. Данное событие производит проверку объекта, с которым закончилось пересечение, на его соответствие классу BP\_FloorPlatform. В случае если результат данной операции является положительным, тогда полю OverlapedObject присваивается значение None. Полю CanBeActivate присваивается значение false.

**3.4.5 BP\_FloorPlatform**

Класс BP\_FloorPlatform наследуется от класса Pawn и представляет собой интерактивную платформу, с которой будет взаимодействовать игрок.

Данный класс содержит следующие поля:

Поле BoxCollision содержит в себе компоненту коллизии, для настройки взаимодействия данного объекта с объектами других классов.

Поле isActivate? является булевой переменной для хранения состояния активации платформы.

Поле PlayerInZone? является булевой переменной для отображения нахождения игрока в зоне компоненты BoxCollision.

Поле M\_Floor содержит в себе указатель на класс материала данного объекта. Данное поле принимает объекты класса M\_Floor.

Событие EventBeginPlay() вызывается при создании объекта класса или запуске уровня. Данное событие создает материал динамического типа, посредством функции CreateDynamicMaterialInstance (PrimitiveComponent, SourceMaterial). Производится присвоение результирующего значения функции полю M\_Floor. Для компоненты BoxCollision, осуществляется выключение коллизии с помощью функции SetCollisionEnabled(ECollisionEnabled).

Событие OnComponentBeginOverlap (BoxCollision) вызывается при условии, что компонента BoxCollision произвела пересечение с некоторым объектом в текущий момент времени. Данное событие производит проверку объекта, с которым произошло данное событие, на его соответствие классу BP\_PlayerCharacter. В случае если результат данной операции является положительным, тогда полю PlayerInZone? присваивается значение true.

Событие OnComponentEndOverlap(BoxCollision) вызывается при условии, что компонента BoxCollision прекратила пересечение с объектом в текущий момент времени. Данное событие производит проверку объекта, с которым закончилось пересечение, на его соответствие классу BP\_PlayerCharacter. В случае если результат данной операции является положительным, тогда полю PlayerInZone? присваивается значение false.

Событие IsActivateEvent() является пользовательским событием, вызываемым в блоке игровой логики. Данное событие устанавливает значение true полю isActivate?. Далее производится присвоение значения 1.f для параметра IsActivate?, хранящегося в объекте материала M\_Floor. После смены параметра у материала производится активация коллизии у компоненты BoxCollision, после чего запускается задержка данного состояния на значение равное четырём тактам музыкальной композиции. Данное значение получается в результате умножения поля DelayTime на 4. По окончанию задержки вызывается событие DeActivateEvent().

Событие DeActivateEvent() является пользовательским событием, вызываемым в блоке игровой логики. Данное событие устанавливает значение false полю isActivate?. Далее производится присвоение значения 0.f для параметра IsActivate?, хранящегося в объекте материала M\_Floor. После смены параметра у материала производится деактивация коллизии у компоненты BoxCollision, во избежание ситуаций ложного срабатывания деактивированного объекта.

**3.4.6 M\_Floor**

Класс M\_Floor представляет собой компоненту материала, применяемого к объекту класса BP\_FloorPlatform.

Данный класс содержит следующие поля:

Поле WhiteColor хранит константное значение Vector<Float>, содержащее параметры белого цвета в RGBA формате. Поле по умолчанию инициализировано значениями (1, 1, 1, 1).

Поле BlueColor хранит константное значение Vector<Float>, содержащее параметры синего цвета в RGBA формате. Поле по умолчанию инициализировано значениями (0.01, 0.01, 1, 1).

Поле isActivate? представляет собой параметр для скалярной величины. Данное поле используется для установления состояния объекта, а так же смены цвета материала.

Функция Lerp(Vector<Float>, Vector<Float>, Float) осуществляет математическую операцию интерполяции, принимая значения двух векторов и альфа-значение. Результатом данной функции является вектор с параметрами нового цвета.

Событие EventConstruct() инициализирует базовые параметры класса. Данное событие вызывается при создании экземпляра класса.

**3.4.7 M\_Wall**

Класс M\_Wall представляет собой компоненту материала, применяемого к объектам стен.

Данный класс содержит следующее поле:

Поле Param хранит константное значение Vector<Float>, содержащее параметры белого цвета в RGBA формате. Поле по умолчанию инициализировано значениями (0.225, 0.225, 0, 1).

**3.4.8 M\_Player**

Класс M\_Player представляет собой компоненту материала, применяемого к компоненте SphereComponent для подконтрольного игрового персонажа.

Данный класс содержит следующее поле:

Поле Param хранит константное значение Vector<Float>, содержащее параметры белого цвета в RGBA формате. Поле по умолчанию инициализировано значениями (1, 1, 1, 1).

**3.5 Блок игрового меню**

Данный блок реализован с использованием базовых инструментов Unreal Engine 4, для создания графических интерфейсов. Логика, описывающая функционал интерфейса описана в классах: UserWidget, MainMenu, FileSelectorMenu, OptionsMenu, HelpMenu.

**3.5.1 UserWidget**

Базовый класс UserWidget включает в себя следующие поля:

Поле ShowMouseCursor является булевой переменной, отвечающей за отображение курсора мыши при использовании графического интерфейса.

Поле IsEnabled является булевой переменной, отвечающей за активное состояние элемента пользовательского интерфейса.

Поле ESlateVisibility представляет собой набор состояний видимости для элементов пользовательского интерфейса. Имеет набор предустановленных константных значений.

Метод SetVisibility(ESlateVisibility) принимает и переводит элемент интерфейса в переданное состояние.

Метод GetVisibility() возвращает активное состояние видимости для текущего элемента пользовательского интерфейса.

Метод SetIsEnabled(Boolean) инициализирует переменную IsEnabled, переданным булевым значением.

Метод GetIsEnabled() возвращает активное булево состояние для текущего элемента пользовательского интерфейса.

Метод RemoveFromParent(Widget) принимает указатель на объект типа Widget. Значение по умолчанию указывает на собственный экземпляр объекта. Задачей метода является удаление всех активных компонентов, связанных с данным объектом. Данный метод является ключевым для реализации перехода между несколькими меню приложения.

Метод CreateWidget(UserWidget, PlayerController) принимает класс, наследуемый от базового класса Widget, а так же указатель на объект класса, наследуемого от базового класса PlayerController. Данный метод создаёт экземпляр принимаемого класса и возвращает указатель на данный объект. Данный метод так же является ключевым для реализации перехода между несколькими меню приложения.

Заключительным методом для реализации перехода между несколькими меню приложения, является AddToViewport(UserWidget). Данный метод принимает ссылку на объект класса UserWidget и выводит его на активное рабочее пространство приложения.

**3.5.2 MainMenu**

Класс MainMenu наследуется от базового класса UserWidget, и содержит в себе поля, методы и события посредством которых игрок будет осуществлять взаимодействие с главным меню приложения. Производный класс MainMenu включает в себя следующие поля:

Поля Play\_Btn, Settings\_Btn, Help\_Btn и Quit\_Btn хранят в себе объекты класса Button, а так же размеры и цвета данных кнопок.

Поле Background является объектом класса Image и хранит в себе ссылку на исходное изображение, а так же его параметры: размер, прозрачность, положение на экране, приоритет.

Событие EventConstruct() вызывается при создании экземпляра класса MainMenu и осуществляет инициализацию всех полей класса. Данное событие вызывает следующие методы:

Метод SetInputModeUIOnly(PlayerController, Widget, EMouseLockMode) принимает объект класса, наследуемого от базового класса PlayerController, и устанавливает считывание принимаемых сигналов контроллера в режим «только для пользовательского интерфейса».

Функция GetPlayerController(Integer) возвращает указатель на объект контроллера игрока PlayerController, после чего полю ShowMouseCursor присваивается значение true.

Метод GetGameUserSetting() возвращает указатель на объект класса GameUserSettings необходимый для последующего изменения настроек приложения.

Метод LoadSettings(GameUserSettings, Boolean) осуществляет загрузку принимаемых параметров приложения GameUserSettings в постоянное хранилище приложения. Если входное поле Boolean равняется true, то загрузка осуществляется в принудительном режиме.

Событие OnClicked(Play\_Btn) вызывается при нажатии на кнопку Play\_Btn в главном меню игры. Данное событие вызывает выполнение таких методов как RemoveFromParent(Widget), CreateWidget (UserWidget, PlayerController), AddToViewport (UserWidget). Посредством данных методов осуществляется удаление текущего окна, создание и отображение меню выбора файла FileSelectorMenu.

Событие OnClicked(Settings\_Btn) вызывается при нажатии на кнопку Settings\_Btn в главном меню игры. Данное событие вызывает выполнение таких методов как RemoveFromParent(Widget), CreateWidget (UserWidget, PlayerController), AddToViewport (UserWidget). Посредством данных методов осуществляется удаление текущего окна, создание и отображение меню настроек приложения OptionsMenu.

Событие OnClicked(Help\_Btn) вызывается при нажатии на кнопку Help \_Btn в главном меню игры. Данное событие вызывает выполнение таких методов как RemoveFromParent(Widget), CreateWidget (UserWidget, PlayerController), AddToViewport (UserWidget). Посредством данных методов осуществляется удаление текущего окна, создание и отображение меню управления для приложения HelpMenu.

Событие OnClicked(Quit\_Btn) вызывается при нажатии на кнопку Quit\_Btn в главном меню игры. Данное событие вызывает выполнение метода QuitGame(PlayerController, EQuitPreference, Boolean) осуществляющего корректное закрытие приложения.

**3.5.3 FileSelectorMenu**

Класс FileSelectorMenu наследуется от базового класса UserWidget, и содержит в себе поля, методы и события посредством которых игрок будет осуществлять взаимодействие с меню выбора аудиофайла, для дальнейшего взаимодействия с данным файлом в игровом процессе непосредственно. Производный класс FileSelectorMenu включает в себя следующие поля:

Поля Apply\_Btn, File\_Btn и Back\_Btn хранят в себе объекты класса Button, а так же размеры и цвета данных кнопок.

Поле Background является объектом класса Image и хранит в себе ссылку на исходное изображение, а так же его параметры: размер, прозрачность, положение на экране, приоритет.

Поле filename предназначено для хранения полученного названия исполняемого аудиофайла.

Поля wavHeader1 и wavHeader2 предназначены для хранения получаемой информации из заголовка аудиофайла, а также для вывода данной информации на экран.

Функция GetFilename() осуществляет отображение поля Filename в меню выбора файла FileSelectorMenu.

Функция GetWavHeader1Text() осуществляет отображение поля wavHeader1 в меню выбора файла FileSelectorMenu.

Функция GetWavHeader2Text() осуществляет отображение поля wavHeader2 в меню выбора файла FileSelectorMenu.

Событие OnClicked(Apply\_Btn) вызывается при нажатии на кнопку Apply\_Btn в меню выбора аудиофайла. Данное событие, при условии, что поле isEnabled для данной кнопки не ровняется false, вызывает выполнение таких методов как OpenLevelByName (LevelName, Boolean, String), SetShowMouseCursor (Boolean, PlayerController), SetInputModeGameOnly (PlayerController). Посредством данных методов осуществляется открытие игрового уровня, отключение отображения курсора мыши и установление считывания принимаемых сигналов контроллера в режим «только для игры».

Событие OnClicked(File\_Btn) вызывается при нажатии на кнопку File\_Btn меню выбора аудиофайла. Данное событие вызывает выполнение следующих методов:

Метод OpenFileDialog(String, String, String, String, EEasyFileDialogFlags) осуществляет открытие диалогового окна для выбора аудиофайла и возвращает имя выбранного пользователем аудиофайла. Данный метод принимает следующие параметры: имя диалогового окна, путь по умолчанию, имя аудиофайла по умолчанию, тип поддерживаемых аудиофайлов, количество одновременно открываемых аудиофайлов. В качестве пути по умолчанию принимается результат функции GetProjectDirectory(). После выполнения данного метода осуществляется проверка на успешность получения имени аудиофайла, выбранного пользователем. В случае если метод возвращает значение false, в поле wavHeader1 передается следующее значение «File is not selected.».

Метод GetFileHeader(String), осуществляет открытие выбранного аудиофайла и возвращает информацию, хранящуюся в его заголовке. В качестве входных параметров, принимает имя аудиофайла. В результате данного метода происходит запись полученных данных в поля filename, wavHeader1 и wavHeader2.

Метод ReadWAVFile(String) так же принимает имя аудиофайла и осуществляет вычисление его ритма. Результатом данного метода является вычисленный музыкальный ритм аудиофайла, представленный в формате Integer, а так же булева переменная, указывающая на успешность совершенной операции.

Событие OnClicked(Back\_Btn) вызывается при нажатии на кнопку Back\_Btn в меню выбора файла. Данное событие вызывает выполнение таких методов как RemoveFromParent(Widget), CreateWidget (UserWidget, PlayerController), AddToViewport (UserWidget). Посредством данных методов осуществляется создание и отображение главного меню приложения MainMenu.

**3.5.4 OptionsMenu**

Класс OptionsMenu наследуется от базового класса UserWidget, и содержит в себе поля, методы и события посредством которых игрок будет осуществлять изменение ключевых графических параметров приложения. Производный класс OptionsMenu включает в себя следующие поля:

Поля, хранящие в себе объекты класса Button, а так же размеры и цвета данных кнопок: WindowMode+, WindowMode-, Resolution+, Resolution-, VSync+, VSync-, Apply\_Btn и Back\_Btn.

Поле Background является объектом класса Image и хранит в себе ссылку на исходное изображение, а так же его параметры: размер, прозрачность, положение на экране, приоритет.

Поле windowmode типа EWindowMode хранит одно из трёх состояний оконного режима приложения. По умолчанию установлено значение «Fullscreen».

Поле resolutionIndex типа Integer хранит номер одного из типов разрешений, доступных к выбору пользователя. По умолчанию установлено значение 0.

Поле resolution типа IntPoint хранит соотношение разрешение изображения, представленное в виде соотношения пикселей по X и Y. По умолчанию установлено значение 1920x1080.

Поле vsync? типа Boolean хранит значение активации вертикальной синхронизации изображения. По умолчанию установлено значение false.

Функция GetWindowMode() осуществляет отображение поля windowmode в меню настроек приложения OptionsMenu.

Функция GetResolution() осуществляет отображение поля resolution в меню настроек приложения OptionsMenu.

Функция GetVSync() осуществляет отображение поля vsync? в меню настроек приложения OptionsMenu.

Событие EventConstruct() вызывается при создании экземпляра класса OptionsMenu и осуществляет инициализацию всех полей класса. Данное событие вызывает следующие методы:

Метод GetGameUserSetting() возвращает указатель на объект класса GameUserSettings необходимый для последующего изменения настроек приложения.

Метод LoadSettings(GameUserSettings, Boolean) осуществляет загрузку принимаемых параметров приложения GameUserSettings в постоянное хранилище приложения. Если входное поле Boolean равняется true, то загрузка осуществляется в принудительном режиме.

По окончанию данных методов инициализируются следующие поля класса: windowmode, vsync, resolution, resolutionIndex. Для инициализации данных полей используются следующие функции:

Функция GetFullscreenMode(GameUserSettings), возвращает текущий оконный режим приложения.

