МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-12 Информационные системы и  
технологии (в игровой индустрии)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Программирование сетевых приложений»

на тему: «Сетевое игровое приложение Windows Form «Танки» с использованием winsock»

Исполнитель: студент группы ИТИ-41

Карпенко Д.Е.

Руководитель: доцент

Курочка К.С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2024

СОДЕРЖАНИЕ

[Введение 4](#_Toc184484506)

[1 Аналическийобзор средств разработки 5](#_Toc184484507)

[1.1 Выбор языка программирования 5](#_Toc184484508)

[1.2 Выбор технологии работы с графикой 9](#_Toc184484509)

[1.3 Перечень используемых технологий 12](#_Toc184484510)

[2 Проектирование архитектуры и алгоритмов для сетевого приложения «Танковая Дуэль» 13](#_Toc184484511)

[2.1 Архитектура приложения 13](#_Toc184484512)

[2.2 Модели синхронизации пользователей и сервера в игровых приложениях 16](#_Toc184484513)

[2.3 Алгоритмы обработки коллизий 18](#_Toc184484514)

[2.4 Архитектура сети 19](#_Toc184484515)

[2.5 Выбранные архитектурные и алгоритмические решения 20](#_Toc184484516)

[3 Описание программы и разработанных алгоритмов для игры «Танковая Дуэль» 21](#_Toc184484517)

[3.1 Описание главных клиентских классов 21](#_Toc184484518)

[3.2 Описание главных серверных классов 23](#_Toc184484519)

[3.3 Апробация игрового приложения «Танковая Дуэль» 26](#_Toc184484520)

[3.4 Нагрузочные тесты для сервера 27](#_Toc184484521)

[3.5 Модульные тесты 28](#_Toc184484522)

[Заключение 29](#_Toc184484523)

[Список использованных источников 30](#_Toc184484524)

[Приложение А 31](#_Toc184484525)

[Приложение Б 89](#_Toc184484526)

[Приложение В 91](#_Toc184484527)

[Приложение Г 92](#_Toc184484528)

[Приложение Д 93](#_Toc184484529)

[Приложение Е 94](#_Toc184484530)

# **ВВЕДЕНИЕ**

Онлайн-игры занимают значительное место в современной культуре, объединяя миллионы людей по всему миру. Они выступают в роли интерактивных платформ, через которые обеспечивается взаимодействие пользователей в реальном времени в виртуальной среде. С развитием технологий и распространением интернета наблюдается стремительный рост популярности таких игр, что делает их важной частью досуга и социальной активности.

Одна из актуальных тенденций в данной области заключается в создании кооперативных игр, ориентированных на совместное взаимодействие игроков. Такие игры способствуют развитию командных навыков и коммуникации, создавая уникальные сценарии взаимодействий.

Кооперативные игры предоставляют пользователям уникальную возможность взаимодействовать друг с другом, объединяя элементы стратегии, тактики и соревновательности. В условиях постоянного развития онлайн-сервисов и росте интереса к играм, обеспечивающим взаимодействие в режиме реального времени, создание качественной кооперативной игры является актуальной задачей. Цель работы – создание кооперативной игры для двух игроков под названием «Танковая дуэль».

Задачи проекта:

* разработка клиент-серверного взаимодействия с использованием технологии *winsock* для передачи данных между клиентами;
* создание графического интерфейса игры с применением библиотеки *OpenTK*;
* разработка игрового процесса, включающего основные элементы управления и взаимодействия игроков;
* оптимизация производительности клиентского и серверного приложения для обеспечения стабильной работы игры.

Проект «Танковая дуэль» направлен на разработку локальной кооперативной игры, которая может быть использована как для развлечения, так и для демонстрации возможностей современных технологий разработки игр.

Для реализации проекта будут использованы следующие технологии и инструменты:

– язык программирования *C*#;

* среда разработки *Visual* *Studio* 2022;
* графическая библиотека *OpenTK*;
* технология *winsock* – программный интерфейс для разработки сетевых приложений, обеспечивающий эффективное клиент-серверное взаимодействие.