Функция IsVSyncEnabled(GameUserSettings), возвращает текущее состояние активации вертикально синхронизации изображения приложения.

Функция GetScreenResolution (GameUserSettings) возвращает текущее разрешение изображения приложения.

Событие OnClicked(WindowMode+) вызывается при нажатии на «>» кнопку в строке «Window mode». Данное событие осуществляет приведение типа EWindowMode к типу Integer и инкрементирует данное значение, при условии, что оно остается в диапазоне от 0 до 2.

Событие OnClicked(WindowMode-) вызывается при нажатии на «<» кнопку в строке «Window mode». Данное событие осуществляет приведение типа EWindowMode к типу Integer и декрементирует данное значение, при условии, что оно остается в диапазоне от 0 до 2.

Событие OnClicked(Resolution+) вызывается при нажатии на «>» кнопку в строке «Resolution». Данное событие инкрементирует значение поля resolutionIndex, при условии, что оно остается в диапазоне от 0 до 3. Исходя из значения индекса, осуществляется изменение поля resolution.

Событие OnClicked(Resolution-) вызывается при нажатии на «<» кнопку в строке «Resolution». Данное событие декрементирует значение поля resolutionIndex, при условии, что оно остается в диапазоне от 0 до 3. Исходя из значения индекса, осуществляется изменение поля resolution.

Событие OnClicked(VSync+) вызывается при нажатии на «>» кнопку в строке «VSync». Данное событие присваивает полю vsync? значение true.

Событие OnClicked(VSync-) вызывается при нажатии на «<» кнопку в строке «VSync». Данное событие присваивает полю vsync? значение false.

Событие OnClicked(Apply\_Btn) вызывается при нажатии на кнопку Apply\_Btn в меню настроек приложения. Данное событие вызывает следующие методы:

Метод GetGameUserSetting() возвращает указатель на объект класса GameUserSettings необходимый для последующего изменения настроек приложения.

Метод SetFullscreenMode (GameUserSettings, EWindowMode) принимает объект GameUserSettings и значение поля windowmode, и устанавливает данное значение для параметра оконного режима приложения.

Метод SetScreenResolution (GameUserSettings, IntPoint) принимает объект GameUserSettings и значение поля resolution, и устанавливает данное значение для параметра разрешения изображения приложения.

Метод SetVSyncEnabled(GameUserSettings, Boolean) принимает объект GameUserSettings и значение поля vsync?, и устанавливает данное значение для параметра активации вертикальной синхронизации изображения приложения.

Метод ApplySettings(GameUserSettings, Boolean) принимает объект GameUserSettings и применяет новые настройки для приложения.

Событие OnClicked(Back\_Btn) вызывается при нажатии на кнопку Back\_Btn в меню настроек приложения. Данное событие вызывает выполнение таких методов как RemoveFromParent(Widget), CreateWidget (UserWidget, PlayerController), AddToViewport (UserWidget). Посредством данных методов удаление окна текущего меню, осуществляется создание и отображение главного меню приложения MainMenu.

**3.5.5 HelpMenu**

Класс HelpMenu наследуется от базового класса UserWidget, и содержит в себе поля, методы и события посредством которых игрок будет осуществлять взаимодействие с меню помощи управления. Производный класс HelpMenu включает в себя следующие поля:

Поле Back\_Btn хранит в себе объект класса Button, а так же размеры и цвет данной кнопки.

Поля Background является объектом класса Image и хранит в себе ссылку на исходное изображение, а так же его параметры: размер, прозрачность, положение на экране, приоритет.

Поля Image\_MF1, Image\_MF2, Image\_MF3, Image\_MB1, Image\_MB2, Image\_MB3, Image\_ML1, Image\_ML2, Image\_ML3, Image\_MR1, Image\_MR2, Image\_MR3, Image\_I1, Image\_I2, Image\_I3 также являются объектами класса Image. Они также хранят в себе ссылку на исходные изображения и их параметры. Данные поля используются для отображения привязанных клавиш и других органов управления, связанных с определёнными действиями в игре.

Событие OnClicked(Back\_Btn) вызывается при нажатии на кнопку Back\_Btn в меню помощи приложения. Данное событие вызывает выполнение таких методов как RemoveFromParent(Widget), CreateWidget (UserWidget, PlayerController), AddToViewport (UserWidget). Посредством данных методов удаление окна текущего меню, осуществляется создание и отображение главного меню приложения MainMenu.

**3.6 Алгоритм чтения аудиофайла**

Данный алгоритм, является частью подключаемого плагина для работы и анализа аудио файла. К данному алгоритму относятся следующие классы: WAV\_Header, WAV\_Reader.

**3.6.1 WAV\_Header**

В классе WAV\_Header задана структура WAVHEADER содержащая следующие поля:

– chunkId поле содержит в себе информацию о начале RIFF-цепочки.

– chunkSize хранит в себе размер последующей RIFF-цепочки.

– format содержит в себе символьное краткое название формата файла.

– subchunk1Id содержит символьный идентификатор подцепочки.

– subchunk1Size поле хранит размер подцепочки, начиная с этой позиции.

– audioFormat содержит номер аудиоформата файла.

– numChannels хранит в себе количество каналов для воспроизведения аудиофайла.

– sampleRate частота дискретизации аудиосигнала.

– byteRate количество байт, передаваемых за секунду воспроизведения.

– blockAlign количество байт для одного сэмпла, включая все каналы.

– bitsPerSample количество бит, необходимых для хранения одного сэмпла.

– subchunk2Id содержит символьный идентификатор подцепочки

– subchunk2Size поле хранит размер подцепочки, начиная с этой позиции.

– subchunk2ListType содержит тип хранящейся информации о лицензии аудиофайла.

– subchunk2Text хранит информацию о лицензии аудиофайла.

Метод GetHeader(String) принимает имя аудиофайла и функцией fopen\_s(FILE, string, string) осуществляет его открытие в режиме чтения «rb». Таким образом, производится чтение участка, хранящего заголовок данного аудиофайла. Размер читаемого участка равняется сумме размеров всех полей класса. За размер каждого поля берётся максимальное требуемое значения типа данных каждого поля. По окончанию чтения осуществляется закрытие файла. Метод возвращает структуру WAVHEADER, хранящий в себе всю информацию аудиофайла, необходимую для корректного чтения блока данных.

Метод PrintHeader(FString, FString) принимает указатели на два поля типа FString. Далее метод осуществляет приведение типов для всех полей данного класса к типу данных FString. Так же, на основе имеющейся в полях информации, вычисляется продолжительность композиции, которая приводится к типу данных FString. Полученная в результате приведения типов информация разбивается на две части и сохраняется в полученные методом поля. Это позволяет вывести пользователю полную информацию об аудиофайле.

**3.6.2 WAV\_Reader**

В классе WAV\_Reader содержаться следующие статические поля для работы с файлами:

Поля sampleDataL и sampleDataR являются векторами для комплексных значений типа Double. Данные поля предназначены для преобразованных амплитудно-частотных значений аудиоданных, полученных в результате обработки исходного сигнала.

Поле DSIZE хранит в себе размер массивов данных, с которыми происходит работа в данных алгоритмах.

Метод GetAmplitudeValues(Vector<Complex<Double>>, Integer) принимает вектор комплексных значений Double, представляющих из себя преобразованные амплитудно-частотные значения аудиоданных, а также значение типа Integer, хранящее частоту дискретизации читаемого аудиосигнала. Данный метод осуществляет перебор значений входного вектора и вычисление модуля каждого значения. Модуль числа записывается в новый вектор Double, посредством функции push\_back(). Полученный вектор Double является результирующей данного метода.

Метод PushBackZeros(Vector< Complex<Double>>, Integer) принимает вектор комплексных значений Double, представляющих из себя исходные амплитудно-временные значения аудиоданных, а также значение типа Integer, хранящее требуемый размер вышеуказанного вектора. Данный метод осуществляет подготовку данного вектора к осуществлению операции быстрого преобразования Фурье. В методе происходит осуществление добавления нулевых значений в конец вектора, посредством функции push\_back(). Результатом метода является вектор комплексных значений типа Double, размер которого является кратным степени 2.

Функция FFT(Vector<Complex<Double>>) принимает вектор комплексных амплитудно-временных значений типа Double и преобразовывает значения этого вектора к амплитудно-частотному представлению. В данной функции значения входного вектора разбиваются на два вектора, хранящих значения четных и нечётных индексов. После этого полученные вектора отправляются в рекурсивные версии данной функции, до тех пор, пока размер входного вектора не станет равен единице. По достижению минимальных размеров векторов, осуществляется операция данного преобразования, в соответствии с формулой. Результаты вычисления перезаписывают исходные значения входного вектора комплексных значений Double.

Метод AmplitudeBlockProc(Vector<Double>, Vector< Double>) принимает вектора для значений левого и правого каналов. Данный метод перезаписывает значения входные векторов в локальные вектора комплексных значений типа Double, с соблюдением размерности вектора, равного DSIZE значений. После осуществления перезаписи, данные вектора поочерёдно преобразовываются с помощью рекурсивной функции FFT(Vector<Complex<Double>>). Обработанные амплитудно-частотные значения отправляются в качестве входных значений для метода GetAmplitudeValues(Vector<Complex<Double>>, Integer).

Метод ReadFromFile(String, Vector<Vector<Double>>, Vector<Vector<Double>>) принимает в качестве входных параметров имя файла, а так же 2 указателя на сегменты памяти, для хранения прочитанных данных, для левого канала и правого канала соответственно.

Первым делом метод осуществляет открытие аудиофайла, посредством функции fopen\_s(FILE, string, string), в режиме чтения «rb». После открытия файла, функцией fseek(FILE, Long, Integer), осуществляется перемещение указателя файла, на начало блока данных. Производится очистка входных векторов для данных, посредством функции clear().

После того, как файл был подготовлен к последующему чтению, запускается цикл while, получающий данные посредством функции fread(Void, Size\_t, Size\_t, FILE). Буфер для чтения данных проверяется на наполненность, после чего данные из него разбиваются на левый и правый канал. После разбиения, они записываются в вектора комплексных значений Double посредством функции push\_back(). Следующей задачей цикла является проверка совпадения размера заполняемого вектора, с одной десятой от частоты дискретизации. Таким образом, вычисляется набор аудиоданных, отвечающий на 0.1 секунды музыкальной композиции. В случае если условие истинно, производится вызов метода AmplitudeBlockProc(Vector<Double>, Vector< Double>), результаты которого записываются во входящие поля для прочитанных данных и продолжается выполнение чтения файла. По окончанию чтения, файл закрывается с помощью функции fclose(FILE). Результатом корректного выполнения метода является значение true.

**3.7 Алгоритм анализа аудиофайла**

Данный алгоритм также является частью подключаемого плагина для работы и анализа аудио файла. К данному алгоритму относится класс BPM\_Detector.

**3.7.1 BPM\_Detector**

В классе BPM\_Detector инициализированы следующие методы:

Метод GetAverageSum(Vector<Double>) осуществляет суммирование всех значений входного вектора. Результатом метода является среднее арифметическое данного вектора.

Метод max\_element(Vector<Double>, Integer, Integer) принимает вектор Double, и два значения типа Integer, определяющие диапазон работы метода. Данный метод осуществляет перебор значений из вектора и вычисление индекса максимального значения. Результатом метода является максимальный элемент вектора.

Метод min\_element(Vector<Double>, Integer, Integer) принимает вектор Double, и два значения типа Integer, определяющие диапазон работы метода. Данный метод осуществляет перебор значений из вектора и вычисление индекса минимального значения. Результатом метода является минимальный элемент вектора.

Метод FindMinMaxPeaks(Vector<Vector<Double>>, Vector <Double>, Vector<Double>) принимает вычисленные вектора амплитудных значений, взятые с интервалом 0.1 секунды, а также указатель на вектор максимальных и минимальных амплитудных значений. Метод осуществляет вычисление максимальных и минимальных амплитудных значений в каждом указанном временном диапазоне, и записывает результаты в конец входных векторов maxPeaksVec и minPeaksVec, посредством функции push\_back().

Метод FindBassPeaks(Vector<Double>, Double, Double, Double) принимает следующие значения: вектор максимальных амплитудных значений maxPeaksVec, пороговое значение threshold, среднее арифметическое вектора averageSum, центральное значение рабочего диапазона center. Перед выполнением метода происходит инициализация локальных переменных. Метод осуществляет перебор всех значений входящего вектора, осуществляется сравнение каждого значения вектора, с входным полем threshold. В случае если значение превышает значение, хранящееся в поле threshold, осуществляется следующий ряд проверок данного значения вектора. Если значение входного поля averageSum больше, чем значение входного поля center, выполняется сравнение текущего значения вектора с двумя соседними значениями этого же вектора. В случае если значение входного поля averageSum меньше, чем значение входного поля center, сравнение текущего значения вектора, осуществляется с четырьмя соседними значениями данного вектора. Данный подход позволяет более точно определять амплитудные пиковые значения в композициях с менее активным музыкальным ритмом. В результате данных проверок вычисляется момент пикового амплитудного значения, и производится инкрементация счётчика numValuesAfterThreshold. Так же на данном этапе совершается вычисление момента первого и последнего пика для музыкальной композиции. Результатом метода является деление счетчика numValuesAfterThreshold на разницу времени между первым и последним пиком музыкальной композиции, и умножение полученного результата на 600.

Метод Detect\_BPM(Vector<Vector<Double>>, Vector< Vector<Double>>) принимает вектора амплитудных значений для левого и правого канала аудиосигнала. Изначально метод дважды вызывает метод FindMinMaxPeaks(Vector<Vector<Double>>, Vector<Double>, Vector<Double>) поочерёдно передовая вектора левого и правого канала. После этого осуществляется вычисление значений center и threshold для каждого канала по отдельности. Также вычисляются значения averageSum посредством метода GetAverageSum(Vector<Double>). В окончании метода осуществляется вычисление ритма данной музыкальной композиции посредством метода FindBassPeaks(Vector<Double>, Double, Double, Double), принимающего в качестве входных параметров все вышеописанные значения. Результатом метода Detect\_BPM (Vector<Vector<Double>>, Vector<Vector <Double>>)) является количество пиковых амплитудных значений за минуту времени.

1. **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В данном разделе представлено несколько ключевых алгоритмов программного средства, а так же приведено описание нескольких ключевых представлений.

* 1. **Представления**

В работе данного программного средства учувствует большое количество системных представлений Unreal Engine 4. Для описания представлений в данном разделе были выбраны лишь ключевые из системных представлений, а так же те представления, которые были разработаны для корректной работы финального продукта.

* + 1. **Заголовок аудиофайла**

Заголовок аудиофайла хранит в себе всю необходимую информацию для корректного чтения файла, а так же, может хранить корпоративную информацию о лицензии данной композиции.

Исходный код представления:

struct WAVHEADER {

char chunkId[4];

unsigned long chunkSize;

char format[4];

char subchunk1Id[4];

unsigned long subchunk1Size;

unsigned short audioFormat;

unsigned short numChannels;

unsigned long sampleRate;

unsigned long byteRate;

unsigned short blockAlign;

unsigned short bitsPerSample;

char subchunk2Id[4];

unsigned long subchunk2Size;

char subchunk2Text[4];

};

Представление принимает в себя набор параметров, в порядке, соответствующем формату RIFF-заголовка. Массив char chunkId[4] предназначен для хранения формата заголовка в форме четырёх символов «RIFF» в ASCII кодировке.

Поле unsigned long chunkSize хранит в себе размер последующей RIFF-цепочки. В данную цепочку входит размер заголовочного блока, а так же размер блока данных, хранящий аудиосигнал файла.

Массив char format[4] содержит в себе краткое название формата файла в ASCII кодировке. В случае данного проекта, все аудиофайлы, с которыми будут осуществляться взаимодействия, соответствуют формату .wav, или же «WAVE».

Массив char subchunk1Id[4] получает информацию о названии последующей подцепочки аудиофайла. В данном проекте все файлы имеют схожий идентификатор подцепочки – «fmt». Данное обозначение символизирует о том, что в последующем блоке данных, будет описан формат звуковых данных: количество каналов, частота дискретизации, количество бит в каждом сэмпле и так далее.

Последующее поле unsigned long subchunk1Size получает из заголовка аудиофайла размер данной подцепочки. Исходя из данной информации, высчитывается положение начала следующей подцепочки.

Поле unsigned short audioFormat хранит число, указывающее на формат сжатия аудиофайла. В данном проекте все аудиозаписи были созданы, без какого либо специфического сжатия. В связи с этим в данном поле будет храниться значение 1, символизирующее линейному квантованию аудиосигнала.

Следующее поле unsigned short numChannels хранит в себе количество каналов для воспроизведения аудиофайла. Если значение равняется «1», то аудиосигнал хранится в формате моно сигнала. Если же значение поля равно «2», то значения в блоке данных записываются в формате стерео сигнала. Из этого следует, что при чтении блока данных, каждое первое значение, будет соответствовать левому каналу, а каждое второе – правому.

Значение поля unsigned long sampleRate является ключевым для реализации алгоритмов анализа аудиосигнала. Данное поле хранит в себе частоту дискретизации сигнала – что соответствует количеству точек отсчета непрерывного по времени сигнала, при его дискретизации аналогово–цифровым преобразователем. Данное число является количеством значений, содержащих в себе амплитудно-временную информацию, за одну секунду воспроизведения аудиосигнала.

Поле unsigned long byteRate указывает на скорость потока данных в байтах в секунду. Данное поле используется для оценки объёма памяти, необходимого для воспроизведения аудиофайла. Вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (4.1) |

где  – это частота дискредитации аудиосигнала,

 – количество звуковых каналов(1 для моно, 2 для стерео),

 – количество бит, используемых для каждого сэмпла.

Таким образом, поле byteRate отражает количество байтов данных, которые необходимо обработать за секунду для воспроизведения аудиофайла. Данное поле является крайне важным для чтения и воспроизведения WAV-файлов в реальном времени.

Следующее поле unsigned short blockAlign содержит информацию о количестве байт для одного сэмпла, включая все каналы.

Поле unsigned short bitsPerSample содержит значение бит, используемых для каждого сэмпла. Также данное значение называют глубиной звучания аудиосигнала.

Массив char subchunk2Id[4] получает символьную информацию о названии второй подцепочки аудиофайла, представленную в ASCII кодировке. Аудиофайлы могут иметь несколько вариаций значения данного поля. В случае если идентификатор подцепочки равен «LIST», то в последующем блоке подцепочки будут описываться размер подцепочки, а так же массив subchunk2Text содержащий в себе корпоративную информацию о лицензии данного аудиофайла. В другом случае, идентификатор подцепочки хранит в себе значение «data». Последующий блок хранит лишь размер блока данных.

Поле unsigned long subchunk2Size содержит размер последующей подцепочки.

Массив char subchunk2Text[4] предназначен для извлечения корпоративной информации, в том случае, если идентификатор подцепочки, является равным «LIST». Данная информация не используется в дипломном проекте, но использование инициализация данного поля, необходима для корректной работы алгоритма чтения аудиофайла.

**4.1.2 Игровой режим**

Игровой режим представляет собой некоторую конфигурацию настроек проекта, требуемых к последующему исполнению данным приложением. Данные конфигурации привязываются к глобальным настройкам игровых уровней и вступают в силу после перехода игрока на данный уровень.

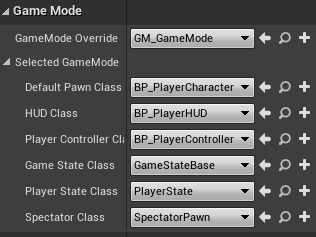


Рисунок 4.1 – Конфигурация GM\_GameMode

Рассмотрим конфигурацию представления GM\_GameMode (рисунок 4.1):

Поле DefaultPawnClass по умолчанию инициализируется ссылкой на объект класса BP\_PlayerCharacter. Данное поле принимает любые классы, наследуемые от базового класса Pawn и имеющие одну из компонент Movement. Наличие данной компоненты и обеспечивает игроку возможность управления данным объектом. Объект указанного класса будет являться подконтрольным игроку, и появляться на игровом уровне с помощью объекта PlayerStart, который также использует значения полей из данного представления.

Поле HUDClass также инициализируется по умолчанию, но ссылкой на объект класса BP\_PlayerHUD. Этот класс играет ключевую роль в контроле и отображении количества очков, которые игрок набрал во время игры. Это обеспечивает непрерывную обратную связь для игрока о его прогрессе.

Поле PlayerControllerClass по умолчанию инициализируется ссылкой на объект класса BP\_PlayerController. Это означает, что управление персонажем, указанным в поле DefaultPawnClass, будет связано с этим инициализированным классом. Это обеспечивает плавное и интуитивно понятное управление любым подконтрольным игроку персонажем или объектом.

Поле GameStateClass содержит объект класса GameState, который служит для хранения уникальных аспектов игрового мира, а также некоторых данных каждого отдельного пользователя в сетевых игровых сессиях. Класс используемый для хранения уникальных аспектов игрового мира игры хранит различного рода данные о состоянии данного игрового режима, прогрессе игроков и многое другое. По умолчанию это поле инициализируется объектом базового класса GameState, что соответствует одиночному режиму игры. Это обеспечивает сохранение состояния игры и поддержание согласованности игрового мира.

Поле PlayerStateClass содержит объект класса PlayerState, который служит для хранения некоторых аспектов состояний и параметров управляемого персонажа или пешки. По умолчанию это поле инициализируется объектом базового класса PlayerState, что соответствует отсутствию начальных состояний персонажа. Это обеспечивает гибкость в управлении состоянием персонажа.

Наконец, поле SpectatorClass содержит объект класса Pawn, который предназначен для реализации возможности наблюдения за игровым процессом со стороны. Это поле часто используется при разработке сетевых игр. По умолчанию это поле инициализируется объектом базового класса Pawn, что равняется отсутствию наблюдателя. Это обеспечивает возможность наблюдения за игровым процессом, что может быть полезно для разработчиков и игроков.

* 1. **Алгоритмы**

Некоторые из представленных ниже алгоритмов в исходном коде представляют сразу несколько методов или функций. Это обусловлено тем, что для улучшения читаемости исходного кода, крупные методы разбиваются на несколько методов меньшего размера, где каждый из небольших методов выполняет конкретную поставленную задачу.

* + 1. **Алгоритм чтения заголовка аудиофайла**

Целью данного алгоритма является чтение заголовка аудиофайла, для получения ключевой информации, для последующего корректного чтения файла, а так же анализа, для вычисления музыкального ритма.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Объявить переменную file типа FILE.

Шаг 3. Объявить переменную err типа errno\_t.

Шаг 4. Инициализировать переменную formattedFilename типа FString значением, полученным в результате метода filename.c\_str().

Шаг 5. Открыть файл с использованием функции fopen\_s(&file, filename.c\_str(), “rb”) и присвоить результат открытия переменной err.

Шаг 6. Если значение err равно NULL, то переход к шагу 10. Иначе перейти к шагу 7.

Шаг 7. Прочитать данные из файла в статическое поле header с помощью функции fread(&header, sizeof(WAVHEADER)), sizeof(WAVHEADER), 1, file).

Шаг 8. Закрыть файл с помощью функции fclose(file).

Шаг 9. Вернуть данные параметра header.

Шаг 10. Конец алгоритма.

* + 1. **Алгоритм чтения аудиофайла**

Целью данного алгоритма является открытие аудиофайла в режиме чтения, считывание данных в зависимости от количества каналов аудиофайла, произведения операции быстрого преобразования Фурье с фрагментами данных, запись результатов в вектора типа данных double и возвращение данных векторов по окончанию алгоритма, для последующего анализа.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Объявить переменную file типа FILE.

Шаг 3. Объявить переменную err типа errno\_t.

Шаг 4. Инициализировать переменную sampleRate значением header.sampleRate.

Шаг 5. Инициализировать переменную formattedFilename значением, полученным в результате метода filename.c\_str().

Шаг 6. Открыть файл с использованием функции fopen\_s(&file, filename.c\_str(), “rb”) и присвоить результат открытия переменной err.

Шаг 7. Если значение err не равно NULL, то вернуть значение false и переход к шагу 43.

Шаг 8. Создать константу numSamples и присвоить ей значение sizeof(int16\_t).

Шаг 9. Инициализировать массив sampleBuff типа int16\_t размером numSamples \* header.numChannels.

Шаг 10. Переместить указатель в файле на начало данных с помощью функции fseek (file, header.subchunk1Size + header.subchunk2Size + 36, SEEK\_SET)).

Шаг 11. Вызвать функцию удаления всех значений clear() для статических полей DataL и DataR.

Шаг 12. Объявить переменную bytesRead.

Шаг 13. Прочитать данные из файла в буфер sampleBuff с помощью функции fread(sampleBuff, sizeof(int16\_t), sizeof(int16\_t), file) и присвоить результат функции переменной bytesRead.

Шаг 14. Если значение bytesRead меньше или равно 0, то переход к шагу 40. Иначе переход к шагу 15.

Шаг 15. Если значение bytesRead равно NULL, то переход к шагу 13. Иначе переход к шагу 16.

Шаг 16. Добавить считанное значение в конец вектора sampleDataL с помощью функции push\_back(complex<double>((double) sampleBuff[0], 0.0f)).

Шаг 17. Добавить считанное значение в конец вектора sampleDataR с помощью функции push\_back(complex<double>((double) sampleBuff[1], 0.0f)).

Шаг 18. Если sampleDataL.size() не равно sampleRate / 10, перейти к шагу 13. В ином случае перейти к шагу 19.

Шаг 19. Объявить вектор blockAmplitudesL типа данных vector<double>.

Шаг 20. Объявить вектор blockAmplitudesR типа данных vector<double>.

Шаг 21. Объявить вектор newSampleDataL. для комплексных чисел типа Double.

Шаг 22. Объявить вектор newSampleDataR. для комплексных чисел типа Double.

Шаг 23. Инициализировать цикл for с условием выхода при i < DSIZE. По окончанию цикла перейти к шагу 26.

Шаг 24. Добавить значение в конец вектора newSampleDataL с помощью функции push\_back(sampleDataL[i]).

Шаг 25. Добавить значение в конец вектора newSampleDataR с помощью функции push\_back(sampleDataR[i]).

Шаг 26. Вызвать функцию удаления всех значений clear() для вектора sampleDataL.

Шаг 27. Вызвать функцию удаления всех значений clear() для вектора sampleDataR.

Шаг 28. Присвоить полю sampleDataL значение newSampleDataL.

Шаг 29. Присвоить полю sampleDataR значение newSampleDataR.

Шаг 30. Вызвать функцию удаления всех значений clear() для вектора newSampleDataL.

Шаг 31. Вызвать функцию удаления всех значений clear() для вектора newSampleDataR.

Шаг 32. Вызвать функцию FFT(sampleDataL).

Шаг 33. Вызвать функцию FFT(sampleDataR).

Шаг 34. Вызвать функцию GetAmplitydeValues(sampleDataL, DSIZE), и присвоить результат переменной blockAmplitudesL.

Шаг 35. Вызвать функцию GetAmplitydeValues(sampleDataR, DSIZE), и присвоить полученный результат переменной blockAmplitudesR.

Шаг 36. Добавить значение в конец вектора DataL с помощью функции push\_back(blockAmplitudesL).

Шаг 37. Добавить значение в конец вектора DataR с помощью функции push\_back(blockAmplitudesR).

Шаг 38. Вызвать функцию clear() для вектора blockAmplitudesL.

Шаг 39. Вызвать функцию clear() для вектора blockAmplitudesR. Перейти к шагу 13.

Шаг 40. Вызвать оператор delete[] для удаления значений буфера sampleBuff.

Шаг 41. Вызвать функцию fclose(file), для закрытия текущего файла.

Шаг 42. Возвращение значения true.

Шаг 43. Завершение алгоритма.

* + 1. **Алгоритм вычисления пиковых значений амплитуды**

Данный алгоритм вычисляет те амплитудные значения вектора, которые превышают некоторое граничное значение, а так же соседние значения в этом же векторе. На основе данного анализа, высчитывается музыкальный ритм.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Объявить переменную maxPeaksVecL типа данных vector<double>.

Шаг 3. Объявить переменную maxPeaksVecR типа данных vector<double>.

Шаг 4. Объявить переменную minPeaksVecL типа данных vector<double>.

Шаг 5. Объявить переменную minPeaksVecR типа данных vector<double>.

Шаг 6. Вызвать функцию FindMinMaxPeaks (amplitudeValuesL, maxPeaksVecL, minPeaksVecL).

Шаг 7. Вызвать функцию FindMinMaxPeaks (amplitudeValuesR, maxPeaksVecR, minPeaksVecR).

Шаг 8. Инициализировать Float переменную minL и присвоить ей результат функции min\_element(minPeaksVecL, 0, minPeaksVecL.size()).

Шаг 9. Инициализировать Float переменную maxL и присвоить ей результат функции max\_element(maxPeaksVecL, 0, maxPeaksVecL.size()).

Шаг 10. Инициализировать Float переменную minR и присвоить ей результат функции min\_element(minPeaksVecR, 0, minPeaksVecR.size()).

Шаг 11. Инициализировать Float переменную maxR и присвоить ей результат функции max\_element(maxPeaksVecR, 0, maxPeaksVecR.size()).

Шаг 12. Инициализировать Float переменную centerL и присвоить значение равное (maxL – minL) \* 0.3f.

Шаг 13. Инициализировать Float переменную thresholdL и присвоить значение равное minL + centerL.

Шаг 14. Инициализировать Float переменную centerR и присвоить значение равное (maxR – minR) \* 0.3f.

Шаг 15. Инициализировать Float переменную thresholdR и присвоить значение равное minR + centerR.

Шаг 16. Инициализировать Float переменную averageSumL значением, полученным в результате функции GetAverageSum(maxPeaksVecL).

Шаг 17. Инициализировать Float переменную averageSumR значением, полученным в результате функции GetAverageSum(maxPeaksVecR).

Шаг 18. Инициализировать булеву переменную startCounting значением true.

Шаг 19. Инициализировать double переменную numValuesAfterThreshold.

Шаг 20. Инициализировать переменную first\_peak типа Float значением 0.f.

Шаг 21. Инициализировать переменную last\_peak типа Float значением 0.f.