# **1 АНАЛИЧЕСКИЙ****ОБЗОР СРЕДСТВ РАЗРАБОТКИ**

## **1.1 Выбор языка программирования**

*C*# – объектно-ориентированный язык программирования. С момента создания язык *C*# обогатился функциями для поддержки новых рабочих нагру­зок и современными рекомендациями по разработке ПО. Основой *C*# является концепция объектно-ориентированного подхода. Обеспечивается это при помощи объектов, классов.

Разработчику приложений нет нужды следить за очисткой объектов которые не используются, за это отвечает так называемый сборщик мусора(*Garbage* *Collector*). Сборка мусора автоматически освобож­дает память, занятую недостижимыми неиспользуемыми объектами. Типы, до­пускающие значение *null*, обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восста­новлению после них. Имеются лямбда-выражения(*LINQ*). Синтаксис *LINQ* создает общий шаблон для работы с данными из любого источника. Поддержка языков для асинхронных операций предоставляет синтаксис для создания распреде­ленных систем. В *C*# имеется Единая система типов. Все типы *C*# наследуются от базового класса *object*. Все типы используют общий набор операций, а зна­чения любого типа можно хранить, передавать и обрабатывать схожим обра­зом. *C*# поддерживает как ссылочные типы, так и значимые типы. *C*# позволяет динамически выделять объекты и хранить упрощенные структуры в стеке. *C*# поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность. *C*# предоставляет итераторы, кото­рые позволяют разработчикам классов коллекций определять пользователь­ские варианты поведения для клиентского кода.

*C*# подчеркивает управление версиями, чтобы обеспечить совместимость программ и библиотек с течением времени. Вопросы управления версиями су­щественно повлияли на такие аспекты разработки *C*#, как раздельные модифи­каторы *virtual* и *override*, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса.

Программы *C*# выполняются в .*NET*, виртуальной системе выполнения, вызывающей общеязыковую среду выполнения (*CLR*) и набор библиотек клас­сов. Название исполняющей среды – «общеязыковая исполняющая среда» (*common language runtime*, *CLR*) – говорит само за себя: это исполняющая среда, которая подходит для разных языков программирования [1, с. 22]. Среда *CLR* – это реализация общеязыковой инфраструктуры языка (*CLI*), являющейся международным стандартом, от корпорации Майкрософт. *CLI* является основой для создания сред выполнения и разработки, в которых языки и библиотеки прозрачно работают друг с другом.

*IL* – не зависящий от процессора машинный язык, разработанный *Microsoft* после консультаций с несколькими коммерческими и академиче­скими организациями, специализирующимися на разработке языков и компи­ляторов. *IL* – язык более высокого уровня в сравнении с большинством других машинных языков. Он позволяет работать с объектами и имеет команды для со­ здания и инициализации объектов, вызова виртуальных методов и непосред­ственного манипулирования элементами массивов[1, c. 29].

Исходный код, написанный на языке *C*# компилируется в [промежуточный язык (*IL*)](https://learn.microsoft.com/ru-ru/dotnet/standard/managed-code), который соответствует спецификациям *CLI*. Код на языке *IL* и ре­сурсы, в том числе растровые изображения и строки, сохраняются в сборке, обычно с расширением .*dll*.

При выполнении программы *C*# сборка загружается в среду *CLR*. Среда *CLR* выполняет *JIT*-компиляцию из кода на языке *IL* в инструкции машинного языка. Среда *CLR* выполняет другие операции такие как, автоматическую сборку мусора, обработку исключений и управление ресурсами. Код, выполня­емый средой *CLR*, иногда называют «управляемым кодом». «Неуправляемый код» компилируется на машинный язык, предназначенный для конкретной платформы.