Шаг 22. Открыть цикл for для прохода по всем элементам вектора maxPeaksVec. При условии окончания вектора перейти к шагу 33.

Шаг 23. Если элемент maxPeaksVec[i] больше или равен пороговому значению threshold, то переход к шагу 24, иначе к шагу 22.

Шаг 24. Если переменная startCounting ложно, установить его в истину и перейти шагу 25, иначе перейти к шагу 26.

Шаг 25. Присвоить переменной first\_peak значение текущего индекса цикла i.

Шаг 26. Если значение переменной averageSum больше значения threshold, то перейти к шагу 27. Если нет, то к шагу 30.

Шаг 27. Если значение переменной maxPeaksVec[i] больше значения maxPeaksVec[i+1], и больше значения maxPeaksVec[i-1], то перейти к шагу 28. Если нет, то к шагу 22.

Шаг 28. Инкрементировать значение счетчика пиковых амплитудных значений valUpperLine.

Шаг 29. Присвоить переменной last\_peak значение индекса i. Перейти к шагу 22.

Шаг 30. Проверить, является ли значение переменной maxPeaksVec[i] больше значений maxPeaksVec[i+1], maxPeaksVec[i+2], maxPeaksVec[i-1], maxPeaksVec[i-2]. Если условие истинно, то перейти к шагу 31. Если условие ложно, то перейти к шагу 22.

Шаг 31. Инкрементировать значение переменной valUpperLine.

Шаг 32. Присвоить переменной last\_peak значение индекса i. Перейти к шагу 22.

Шаг 33. Вернуть результатом данного алгоритма значение, равное valUpperLine / (last\_peak - first\_peak) \* 600.

Шаг 34. Завершение алгоритма.

**4.2.4 Алгоритм передвижения игровой пешки**

Данный алгоритм осуществляет вычисление направления и возможности передвижения игровой пешки, управляемой игроком. Рассмотрим алгоритм передвижения игровой пешки по оси X.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Осуществить приём нажатия клавиши игроком.

Шаг 3. Получить значение AxisValue, указывающее направление движения пешки.

Шаг 4. Если значение AxisValue не равно нулю, и разница значений GameTimeInSeconds и LastMovement больше 0.2 секунд, перейти к шагу 5. Иначе перейти к шагу 23.

Шаг 5. Если значение AxisValue равняется 1, то перейти к шагу 6. Иначе перейти к шагу 14.

Шаг 6. Получить текущее значение ActorLocation для игровой пешки в формате вектора, посредством вызова функции GetActorLocation().

Шаг 7. Разделить вектор на значения координат XYZ.

Шаг 8. Прибавить к значению координаты X значение поля DistanceToMove.

Шаг 9. Вычислить интерполяцию, передав во входные значения функции Lerp(Vector<Double>, Vector<Double>, Double) значения ActorLocation и полученное значение со смещением по оси X. Альфа-значение инициализировать единицей.

Шаг 10. Присвоить полученное значение полю NewActorLocation.

Шаг 11. Вызвать метод LineTraceByChannel (Vector<Double>, Vector<Double>, ArrayOfActorObject). Передать во входные значения метода значения ActorLocation, NewActorLocation и результат функции GetPlayerPawn().

Шаг 12. Если результат метода LineTraceByChannel (Vector<Double>, Vector<Double>, ArrayOfActorObject) равен false, перейти к шагу 13. Иначе перейти к шагу 23.

Шаг 13. Присвоить полю ActorLocation, значение поля NewActorLocation. Перейти к шагу 22.

Шаг 14. Получить текущее значение ActorLocation для игровой пешки в формате вектора.

Шаг 15. Разделить вектор на значения координат XYZ, представленных в формате (Float, Float, Float).

Шаг 16. Вычесть от значения координаты X значение поля DistanceToMove.

Шаг 17. Вычислить интерполяцию, передав во входные значения функции Lerp(Vector<Double>, Vector<Double>, Double) значения ActorLocation и полученное значение со смещением по оси X. Альфа-значение инициализировать единицей.

Шаг 18. Присвоить полученное значение вектора полю NewActorLocation.

Шаг 19. Вызвать метод LineTraceByChannel(Vector<Double>, Vector<Double>, ArrayOfActorObject). Передать во входные значения метода значения ActorLocation, NewActorLocation и результат функции GetPlayerPawn().

Шаг 20. Если результат метода LineTraceByChannel (Vector<Double>, Vector<Double>, ArrayOfActorObject) равен false, перейти к шагу 21. Иначе перейти к шагу 23.

Шаг 21. Присвоить полю ActorLocation, значение поля NewActorLocation. Перейти к шагу 22.

Шаг 22. Присвоить полю LastMovement значение GameTimeInSeconds, полученное посредством функции GetGameTimeInSeconds()..

Шаг 23. Завершение алгоритма.

**4.2.5 Алгоритм взаимодействия игровой пешки с платформой**

Данный алгоритм осуществляет вычисление возможности взаимодействия игровой пешки, с активной платформой, с которой происходит пересечение коллизий.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Осуществить приём нажатия клавиши игроком.

Шаг 3. Получить значение поля CanBeActivate у объекта AsBPPlayerCharacter.

Шаг 4. Если разница значений GameTimeInSeconds и LastInteract больше 0.2 секунд и значение поля CanBeActivate равняются значению true, перейти к шагу 5. Иначе перейти к шагу 15.

Шаг 5. Присвоить полю isActivate? объекта BP\_FloorPlatform значение false.

Шаг 6. Получить указатель на компоненту M\_Floor объекта BP\_FloorPlatform.

Шаг 7. Присвоить полю isActivate? компоненты M\_Floor значение 0, посредством функции SetScalarParameterValue (Double).

Шаг 8. Получить указатель на компоненту BoxCollision объекта BP\_FloorPlatform.

Шаг 9. Присвоить полю Type компоненты BoxCollision значение NoCollision.

Шаг 10. Получить указатель на объект пользовательского интерфейса, в результате функции GetHUD(PlayerController).

Шаг 11. Осуществить проверку полученного значения на отношение к классу BP\_PlayerHUD. Если результат операции положительный, перейти к шагу 12. Иначе перейти к шагу 15.

Шаг 12. Получить указатель на поле ScoreInt объекта BP\_PlayerHUD.

Шаг 13. Произвести инкрементацию значения поля ScoreInt.

Шаг 14. Присвоить полю LastInteract значение GameTimeInSeconds, полученное посредством функции GetGameTimeInSeconds().

Шаг 15. Завершение алгоритма.

**4.2.6 Алгоритм активации платформ**

Данный алгоритм осуществляет выбор и вычисление возможности активации платформы, с которой в последующем происходит взаимодействие игровой пешки.

Шаги алгоритма:

Шаг 1. Начало алгоритма.

Шаг 2. Получить текущее значение GameTimeInSeconds, посредством функции GetGameTimeInSeconds().

Шаг 3. Если текущее значение GameTimeInSeconds меньше чем поле firstBit, перейти к шагу 2. Иначе перейти к шагу 4.

Шаг 4. Вычислить случайный индекс в диапазоне от 0 до 35 посредством функции GetRandomIntegerInRange(Integer, Integer).

Шаг 5. Получить указатель на объект из массива FloorPlatformsArray, соответствующий полученному случайному индексу.

Шаг 6. Получить значение поля IsActivate?, относящемуся к полученному указателю на объект BP\_FloorPlatform.

Шаг 7. Если значение поля IsActivate? равняется false, перейти к шагу 8. Иначе перейти к шагу 4.

Шаг 8. Для указателя BP\_FloorPlatform присвоить значение true, в поле IsActivate?.

Шаг 9. Получить указатель на компоненту M\_Floor объекта BP\_FloorPlatform.

Шаг 10. Присвоить полю isActivate? компоненты M\_Floor значение 1, посредством функции SetScalarParameterValue (Double).

Шаг 11. Получить указатель на компоненту BoxCollision объекта BP\_FloorPlatform.

Шаг 12. Присвоить полю Type компоненты BoxCollision значение CollisionEnabled.

Шаг 13. Осуществить задержку выполнения алгоритма на значение, равное полю DelayTime, умноженному на 4.

Шаг 14. Присвоить полю isActivate? объекта BP\_FloorPlatform значение false.

Шаг 15. Получить указатель на компоненту M\_Floor объекта BP\_FloorPlatform.

Шаг 16. Присвоить полю isActivate? компоненты M\_Floor значение 0, посредством функции SetScalarParameterValue (Double).

Шаг 17. Получить указатель на компоненту BoxCollision объекта BP\_FloorPlatform.

Шаг 18. Присвоить полю Type компоненты BoxCollision значение NoCollision.

Шаг 19. Осуществить задержку выполнения алгоритма посредством функции Delay(Float). Время задержки определить равным значению DelayTime.

Шаг 20. Если значение GameTimeInSeconds больше значения lastBit, перейти к шагу 21. Иначе перейти к шагу 4.

Шаг 21. Завершение алгоритма.

**5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ**

В данном разделе представлена методика испытания программного средства. Проведено некоторое тестирование функционала и графического интерфейса.

**5.1 Тесты графического интерфейса**

В рамках тестирования графического интерфейса были использованы инструменты Unreal Engine 4, позволяющие оценить работу интерфейса на разнообразных устройствах. Это важно, поскольку устройства могут значительно отличаться по размерам и разрешению экранов.

**5.1.1 Адаптивность интерфейса для разных дисплеев**

Адаптивность интерфейса проверялась путем тестирования с использованием встроенных инструментов Unreal Engine 4, позволяющих устанавливать различные параметры дисплея. Были выбраны разрешения 1920х1080, 1600x900 и 1280х720, что позволило оценить масштабируемость интерфейса и его корректное отображение на экранах разного размера. Результаты тестирования иллюстрированы на рисунках 5.1, 5.2 и 5.3, демонстрирующих интерфейс при разных разрешениях.