Обеспечение взаимодействия между языками является ключевой особен­ностью .*NET*. Код *IL*, созданный компилятором *C*#, соответствует специфика­ции общих типов (*CTS*). Код *IL*, созданный из кода на *C*#, может взаимодей­ствовать с кодом, созданным из версий .*NET* для языков *F*#, *Visual* *Basic*, *C*++. Существует более 20 других языков, совместимых с *CTS*. Одна сборка может содержать несколько модулей, написанных на разных языках .*NET*, и все типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

В дополнение к службам времени выполнения .*NET* также включает рас­ширенные библиотеки. Эти библиотеки поддерживают множество различных рабочих нагрузок. Они упорядочены по пространствам имен, которые предо­ставляют разные полезные возможности: от операций файлового ввода и вывода до управления строками и синтаксического анализа *XML*, от платформ веб-приложений до элементов управления *Windows Forms* и *Windows Presentation*. Обычно приложение *C*# активно используют библиотеку классов .*NET* для решения типовых задач.

В *C*# существуют две разновидности типов ссылочные типы и значимые типы. Переменные значимого типа содержат непосредственно данные, а в пе­ременных ссылочных типов хранятся ссылки на нужные данные, которые име­нуются объектами. Две переменные ссылочного типа могут ссылаться на один и тот же объект, поэтому может случиться так, что операции над одной пере­менной затронут объект, на который ссылается другая переменная.

*Java* – это популярный язык программирования высокого уровня, разработанный компанией *Sun Microsystems* (в настоящее время принадлежит корпорации *Oracle*) в середине 1990-х годов. Он был разработан, чтобы быть простым, безопасным и переносимым, то есть он может работать на любой платформе с установленной виртуальной машиной *Java* (*JVM*).

Данный ЯП(язык программирования) является объектно-ориентированным языком, что означает использование объектов и классов для организации кода. В нем также реализовано автоматическое управление памятью, что снижает риск утечки памяти и облег­чает разработчикам написание надежных и масштабируемых программ.

Одним из ключевых достоинств *Java* является ее платформенная незави­симость. После компиляции *Java*-программы в байткод она может быть запу­щена на любой платформе с установленной *JVM*, включая *Windows*, *Mac* *OS* *X*, *Linux* и многие другие операционные системы. Это делает *Java* популярным выбором для создания приложений, которые должны работать на самых разных устройствах.

*Java* имеет широкий спектр применения, от настольных приложений до веб-приложений корпоративного уровня и разработки мобильных приложений для *Android*. Он также широко используется при разработке внутренних систем и приложений на стороне сервера благодаря своей производительности, мас­штабируемости и безопасности.

*Java* имеет встроенный сборщик мусора, который автоматически управ­ляет памятью, возвращая объекты, на которые программа больше не ссылается. По функциональности сборщик мусора в *Java* похож на сборщик мусора в *C*#, но есть некоторые различия в его работе.

Сборщик мусора в *Java* использует процесс под названием *mark and sweep* для идентификации и возврата мусорных объектов. Этот процесс включает в себя следующие шаги: пометить: сборщик мусора обходит все достижимые объекты и помечает их как живые. Зачистка: сборщик мусора просматривает кучу и разблокировывает все блоки памяти, которые не помечены как живые.

Виртуальная машина *Java* – та программа, которая выполняет байтовый код. Виртуальная машина находится в пакете *JDK* или *JRE*. Еще раз напоми­наю, что для разработки программ нужен пакет *JDK*, потому что с помощью пакета *JRE* можно только вьполнять программы *Java*[5, c. 37].

*JDK* (*Java Development Кit*) – набор инструментов разработки *Java* содер­жащий виртуальную машину, компилятор и другие средства разработки. *SDK* (*SoftwareDevelopment Кit*) – пакет инструментов разработки программного обеспечения. Это устаревшее название пакета *JDK*. На других платформах тер­мин «*SDK*» продолжает широко применяться. *JRE* (*Java Runtime Environment*) – средавыполнения *Java*. Это, собственно, и есть виртуальная машина *Java* [5, с. 44].