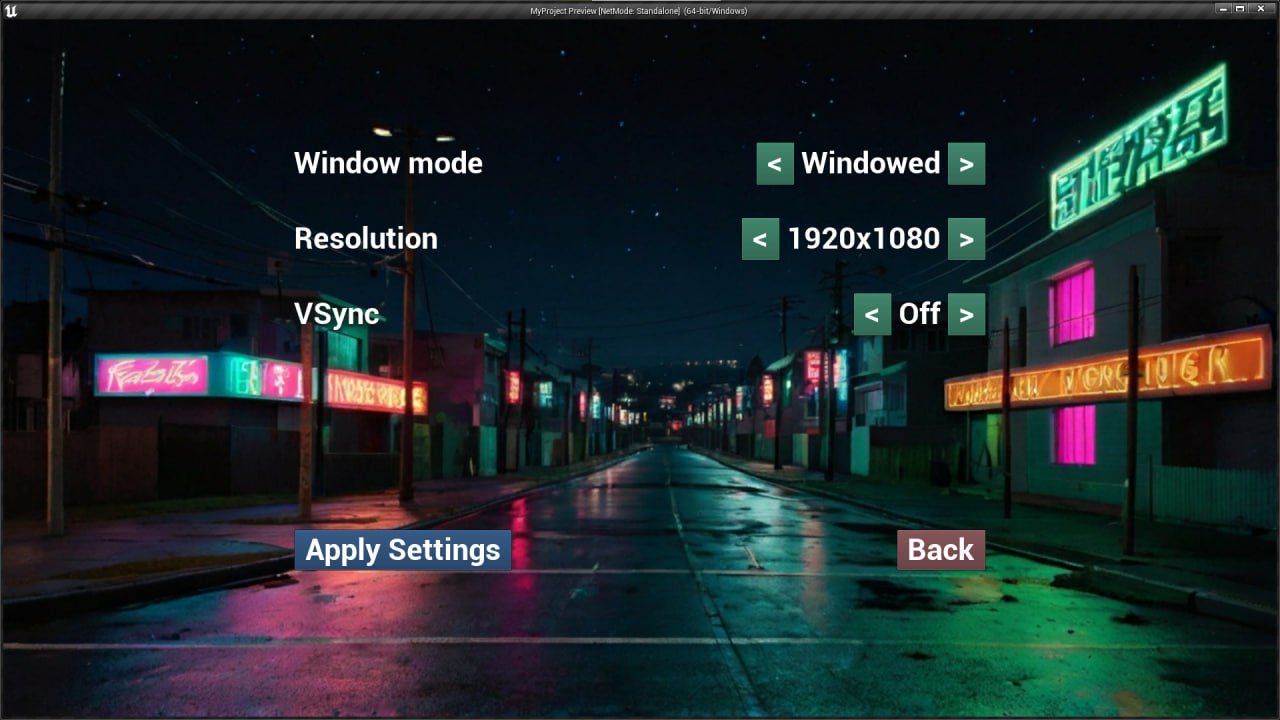


Рисунок 5.1 – Тест масштабируемости интерфейса при разрешении 1920х1080

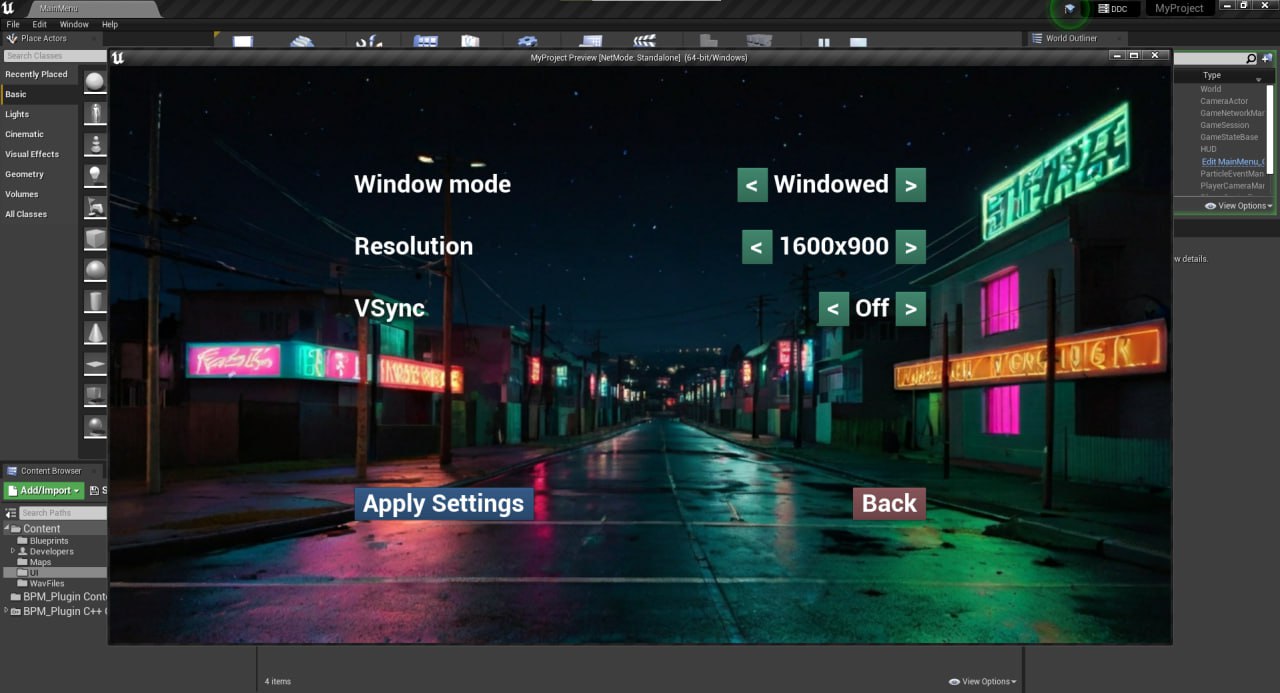


Рисунок 5.2 – Тест масштабируемости интерфейса при разрешении 1600x900

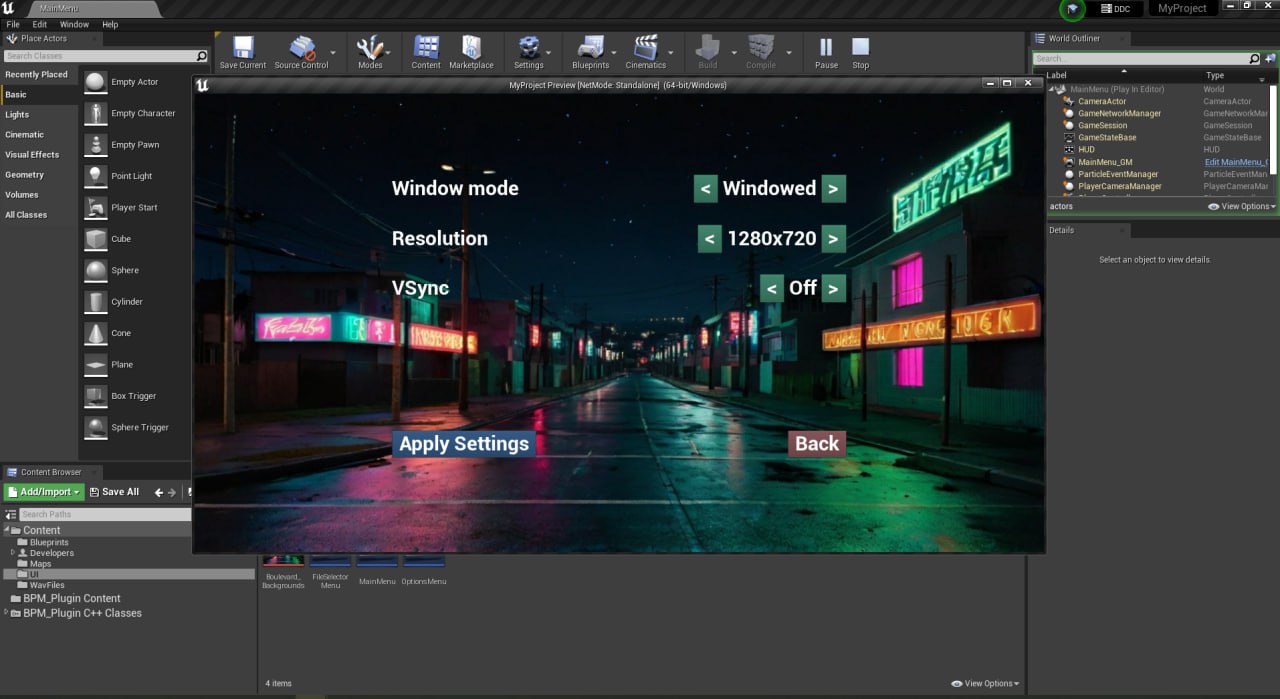


Рисунок 5.3 – Тест масштабируемости интерфейса при разрешении 1280х720

Интерфейс приложения масштабируется в соответствии с установленным разрешением в разработанном приложении. Данную особенность можно наблюдать на рисунках представленных выше. Масштабируемость элементов интерфейса, позволяет сохранять читаемость и функциональность при различных уровнях увеличения, а так же положительно влияет на удобство конечного пользователя.

**5.2 Тестирование функциональности**

Тестирование функциональности играет ключевую роль в процессе разработки программного обеспечения. Этот этап позволяет не только выявить и устранить ошибки в коде, но и проверить соответствие программы заранее определенным требованиям и спецификациям. В процессе тестирования особое внимание уделяется проверке всех функций приложения, чтобы убедиться, что каждая из них работает, как предполагалось. В завершение, тестирование функциональности способствует повышению удовлетворенности пользователя, поскольку качественно протестированное ПО обладает меньшей вероятностью сбоев и ошибок в работе.

**5.2.1 Проверка поддержки формата и наличия аудиофайла**

В данном дипломном проекте реализована поддержка аудиофайлов .wav формата. Другие форматы аудиофайлов должны быть отброшены, в связи с иной структурой заголовка, наличием возможных методов кодирования, а так же более низким разрешением хранящегося аудиосигнала. Кроме того необходимо производить проверку на корректность имени аудиофайла.

Результаты тестирования приведены в таблице 5.1:

Таблица 5.1 – Результаты тестирования загрузки аудиофайла

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Полученный результат |
| Загрузка .wav файла в проект | Корректное чтение блока заголовка файла, а так же блока данных. | Корректное чтение блока заголовка файла, а так же блока данных. |
| Загрузка .mp3 файла | Вывод сообщения об ошибке, информирующее пользователя об отсутствии поддержки файлов данного формата. | Критическая ошибка при чтении файла. Экстренное закрытие приложения. |
| Загрузка .mp3 файла, конвертированного в .wav формат | Корректное чтение блока заголовка файла. Некорректное вычисление музыкального ритма | Корректное чтение блока заголовка файла. Некорректное вычисление музыкального ритма |
| Загрузка файла с именем, содержащим максимально допустимое количество символов | Корректное чтение блока заголовка файла, а так же блока данных. | Корректное чтение блока заголовка файла, а так же блока данных. |

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Полученный результат |
| Загрузка файла с именем, содержащим специальные символы | Корректное чтение блока заголовка файла, а так же блока данных. | Критическая ошибка при чтении файла. Экстренное закрытие приложения. |
| Отмена выбора аудиофайла, закрытием диалогового окна | Вывод сообщения об отсутствии выбранного файла. | Вывод сообщения об отсутствии выбранного файла. |

**5.2.2 Проверка поддержки различных устройств управления**

В данном дипломном проекте реализована поддержка нескольких устройств управления: клавиатура, геймпад. Проверка управления осуществляется на игровом уровне. Взаимодействие с игровым меню осуществляется только посредством мыши. В проекте представлены два набора комбинаций клавиш для клавиатуры и одна комбинация для геймпада.

Результаты тестирования приведены в таблице 5.2:

Таблица 5.2 – Результаты тестирования управления

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Полученный результат |
| Управление с клавиатуры первым набором клавиш | Корректное считывание нажатых клавиш, входящих в первый набор поддерживаемых кнопок. | Корректное считывание нажатых клавиш, входящих в первый набор поддерживаемых кнопок. |
| Управление с клавиатуры вторым набором клавиш | Корректное считывание нажатых клавиш, входящих во второй набор поддерживаемых кнопок. | Корректное считывание нажатых клавиш, входящих во второй набор поддерживаемых кнопок. |
| Управление с геймпада третьим набором клавиш | Корректное считывание нажатых клавиш, входящих в третий набор поддерживаемых кнопок. | Корректное считывание нажатых клавиш, входящих в третий набор поддерживаемых кнопок. |
| Быстрое нажатие одной или нескольких клавиш одновременно | Считывание перовой нажатой клавиши и ограничение ввода на короткий промежуток времени. | Считывание перовой нажатой клавиши и ограничение ввода на короткий промежуток времени. |
| Попытка выхода игровой пешкой за пределы игровой зоны | Невозможность выхода игровой пешкой за пределы игровой зоны. | Невозможность выхода игровой пешкой за пределы игровой зоны. |

**5.2.3 Тестирование меню приложения**

В данном дипломном проекте были разработаны четыре интерактивных окна меню: главное меню, меню выбора файла, меню настроек, меню помощи управления.

Результаты тестирования приведены в таблице 5.3:

Таблица 5.3 – Результаты тестирования управления

|  |  |  |
| --- | --- | --- |
| Тест | Ожидаемый результат | Полученный результат |
| Попытка перехода из главного меню, в меню выбора аудиофайла. | Успешный переход из главного меню приложения в меню выбора аудиофайла. | Успешный переход из главного меню приложения в меню выбора аудиофайла. |
| Попытка перехода из главного меню, в меню настроек. | Успешный переход из главного меню приложения в меню настроек. | Корректный переход из главного меню приложения меню настроек. |
| Попытка перехода из главного меню, в меню помощи. | Успешный переход из главного меню приложения в меню помощи. | Успешный переход из главного меню приложения в меню помощи. |
| Попытка перехода меню выбора аудиофайла, в главное меню. | Успешный переход из меню выбора аудиофайла в главное меню приложения. | Успешный переход из меню выбора аудиофайла в главное меню приложения. |
| Попытка перехода меню настроек, в главное меню. | Успешный переход из меню настроек приложения в главное меню приложения. | Успешный переход из меню настроек приложения в главное меню приложения. |
| Попытка перехода меню помощи, в главное меню. | Успешный переход из меню помощи в главное меню приложения. | Успешный переход из меню помощи в главное меню приложения. |
| Попытка запуска игрового уровня из меню выбора аудиофайла, после выбора композиции. | Выполнение перехода из меню выбора аудиофайла на игровой уровень. | Выполнение перехода из меню выбора аудиофайла на игровой уровень. |
| Попытка запуска игрового уровня из меню выбора аудиофайла, до выбора композиции. | Ограничение игроку возможности перехода на игровой уровень, деактивацией кнопки старта. | Ограничение игроку возможности перехода на игровой уровень, деактивацией кнопки старта. |

**6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

**6.1 Системные требования**

Для работы программного средства необходимо любое устройство, соответствующее следующим системным требованиям:

Минимальные требования к программному обеспечению:

– операционная система: Windows 7;

– Direct Runtime: DirectX End–User Runtimes (June 2010) или более новые версии;

Рекомендуемые характеристики оборудования:

– операционная система: Windows 10 64–bit;

– процессор: Quad–Core Intel или AMD, 2.5 GHz или быстрее;

– память: 8 GB RAM или более;

– видеокарта/DirectX версия:DirectX 11 или DirectX 12 совместимая видеокарта;

Кроме того, для запуска приложения может понадобиться наличие установленных библиотек Visual C++ Redistributable 2017 32–bit или Visual C++ Redistributable 2017 64–bit.

**6.2 Краткое руководство пользователя**

При первом запуске приложения отображается главное меню приложения.

* + 1. **Изменение параметров приложения**



Рисунок 6.1 – Изменение параметров приложения

Для изменения параметров (рисунок 6.1), таких как режим отображения окна, разрешение изображения и активация вертикальной синхронизации:

Шаг 1. Нажать кнопку Settings, для перехода в окно настроек приложения.

Шаг 2. Выбрать желаемые настройки с помощью кнопок < и > напротив указанных параметров.

Шаг 3. Подтвердить выбранные изменения настроек нажатием кнопки Apply Settings синего цвета.

Шаг 4. Выйти из окна настроек нажатием на кнопку Back красного цвета.

* + 1. **Выбор музыкальной композиции**

Для выбора композиции для последующей игры, необходимо совершить следующие действия:

Шаг 1. Для перехода к меню выбора музыкальной композиции, необходимо нажать кнопку Play Game.

Шаг 2. Для выбора композиции, нажать кнопку Choose в левой части экрана (рисунок 6.2).

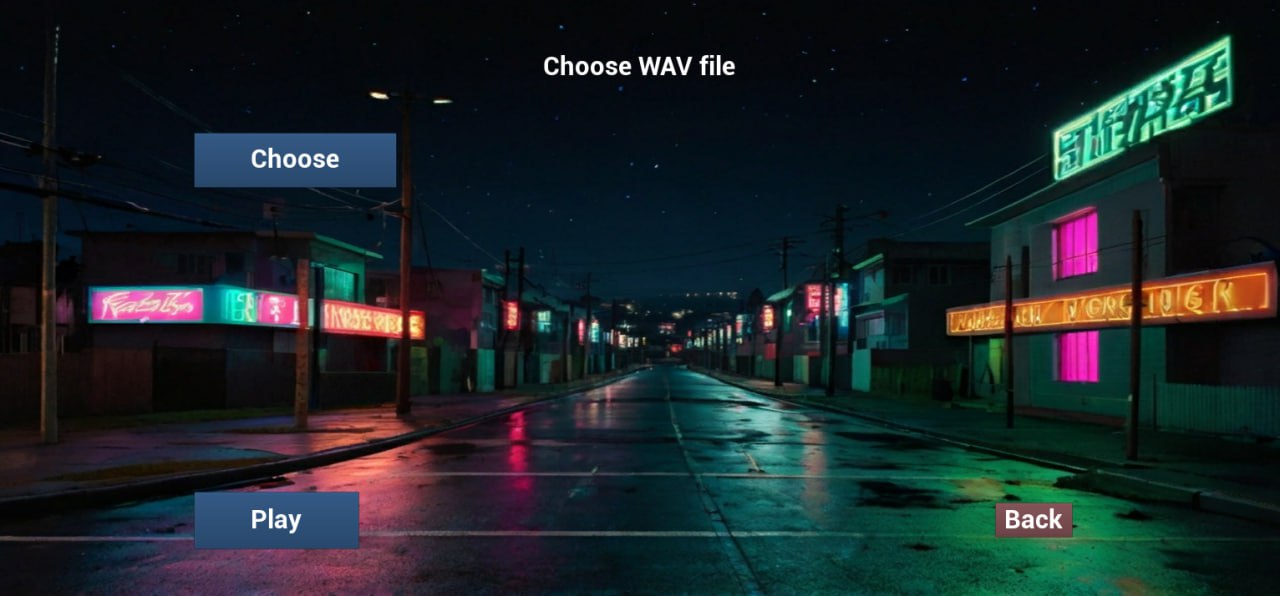


Рисунок 6.2 – Меню выбора файла

Шаг 3. В открывшемся окне, перейти в желаемую директорию и выбрать .wav файл, который будет в дальнейшем использоваться.

Шаг 4. Нажать кнопку Открыть или клавишу Enter на клавиатуре, для подтверждения выбранного файла (рисунок 6.3).

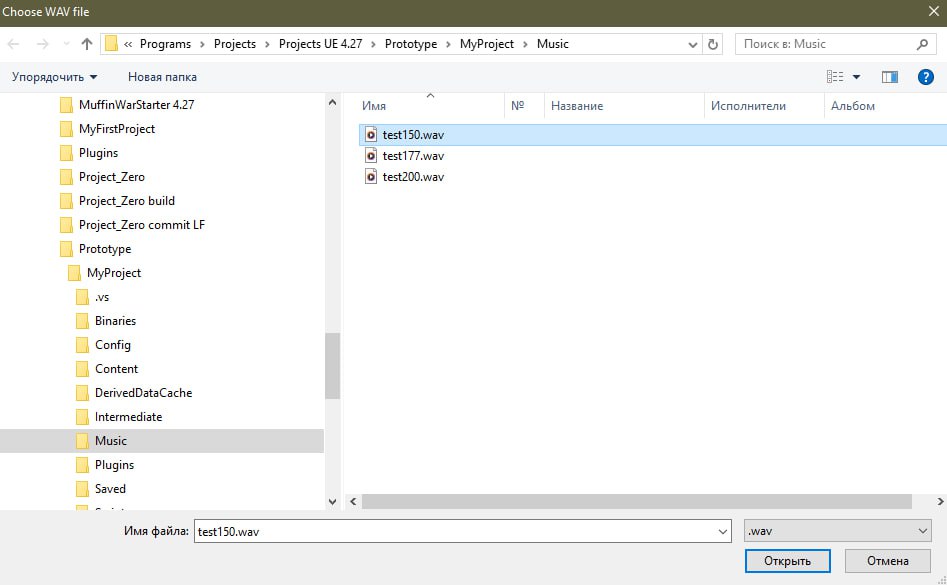


Рисунок 6.3 – Системное окно выбора аудиофайла

Шаг 5. Ожидать обработку выбранного аудиофайла. Результатом обработки, является отображение информации из заголовка выбранного аудиофайла.

Шаг 6. Нажать кнопку Play для начала игры.

* + 1. **Управление в игре**

В данной игре предусмотрено управление с помощью клавиатуры или геймпада, в зависимости от предпочтения конечного пользователя. Вся информация об управлении в игре хранится в специальном разделе – меню помощи (рисунок 6.4). Для доступа к данному меню из главного меню игры необходимо нажать кнопку Help.

Меню помощи содержит подробные инструкции по управлению игровым персонажем в данной игре, включая графическое представление отдельных клавиш, которые отвечают за определенные действия. Это меню разработано таким образом, чтобы удовлетворить потребности различных типов пользователей.

В меню помощи представлены три колонки с различными вариантами управления. Первый столбец предназначен для тех пользователей, кто склонен к управлению левой рукой, а так же к комбинации клавиш смены направления «WASD».

Второй столбец, больше походит пользователям, предпочитающим управление посредством стрелочек, а так же левшам.

Третий столбец содержит информацию об управлении посредством геймпада. Управление описано в соответствии со стандартом поддерживаемых системой Windows 10 геймпадов Xbox One.

После ознакомления с меню помощи, вы можете легко вернуться в главное меню, нажав на кнопку Back. Это позволит вам быстро приступить к игре с учетом полученных знаний об управлении.



Рисунок 6.4 – Меню помощи с управлением

**6.3 Основные правила игры**

Представленная игра является представителем жанра ритм игр. Это означает что действия игрока или игровые события привязаны к некоторому ритму, так же задаваемому игроком, либо самой игрой.

Игровое поле в данной игре представляет собой квадратную зону, заполненную белыми квадратными платформами. Всего на поле представлено 36 платформ, формируя сетку 6х6 (рисунок 6.5).

Игровой персонаж представлен в виде белой сферы, которая начинает свое движение с центра одной из платформ. Передвижение данной сферы на соседнюю платформу осуществляется одним нажатием кнопки направления движения. Движение сферы не может осуществляться по диагонали, только вверх, вниз, влево или вправо.

В начале игры все платформы белого цвета, что означает их неактивное состояние. В процессе игры случайные платформы будут менять свой цвет на синий, становясь активными. Это активное состояние длится четыре такта, после чего после чего платформа возвращается в неактивное состояние и становится снова белой. Также игрок может деактивировать платформу, переместив на нее своего персонажа и осуществив взаимодействие с ней, посредством специальной клавиши.

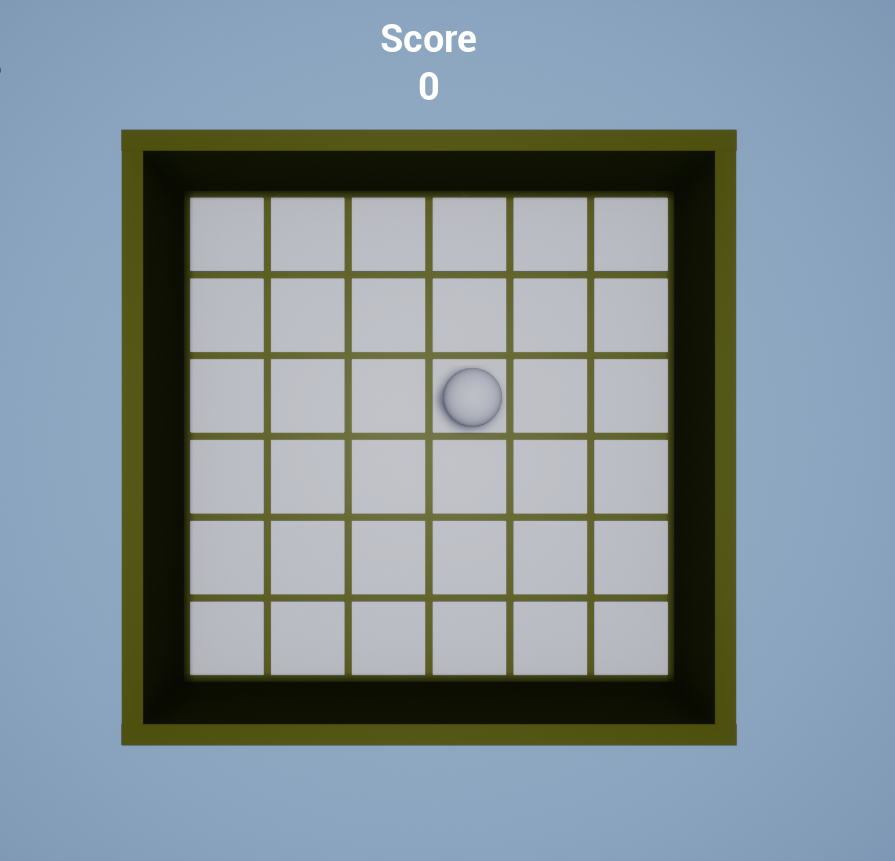


Рисунок 6.5 – Игровое поле

В начале игры все платформы белого цвета, что означает их неактивное состояние. В процессе игры случайные платформы будут менять свой цвет на синий, становясь активными. Это активное состояние длится четыре такта, после чего после чего платформа возвращается в неактивное состояние и становится снова белой. Также игрок может деактивировать платформу, переместив на нее своего персонажа и осуществив взаимодействие с ней, посредством специальной клавиши.

В данной игре музыка играет важную роль в выборе сложности игрового процесса. Ритм выбранной музыкальной композиции задаёт частоту появления активных платформ на игровом поле. Выбор музыкальной композиции с большим ритмом усложняет игровой процесс.

Целью игрока является перемещение сферы по игровому полю, и установление данной сферы на активированные платформы синего цвета. После установления сферы на указанные платформы игрок должен нажимать клавишу действия. Это приведёт к деактивации платформы и начисления очков игроку. Каждая деактивированная игроком платформа приносит ему 1 очко. Текущее количество очков игрока отображается в верхней части экрана (рисунок 6.6).

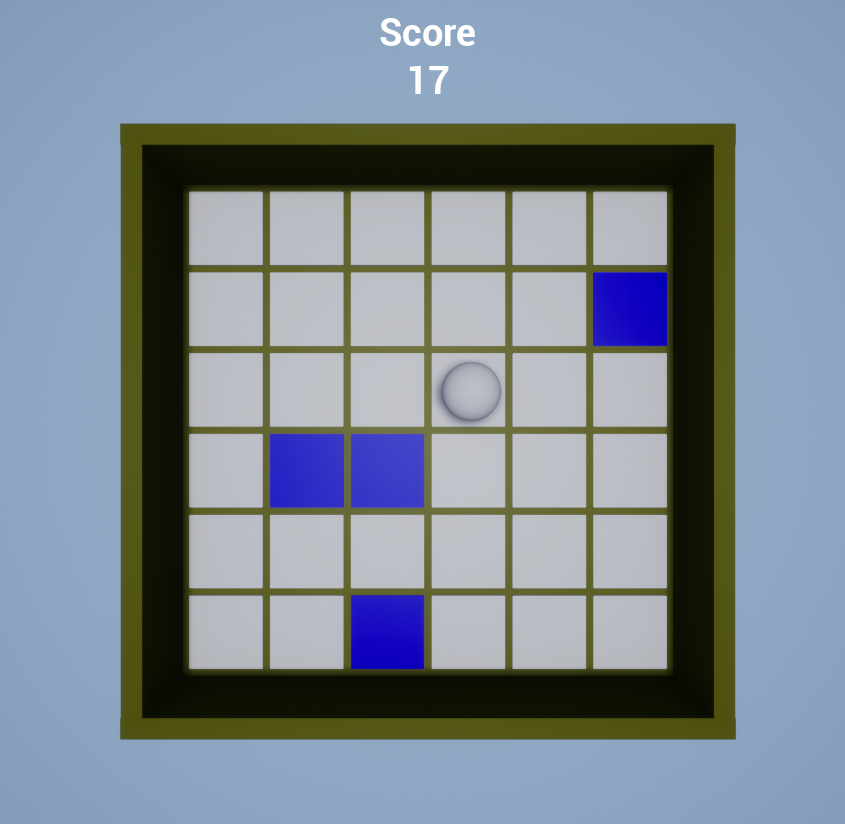


Рисунок 6.6 – Пример игрового процесса

**7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ РИТМ ИГРЫ НА UNREAL ENGINE 4, ПОСТРОЕННОЙ НА АЛГОРИТМАХ ОБРАБОТКИ ЦИФРОВОЙ СПЕКТРОГРАММЫ**

**7.1 Характеристика разработанного проектного решения**

Созданный дипломный проект представляет собой ритм игру на Unreal Engine 4 для операционной системы Windows 10, в основе которой лежат алгоритмы для чтения и обработки данных из WAV аудиофайлов. Основной разработанной частью является программная реализация, позволяющая обрабатывать аудиофайлы и использовать полученные данные, для задания ритма взаимодействия игрока, непосредственно с игровым персонажем и его окружением в самой игре.

Отличительная особенность проекта, заключается в минимизации необходимости создания собственных музыкальных композиций и лицензировании имеющихся треков, для последующего продвижения и продажи продукта.

Целью разработки данного проекта является упрощение этапа получения музыкального ритма из композиций и последующая экономия ресурсов на дальнейшем расширении и развитии продукта. Приложение предназначено для использования в развлекательных целях потребителей и коммерческой выгоды разработчика с внедренных способов монетизации.

Целевой аудиторией данного приложения являются люди, заинтересованные индустрией игр, а так же жанром ритм игр в частности. Продукт предлагает проведение досуга, минимальное время которого составляет около 3 минут, что является средним временем продолжительности проигрывания композиции.

Планируется распространение платной версии приложения через Steam.

**7.2 Расчёт инвестиций в разработку программного средства**

**7.2.1 Расчёт зарплат на основную заработную плату разработчиков**

Расчёт затрат на основную заработную плату разработчиков производится исходя из количества людей, которые занимаются разработкой программного продукта, месячной зарплаты каждого участника процесса разработки и сложности выполняемой ими работы. Затраты на основную заработную плату рассчитаны по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.с) |

где коэффициент премий и иных стимулирующих выплат;

категории исполнителей, занятых разработкой программного средства;

часовая заработная плата исполнителя *i*-й категории, р;

трудоемкость работ, выполняемых исполнителем *i*-йкатегории, ч.

Разработкой всего приложения занимается инженер-программист, Обязанности тестирования приложения лежат на инженере-тестировщике. Задачами инженера-программиста, являются написание алгоритма для чтения аудиофайла, разработки алгоритма для обработки цифровой спектрограммы аудиосигнала, разработка графического интерфейса игры, написание основной игровой логики, связи между данными, полученными в результате обработки аудиофайла и игровой логики. Инженер-тестировщик занимается выявлением неработоспособных частей приложения, а также оценивает пользовательский опыт, получаемый от приложения.

Месячная заработная плата основана на медианных показателях для Junior инженера-программиста за 2024 год по Республике Беларусь, которая составляет 1000 Долларов США в месяц, а для Junior инженера-тестировщика – 900 Долларов США[[4](https://salaries.devby.io/)]. По состоянию на 16 апреля 2024 года, 1 Доллар США по курсу Национального Банка Республики Беларусь составляет 3.2663 Белорусских рублей [[5](https://www.nbrb.by/statistics/rates/ratesdaily)].

В перерасчёте на Белорусские рубли месячные оклады для инженера-программиста и инженера-тестировщика соответственно составляет 3 266,3 и 2 939,67 Белорусских рублей соответственно.

Часовой оклад исполнителей высчитывается путём деления их месячного оклада на количество рабочих часов в месяце, то есть 160 часов.

За количество рабочих часов в месяце для инженера-программиста и инженера-тестировщика принято соответственно 196 и 32 часа.

Коэффициент премии приравнивается к единице, так как она входит сумму заработной платы. Затраты на основную заработную плату приведены в таблице:

Таблица 7.1 – Затраты на основную заработную плату

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, р | Часовой оклад, р | Трудоёмкость работ, ч | Итого, р |
| Инженер-программист | 3 266,3 | 20,41 | 196 | 4 000,36 |
| Инженер-тестировщик | 2 939,67 | 18,37 | 32 | 587,84 |
| Итого | | | | 4 588,2 |
| Премия и иные стимулирующие выплаты (0%) | | | | 0 |
| Всего затраты на основную заработную плату разработчиков | | | | 4 588,2 |

**7.2.2 Расчёт затрат на дополнительную заработную плату разработчиков**

Расчёт затрат на дополнительную заработную плату команды разработчиков рассчитывается по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.1) |

где норматив дополнительной заработной платы.

Значение норматива дополнительной заработной платы принимает за 10 %.

**7.2.3 Расчёт отчислений на социальные нужды**

Размер отчислений на социальные нужды определяется согласно ставке отчислений, которая на апрель 2024 г. равняется 35%: 29% отчисляется на пенсионное страхование, 6% − на социальное страхование. Расчёт отчислений на социальные нужды вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.2) |

где норматив отчислений в ФСЗН.

**7.2.4 Расчёт прочих расходов**

Расчёт затрат на прочие расходы определяется при помощи норматива прочих расчётов. Эта величина имеет значение 30%. Расчёт прочих расходов вычисляется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.3) |

где норматив прочих расходов.

**7.2.5 Расчёт расходов на реализацию**

Для того, чтобы рассчитать расходы на реализацию, необходимо знать норматив расходов на неё. Принимаем значение норматива равным 3%. Формула, которая использована для расчёта расходов на реализацию данного проекта:

|  |  |  |
| --- | --- | --- |
|  |  | (7.4) |

где Нр – норматив расходов на реализацию.

**7.2.6 Расчёт общей суммы затрат на разработку и реализацию**

Определяем общую сумму затрат на разработку и реализацию как сумму ранее вычисленных расходов: на основную заработную плату разработчиков, дополнительную заработную плату разработчиков, отчислений на социальные нужды, расходы на реализацию и прочие расходы. Значение определяется по формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.5) |

Таким образом, величина затрат на разработку программного средства высчитывается по указанной выше формуле и указана в таблице:

Таблица 7.2 – Затраты на разработку

|  |  |  |
| --- | --- | --- |
| Название статьи затрат | Формула/таблица для расчёта | Значение, р. |
| 1 Основная заработная плата разработчиков | См. таблицу 7.1 | 4 588,2 |
| 2 Дополнительная заработная плата разработчиков |  | 458,82 |
| 3 Отчисление на социальные нужды |  | 1 766,46 |
| 4 Прочие расходы |  | 1 376,46 |
| 5 Расходы на реализацию |  | 137,65 |
| 6 Общая сумма затрат на разработку и реализацию |  | 1. 327,59 |

**7.3 Расчёт экономического эффекта от реализации программного средства на рынке**

Для расчёта экономического эффекта организации-разработчика программного средства, а именно чистой прибыли, необходимо знать такие параметры как объем продаж, цену реализации и затраты на разработку.

Соответственно необходимо создать обоснование возможного объёма продаж, количества проданных лицензий программного средства, купленного пользователями. По данным платформы, 14 апреля 2024 года платформой Steam активно пользуются 35,28 миллиона человек. На март этого же года, по статистике используемых устройств на платформе Steam, 54.40% всех пользователей используют операционную систему Windows 10 [[6](https://store.steampowered.com/hwsurvey/?l=russian)]. Предположим, что количество пользователей, заинтересованных жанром ритм игр, составляет 4% от общего количества пользователей. Итоговое количество пользователей, соответствующих системным требованиям проекта, а так же заинтересованных жанром, составляет около 765 тысяч пользователей.

Учитывая небольшое количество конкурентов на платформе, небольшие бюджеты проекта, а так же отсутствие известности разработчиков, предположим, что 10% от общего количества пользователей Steam станут покупателями данного продукта. Это составляет около 76,500 человек.

Стоимость низкобюджетных игр от мало известных и неизвестных студий разработчиков на платформе Steam варьируется от 6 до 12 Долларов США. Расчёты в белорусском регионе Steam ведутся в Долларах США. Наиболее оптимальной ценой полной версии продукта предполагается 7 Долларов США, с вычетом комиссии платформы Steam в 10% составляет 6,3 Доллара США. Таким образом, отпускная цена копии программного средства составляет 20,58 Белорусских рубля.

Для расчёта прироста чистой прибыли необходимо учесть налог на добавленную стоимость, который высчитывается по следующей формуле:

|  |  |  |
| --- | --- | --- |
|  |  | (7.6) |

где N – количество копий(лицензий) программного продукта, реализуемое за год, шт.;

– отпускная цена копии программного средства, р. ;

– количество приобретённых лицензий;

– ставка налога на добавленную стоимость, %.

Ставка налога на добавленную стоимость по состоянию на 15 апреля 2024 года, в соответствии с действующим законодательством Республики Беларусь, составляет 20%. Используя данное значение, посчитаем НДС:

|  |  |  |
| --- | --- | --- |
|  |  |  |

Посчитав налог на добавленную стоимость, можно рассчитать прирост чистой прибыли, которую получит разработчик от продажи программного продукта. Для этого используется формула:

|  |  |  |
| --- | --- | --- |
|  |  | (7.7) |

где *N* – количество копий (лицензий) программного продукта, реализуемое за год, шт.;

Цотп – отпускная цена копии программного средства, р.;

– сумма налога на добавленную стоимость, р.; Нп – ставка налога на прибыль, %;

– рентабельность продаж копий;

– рентабельность продаж копий.

Ставка налога на прибыль, согласно действующему законодательству, по состоянию на 16.04.2023 равна 20%. Рентабельность продаж копий взята в размере 40%. Зная ставку налога и рентабельность продаж копий (лицензий), рассчитывается прирост чистой прибыли для разработчика:

|  |  |  |
| --- | --- | --- |
|  |  |  |

**7.4 Расчёт показателей экономической эффективности разработки и реализации программного средства на рынке**

Для того, чтобы оценить экономическую эффективность разработки и реализации программного средства на рынке, необходимо рассмотреть результат сравнения затрат на разработку данного программного продукта, а также полученный прирост чистой прибыли за год.

Сумма затрат на разработку меньше суммы годового экономического эффекта, поэтому можно сделать вывод, что такие инвестиции окупятся менее, чем за один год.

Таким образом, оценка экономической эффективности инвестиций производится при помощи расчёта рентабельности инвестиций (Return on Investment, ROI). Формула для расчёта ROI:

|  |  |  |
| --- | --- | --- |
|  |  | (7.8) |

где – прирост чистой прибыли, полученной от реализации программного средства на рынке информационных технологий, р.;

Зр – затраты на разработку и реализацию программного средства, р.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**7.5 Вывод об экономической целесообразности реализации проектного решения**

Проведённые расчёты экономического обоснования позволяют сделать предварительный вывод о целесообразности разработки данного программного продукта. Общая сумма затрат на разработку и реализацию составила Белорусских рублей, а отпускная цена была установлена на уровне 20,58 Белорусских рублей.

Прирост чистой прибыли за год, исходя из предполагаемого объёма продаж в размере 76 500 базовых версий в год, составляет 41 983 200 Белорусских рублей. Рентабельность инвестиций за год составляет 504045,86%.

Это результаты вычислений показывают, что разработка данного программного продукта является целесообразной и реализация программного средства по установленной цене имеет смысл.

Однако, следует учитывать возможные риски, связанные с конкуренцией со стороны крупных и средне бюджетных продуктов для выбранной целевой платформы, а так же крупных продуктов на других популярных платформах с большой аудиторией. Релиз продукта в тот же временной период может привести к незамеченности данного продукта на рынке, на фоне информационного шума, созданного конкурентными рекламными компаниями. Кроме того, высокая рентабельность связана с рисками, и расчётные результаты были получены при предполагаемом объёме продаж в 76 500 копий в год.

Тем не менее, при поддержке проект может получить долгосрочное и успешное развитие, благодаря чему количество проданных копий может превысить предполагаемое количество. Добавление, после релиза базовой версии продукта, дополнительной модели монетизации, посредством покупки внутриигрового контента или рекламы, позволит удержать прибыль в долгосрочной перспективе. В целом, инвестирование в предложенный проект хоть и рискованно, но способно привести к большой прибыли с минимальным количеством затрат на разработку, что делает инвестицию вполне оправданной.

**ЗАКЛЮЧЕНИЕ**

Дипломный проект был разработан в полном объеме. Был реализован весь необходимый функционал, для будущей доработки проекта. Основной функционал работает корректно и выполняет все поставленные задачи.

Были разработаны все необходимые алгоритмы для чтения, обработки и анализа аудиофайлов. Данные алгоритмы были приведены к системе типов данных Unreal Engine Blueprints, и имплементированы в подключаемый плагин в формате блоков. Данный плагин, в дальнейшем, может быть доработан и выпущен на рынок через Epic Games Marketplace, отдельно от основного продукта.

Для проекта было разработано функциональное меню, для настройки оконного режима приложения, разрешения изображения, и настроек вертикальной синхронизации.

Управление в игре было реализовано с поддержкой различных устройств ввода, таких как клавиатура и геймпад. Такое решение, позволяет увеличить потенциальное количество заинтересованных пользователей.

Для проекта, с помощью FL Studio, был создан специальный набор музыкальных композиций, позволяющий протестировать и опробовать проект.

Игровой прототип на основе реализованных алгоритмов чтения, обработки и анализа аудиофайлов, для вычисления музыкального ритма, был реализован в полном объёме. Несмотря на это прототип имеет множество вариаций к последующей доработке и расширению до полноценного продукта.

Расположение исходного кода данного проекта на открытом web–сервисе GitHub, может поспособствовать его развитию, доработке и финальному релизу продукта.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[[1](https://www.youtube.com/watch?v=slJA7xwDRWk)] Видеоигра «Guitar Hero» фото [Электронный ресурс] – Режим доступа: <https://www.youtube.com/watch?v=slJA7xwDRWk> – Дата доступа: 02.04.2024

[[2](https://osu.ppy.sh/beatmapsets?q=rock)] Официальная Web-библиотека для игры «Osu!» фото [Электронный ресурс] – Режим доступа: <https://osu.ppy.sh/beatmapsets?q=rock> – Дата доступа: 02.04.2024

[[3](https://www.youtube.com/watch?v=b9j8baO7nlU)] Видеоигра «One Hand Clapping» фото [Электронный ресурс] – Режим доступа: <https://www.youtube.com/watch?v=b9j8baO7nlU> – Дата доступа: 04.04.2024

[[4](https://salaries.devby.io)] Интернет-издание «Dev.by» фото [Электронный ресурс] – Зарплата в ИТ – Режим доступа: <https://salaries.devby.io> – Дата доступа: 16.04.2024

[[5](https://www.nbrb.by/statistics/rates/ratesdaily.asp)] Сайт Национальный банк Республики Беларусь фото [Электронный ресурс] – Официальные курсы белорусского рубля по отношению к иностранным валютам, устанавливаемые Национальным банком Республики Беларусь ежедневно, на 16.04.2024 – Режим доступа: <https://www.nbrb.by/statistics/rates/ratesdaily.asp> – Дата доступа: 16.04.2024

[[6](https://store.steampowered.com/hwsurvey/)] Web-сервис «Steam» фото [Электронный ресурс] – Статистика устройств пользователей – Режим доступа: <https://store.steampowered.com/hwsurvey/> – Дата доступа: 16.04.2024

Редактор звукового сигнала [Электронный ресурс] – Режим доступа: https://uengine.ru/category/site-content/docs/audio – Дата доступа: 20.04.2024

Чтение сэмплов из wav файла в массив [Электронный ресурс] – Режим доступа: https://www.cyberforum.ru/cpp-beginners/thread820676.html – Дата доступа: 20.04.2024

Алгоритмы распознавания музыки, сигнатуры, обработка данных [Электронный ресурс] – Режим доступа: https://habr.com/ru/companies/wunderfund/articles/275043/ – Дата доступа: 02.05.2024

Web-сервис «GitHub». Проект miniBPM [Электронный ресурс] – Режим доступа: https://github.com/breakfastquay/minibpm – Дата доступа: 02.05.2024

Вычислительные машины, системы и сети: дипломное проектирование (методическое пособие) [Электронный ресурс] : Минск БГУИР 2019. – Электронные данные. – Режим доступа: https://www.bsuir.by/m/12\_100229\_1\_136308.pdf – Дата доступа: 25.05.2024

Официальный форум разработчиков приложений на Unreal Engine [Электронный ресурс] – Режим доступа: https://forums.unrealengine.com/categories?tag=unreal-engine – Дата доступа: 25.05.2024

Официальная документация для разработки приложений на Unreal Engine 4.27 [Электронный ресурс] – Режим доступа: https://docs.unrealengine.com/4.27/en-US/ – Дата доступа: 25.05.2024

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Ритм игра на Unreal Engine 4, построенная на алгоритмах обработки цифровой спектрограммы. Листинг кода**

Файл WAV\_Header.h

#pragma once

#include <string>

#include "BPM\_PluginBPLibrary.h"

#include "BPM\_Plugin.h"

#include "WAV\_Reader.h"

#include "BPM\_Detector.h

using namespace std;

struct WAVHEADER

{

// WAV-формат начинается с RIFF-заголовка:

// Содержит символы "RIFF" в ASCII кодировке

// (0x52494646 в big-endian представлении)

char chunkId[4];

// 36 + subchunk2Size, или более точно:

// 4 + (8 + subchunk1Size) + (8 + subchunk2Size)

// Оставшийся размер цепочки, начиная с этой позиции.

unsigned long chunkSize;

// Содержит символы "WAVE"

// (0x57415645 в big-endian представлении)

char format[4];

// Формат "WAVE" состоит из двух подцепочек: "fmt " и "data":

// Подцепочка "fmt" описывает формат звуковых данных:

// Содержит символы "fmt "

// (0x666d7420 в big-endian представлении)

char subchunk1Id[4];

// 16 для формата PCM.

// Это оставшийся размер подцепочки, начиная с этой позиции.

unsigned long subchunk1Size;

// Для PCM = 1 (то есть, Линейное квантование).

// Значения, отличающиеся от 1, обозначают некоторый формат сжатия.

unsigned short audioFormat;

// Количество каналов. Моно = 1, Стерео = 2

unsigned short numChannels;

// Частота дискретизации. 8000 Гц, 44100 Гц

unsigned long sampleRate;

// sampleRate \* numChannels \* bitsPerSample/8

unsigned long byteRate;

// numChannels \* bitsPerSample/8

// Количество байт для одного сэмпла, включая все каналы.

unsigned short blockAlign;

// Так называемая "глубиная" или точность звучания. 8 бит, 16 бит

unsigned short bitsPerSample;

// Подцепочка "data" содержит аудио-данные и их размер.

// Содержит символы "data" или "LIST"

// (0x64617461 в big-endian представлении)

char subchunk2Id[4];

// numSamples \* numChannels \* bitsPerSample/8

// Количество байт в области данных.

unsigned long subchunk2Size;

// Various ASCII character strings. A common one is INFO

char subchunk2ListType[4];

char subchunk2Text[4];

// Далее следуют непосредственно Wav данные.

};

static float fDurationSeconds;

static float iDurationMinutes;

WAVHEADER\* GetHeader(string filename);

void PrintHeader(WAVHEADER& header, FString& Header1, FString& Header2);

Файл WAV\_Header.cpp

#include "WAV\_Header.h"

#include <stdio.h>

#include <tchar.h>

#include <conio.h>

#include <cmath>

#include <string>

using namespace std;

WAVHEADER\* GetHeader(string filename)

{

FILE\* file;

errno\_t err;

FString formattedFilename = FString(filename.c\_str());

GEngine->AddOnScreenDebugMessage(-1, 10.f, FColor::Magenta, FString::Printf(TEXT("%s"), \*formattedFilename));

err = fopen\_s(&file, filename.c\_str(), "rb");

if (err)

{

GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Green, FString::Printf(TEXT("Ошибка при открытии файла, ошибка %d"), err));

return NULL;

}

fread\_s(&header, sizeof(WAVHEADER), sizeof(WAVHEADER), 1, file);

fclose(file);

return &header;

}

void PrintHeader(FString& Header1, FString& Header2)

{

Header1 = "", Header2 = "";

FString chunkId = FString::Printf(TEXT("Chunk Id: %c.%c.%c.%c\n"), header.chunkId[0], header.chunkId[1], header.chunkId[2], header.chunkId[3]);

FString chunkSize = FString::Printf(TEXT("Chunk size: %d\n"), header.chunkSize);

FString format = FString::Printf(TEXT("Format: %c.%c.%c.%c\n"), header.format[0], header.format[1], header.format[2], header.format[3]);

FString subchunk1Id = FString::Printf(TEXT("Subchunk1Id: %c.%c.%c.%c\n"), header.subchunk1Id[0], header.subchunk1Id[1], header.subchunk1Id[2], header.subchunk1Id[3]);

FString subchunk1Size = FString::Printf(TEXT("Subchunk1Size: %d\n"), header.subchunk1Size);

FString audioFormat = FString::Printf(TEXT("Audio format: %d\n"), header.audioFormat);

FString numChannels = FString::Printf(TEXT("Channels: %d\n"), header.numChannels);

FString sampleRate = FString::Printf(TEXT("Sample rate: %d\n"), header.sampleRate);

FString byteRate = FString::Printf(TEXT("Byte rate: %d\n"), header.byteRate);

FString blockAlign = FString::Printf(TEXT("Block Align: %d\n"), header.blockAlign);

FString bitsPerSample = FString::Printf(TEXT("Bits per sample: %d\n"), header.bitsPerSample);

FString subchunk2Id = FString::Printf(TEXT("Subchunk2Id: %c.%c.%c.%c\n"), header.subchunk2Id[0], header.subchunk2Id[1], header.subchunk2Id[2], header.subchunk2Id[3]);

FString subchunk2Size = FString::Printf(TEXT("Subchunk2Size: %d\n"), header.subchunk2Size);

FString subchunk2ListType = FString::Printf(TEXT("Subchunk2ListType: %c.%c.%c.%c\n"), header.subchunk2ListType[0], header.subchunk2ListType[1], header.subchunk2ListType[2], header.subchunk2ListType[3]);

// Посчитаем длительность воспроизведения в секундах

fDurationSeconds = 1.f \* header.chunkSize / (header.bitsPerSample / 8) / header.numChannels / header.sampleRate;

iDurationMinutes = floor(fDurationSeconds) / 60.f;

float intPart;

int x = static\_cast<int>(modff(iDurationMinutes, &intPart) \* 100.f);

fDurationSeconds = (float)(x / 100.f \* 60.f);

FString DurationTime = FString::Printf(TEXT("Duration: %02d:%02.f"), (int)iDurationMinutes, fDurationSeconds);

Header1.Append(chunkId) += chunkSize += format += subchunk1Id += subchunk1Size += audioFormat += numChannels;

Header2.Append(sampleRate) += byteRate += blockAlign += bitsPerSample += subchunk2Id += subchunk2Size += subchunk2ListType += DurationTime;

}

Файл WAV\_Reader.h

#pragma once

#include <string>

#include <complex>

#include <vector>

#include "WAV\_Header.h"

#include "BPM\_Detector.h"

using namespace std;

static vector<complex<double>> sampleDataL;

// Вектор для комплексных значений для FFT левого канала (формат double - 8 байт)

static vector<complex<double>> sampleDataR;

// Вектор для комплексных значений для FFT правого канала (формат double - 8 байт)

static int DSIZE = 4096;

// Размер массивов входных и выходных значений. Должен быть кратен 2

bool ReadFromFile(string filename, vector<vector<double>>& DataL, vector<vector<double>>& DataR);

void AmplitudeBlockProc(vector<double>& blockAmplitudesL, vector<double>& blockAmplitudesR);

void FFT(vector<complex<double>>& inVec);

vector<complex<double>> PushBackZeros(vector<complex<double>> inData, int border);

vector<double> GetAmplitydeValues(vector<complex<double>> inData, int sampleRate);

void PrintSpectrogram(vector<double> inData, int sampleRate);

Файл WAV\_Reader.cpp

#include <string>

#include <iostream>

#include <fstream>

#include <complex>

#include <vector>

#include <algorithm>

#include <numeric>

#include <algorithm>

#include <iomanip>

#include "BPM\_Detector.h"

#include "WAV\_Header.h"

#include "WAV\_Reader.h"

using namespace std;

bool ReadFromFile(string filename, vector<vector<double>>& DataL, vector<vector<double>>& DataR)

{

FILE\* file;

errno\_t err;

int sampleRate = header.sampleRate;

// sampleRate - Частота дискретизации. То сколько раз за 1 секунду, производится замер амплитудных значений сигнала

FString formattedFilename = FString(filename.c\_str());

err = fopen\_s(&file, filename.c\_str(), "rb");

// В случае ошибки выход из функции и вывод сообщения

if (err)

{

GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Green, FString::Printf(TEXT("Error 1! Can't open file, %d"), err));

return false;

}

static const uint16\_t numSamples = sizeof(int16\_t);

// numSamples - Константное значение размера для инициализации sampleBuff равное 2 байтам

int16\_t\* sampleBuff = new int16\_t[numSamples \* header.numChannels];

// sampleBuff - Массив для левого и правого значения амплитуды

//fseek(file, header.subchunk1Size + header.subchunk2Size + 36, SEEK\_SET);

// Пропуск заголовка на размер subchunk1 и subchunk2, плюс 36 (начальные значения)

DataL.clear(), DataR.clear();

sampleDataL.clear(), sampleDataR.clear();

size\_t bytesRead;

while ((bytesRead = fread\_s(sampleBuff, sizeof(int16\_t)\*2, sizeof(int16\_t), 2, file)) > 0)

{

// Чтение data из файла

if (bytesRead == NULL)

{ // Выход из цикла при ошибке чтения

GEngine->AddOnScreenDebugMessage(-1, 5.f, FColor::Green, FString::Printf(TEXT("Error 2! Can't read file, bytesRead != buffSize")));

break;

}

sampleDataL.push\_back(complex<double>(static\_cast<double> (sampleBuff[0]), 0.0f));

// Сохраняем данные как комплексные значения

sampleDataR.push\_back(complex<double>(static\_cast<double> (sampleBuff[1]), 0.0f));

complex<double> x = sampleDataL[25];

if (sampleDataL.size() == sampleRate / 10)

{ // Набран набор значений равный ~DSIZE

vector<double> blockAmplitudesL;

vector<double> blockAmplitudesR;

AmplitudeBlockProc(blockAmplitudesL, blockAmplitudesR);

// Обработка блоков амплитудных значений перед анализом

DataL.push\_back(blockAmplitudesL);

// Сохранение наборов амплитудных значений

DataR.push\_back(blockAmplitudesR);

blockAmplitudesL.clear(), blockAmplitudesR.clear();

}

}

delete[] sampleBuff;

fclose(file);

return true;

}

void AmplitudeBlockProc(vector<double>& blockAmplitudesL, vector<double>& blockAmplitudesR)

{

vector<complex<double>> newSampleDataL;

vector<complex<double>> newSampleDataR;

for (int i = 0; i < DSIZE; i++)

{ // Перезапись значений с обрезанием верхних частот

newSampleDataL.push\_back(sampleDataL[i]);

newSampleDataR.push\_back(sampleDataR[i]);

}

sampleDataL.clear();

sampleDataR.clear();

sampleDataL = newSampleDataL;

sampleDataR = newSampleDataR;

newSampleDataL.clear();

newSampleDataR.clear();

FFT(sampleDataL); // Преобразование данных Фурье

FFT(sampleDataR);

blockAmplitudesL = GetAmplitydeValues(sampleDataL, DSIZE);

blockAmplitudesR = GetAmplitydeValues(sampleDataR, DSIZE);

}

void FFT(vector<complex<double>>& inVec)

{

double pi = 3.141592653589793238460;

int N = inVec.size();

if (N <= 1) return;

vector<complex<double>> even = vector<complex<double>>(N / 2);

vector<complex<double>> odd = vector<complex<double>>(N / 2);

for (int i = 0; i < N / 2; ++i)

{

even[i] = inVec[i \* 2];

odd[i] = inVec[i \* 2 + 1];

}

FFT(even);

FFT(odd);

for (int k = 0; k < N / 2; ++k)

{

complex<double> temp = std::polar(1.0, -2 \* pi \* k / N) \* odd[k];

inVec[k] = even[k] + temp;

inVec[k + N / 2] = even[k] - temp;

}

}

vector<complex<double>> PushBackZeros(vector<complex<double>> inData, int border)

{

for (int i = inData.size(); i < border; i++)

{

inData.push\_back(complex<double>(0.0f, 0.0f));

}

return inData;

}

vector<double> GetAmplitydeValues(vector<complex<double>> inData, int sampleRate)

{

vector<double> amplitudeValues;

for (size\_t i = 0; i < inData.size(); ++i)

{

amplitudeValues.push\_back(abs(inData[i]));

}

return amplitudeValues;

}

}

Файл BPM\_Detector.h

#pragma once

#include <string>

#include <complex>

#include <vector>

#include "WAV\_Header.h"

#include "WAV\_Reader.h"

using namespace std;

float Detect\_BPM(vector<vector<double>>& amplitudeValuesL, vector<vector<double>>& amplitudeValuesR);

double GetAverageSum(vector<double> maxPeaksVec);

double FindBassPeaks(vector<double> maxPeaksVec, double threshold, double averageSum, double center);

void FindMinMaxPeaks(vector<vector<double>> data, vector<double>& maxPeaksVec, vector<double>& minPeaksVec);

double max\_element(vector<double> data, int begin, int end);

double min\_element(vector<double> data, int begin, int end);

Файл BPM\_Detector.cpp

#include "BPM\_Detector.h"

#include "WAV\_Header.h"

#include "WAV\_Reader.h"

using namespace std;

float Detect\_BPM(vector<vector<double>> &amplitudeValuesL, vector<vector<double>> &amplitudeValuesR)

{

vector<double> maxPeaksVecL;

vector<double> maxPeaksVecR;

vector<double> minPeaksVecL;

vector<double> minPeaksVecR;

FindMinMaxPeaks(amplitudeValuesL, maxPeaksVecL, minPeaksVecL);

FindMinMaxPeaks(amplitudeValuesR, maxPeaksVecR, minPeaksVecR);

double minL = min\_element(minPeaksVecL, 0, minPeaksVecL.size());

double maxL = max\_element(maxPeaksVecL, 0, maxPeaksVecL.size());

double minR = min\_element(minPeaksVecR, 0, minPeaksVecR.size());

double maxR = max\_element(maxPeaksVecR, 0, maxPeaksVecR.size());

double centerL = (maxL - minL) \* 0.3f;

double thresholdL = minL + 0.30 \* (maxL - minL);

double centerR = (maxR - minR) \* 0.3f;

double thresholdR = minR + 0.30 \* (maxR - minR);

double averageSumL = GetAverageSum(maxPeaksVecL);

double averageSumR = GetAverageSum(maxPeaksVecR);

double BPM\_L = FindBassPeaks(maxPeaksVecL, thresholdL, averageSumL, centerL);

// Возвращает в Blueprint

double BPM\_R = FindBassPeaks(maxPeaksVecR, thresholdR, averageSumR, centerR);

//int BPM = (int)((BPM\_L + BPM\_R) / 2);

return BPM;

}

double GetAverageSum(vector<double> maxPeaksVec)

{

double averageSum = 0;

for (int i = 0; i < maxPeaksVec.size(); i++) {

averageSum += maxPeaksVec[i];

}

return (averageSum / maxPeaksVec.size());

}

double FindBassPeaks(vector<double> maxPeaksVec, double threshold, double averageSum, double center)

{

bool startCounting = false;

double numValuesAfterThreshold = 0;

bool check = averageSum > center;

double first\_peak = 0, last\_peak = 0;

for (int i = 0; i < maxPeaksVec.size(); i++) {

// Peaks / (last\_peak - first\_peak) \* 600

if (maxPeaksVec[i] >= threshold) {

if (check) {

if (maxPeaksVec[i] > maxPeaksVec[i - 1] && maxPeaksVec[i] > maxPeaksVec[i + 1]) {

if (!startCounting) {

startCounting = true;

first\_peak = i;

}

++numValuesAfterThreshold;

last\_peak = i;

}

}

else {

if (maxPeaksVec[i] > maxPeaksVec[i - 1] && maxPeaksVec[i] > maxPeaksVec[i - 2] && maxPeaksVec[i] > maxPeaksVec[i + 1] && maxPeaksVec[i] > maxPeaksVec[i + 2]) {

if (!startCounting) {

startCounting = true;

first\_peak = i;

}

++numValuesAfterThreshold;

last\_peak = i;

}

}

}

}

// Num of beats per 60 sec

return (numValuesAfterThreshold);// / (last\_peak - first\_peak) \* 600);

}

void FindMinMaxPeaks(vector<vector<double>> data, vector<double>& maxPeaksVec, vector<double>& minPeaksVec)

{

double tempMin = 500000.f, tempMax = 0.f;

for (int count = 0; count < data.size(); count++) {

for (int i = 5; i < 40; i++) {

if (data[count][i] > tempMax) {

tempMax = data[count][i];

}

if (data[count][i] < tempMin) {

tempMin = data[count][i];

}

}

maxPeaksVec.push\_back(tempMax);

minPeaksVec.push\_back(tempMin);

}

}

double max\_element(vector<double> data, int begin, int end)

{

int max = begin;

for (int i = begin; i < end; i++) {

if (data[max] < data[i])

max = i;

};

return data[max];

}

double min\_element(vector<double> data, int begin, int end)

{

int min = data[begin];

if (data.size() == 0 && data[0] != 0) {

return data[0];

}

for (int i = begin; i < end; i++) {

if (min > data[i])

min = i;

}

return data[min];

}

Файл BPM\_PluginBPLibrary.h

// Copyright Epic Games, Inc. All Rights Reserved.

#pragma once

#include "Kismet/BlueprintFunctionLibrary.h"

#include "WAV\_Reader.h"

#include "WAV\_Header.h"

#include "BPM\_Detector.h"

#include "BPM\_PluginBPLibrary.generated.h"

UCLASS()

class UBPM\_PluginBPLibrary : public UBlueprintFunctionLibrary

{

GENERATED\_UCLASS\_BODY()

public:

static WAVHEADER header;

UFUNCTION(BlueprintCallable, meta = (DisplayName = "Get File Header", Keywords = "return Header and name of file"), Category = "BPM\_Plugin")

static FString GetFileHeader(const FString& Filename, FString& Header1, FString& Header2);

UFUNCTION(BlueprintCallable, meta = (DisplayName = "Read WAV File", Keywords = "read wav file and bool state of operation"), Category = "BPM\_Plugin")

static bool ReadWAVFile(const FString& Filename, float& BPM, float& firstBit, float& lastBit);

};

Файл BPM\_PluginBPLibrary.cpp

// Copyright Epic Games, Inc. All Rights Reserved.

#include "BPM\_PluginBPLibrary.h"

#include "BPM\_Plugin.h"

#include "WAV\_Reader.h"

#include "WAV\_Header.h"

#include "BPM\_Detector.h"

#include "string.h"

#include <string>

#include <vector>

using namespace std;

WAVHEADER UBPM\_PluginBPLibrary::header;

UBPM\_PluginBPLibrary::UBPM\_PluginBPLibrary(const FObjectInitializer& ObjectInitializer)

: Super(ObjectInitializer)

{

}

FString UBPM\_PluginBPLibrary::GetFileHeader(const FString& Filename, FString& Header1, FString& Header2) {

FString MutableString;

MutableString.Append(Filename);

GetHeader(TCHAR\_TO\_UTF8(\*MutableString));

PrintHeader(header, Header1, Header2);

return Filename;

}

bool UBPM\_PluginBPLibrary::ReadWAVFile(const FString& Filename, float& BPM, float& firstBit, float& lastBit) {

FString MutableString;

MutableString.Append(Filename);

vector<vector<double>> DataL;

vector<vector<double>> DataR;

BPM = 0;

firstBit = 0;

lastBit = 0;

bool state = ReadFromFile(TCHAR\_TO\_UTF8(\*MutableString), BPM, firstBit, lastBit);

return state;

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Ритм игра на Unreal Engine 4, построенная на алгоритмах обработки цифровой спектрограммы. Спецификация**

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Ритм игра на Unreal Engine 4, построенная на алгоритмах обработки цифровой спектрограммы. Ведомость документов**