*JRE* – это платформенно-независимая виртуальная машина, то есть она предназначена для выполнения *Java*-программ на любой платформе, имеющей совместимую реализацию. *JRE* скрывает различия между разными аппаратными средствами и операционными системами, предоставляя согласованную среду выполнения для *Java*-программ.

*C*++ – это мощный язык программирования, который широко ис­пользуется в различных приложениях, включая системное программирование, разработку игр и высокопроизводительные вычисления.

*C*++ является объектно-ориентированным языком, что означает, что он позволяет разработчикам моделировать концепции и сущности реального мира в виде объектов. Объекты инкапсулируют данные и поведение и могут быть использованы для организации и структурирования кода. Этот язык поддерживает шаблоны, которые являются эффективным инструментом для обобщённого программирования. Они позволяют разработчикам создавать код, способный работать с различными типами данных, исключая необходимость дублирования.

*C*++ дает разработчикам контроль над управлением памятью, что может быть обоюдоострым мечом. С одной стороны, это позволяет эффективно и низкоуровневое манипулировать памятью, но с другой стороны, это может приве­сти к утечкам памяти и другим проблемам при неправильном обращении. *C*++ использует указатели для прямого манипулирования памятью и создания эф­фективных структур данных. Однако указатели могут быть сложны в правиль­ном использовании и могут привести к таким проблемам, как ошибки нулевого указателя и утечки памяти.

*C*++ поддерживает множественное наследование, что означает класс наследуется от нескольких базовых классов. Это может быть полезно для со­здания сложных иерархий объектов, но также может привести к таким пробле­мам, как бриллиантовое наследование и двусмысленность.

В *C*++ нет встроенного сборщика мусора, как в некоторых других языках, таких как *Java* или *C*#. Однако существуют некоторые библиотеки и фрейм­ворки, которые обеспечивают функциональность, подобную сборщику мусора в *C*++. Эти библиотеки не являются частью стандартной библиотеки *C*++, но их можно использовать для автоматического управления памятью и облегче­ния части бремени ручного управления памятью.

Одним из примеров такой библиотеки является сборщик мусора *Boehm-Demers-Weiser* (*BDW*), который представляет собой стороннюю библиотеку, обеспечивающую сборку мусора для программ на *C* и *C*++. Сборщик мусора *BDW* использует стандартный алгоритм сборки мусора, что означает, что он мо­жет собирать объекты, которые явно не помечены как мусор. Это может сде­лать библиотеку более простой в использовании, но также может привести к некоторым накладным расходам и ограничениям производительности. Другим примером является библиотека *C*++ *Garbage* *Collection* (*CGC*), которая явля­ется еще одной сторонней библиотекой, обеспечивающей сборку мусора для программ на *C*++. *CGC* использует гибридный подход, сочетающий подсчет ссылок и сборку мусора, что в некоторых случаях может обеспечить лучшую производительность, чем чистая сборка мусора.

Стоит отметить, что хотя эти библиотеки могут облегчить управление па­мятью в *C*++, они не являются идеальной заменой полноценному сборщику мусора, как в некоторых других языках. Разработчикам *C*++ по-прежнему необходимо знать о проблемах управления памятью и быть осторожными, чтобы избежать утечек памяти, висячих указателей и других проблем, которые могут возникнуть при работе с ручным управлением памятью.

*Python* ­ это высокоуровневый язык программирования, который был разработан Гвидо ван Россумом и впервые выпущен в 1991 году. Он стал одним из самых популярных языков программирования благодаря своей простоте, читабельности и универсальности. *Python* поддерживает несколько парадигм программирования, включая объектно-ориентированное, процедурное и функциональное программирование, что делает его подходящим для решения широкого спектра задач. Динамические языки обычно медленнее, чем статические, но их скорость повышается, поскольку интерпретаторы становятся более оптимизированными. Долгое время динамические языки использовались для коротких программ (сценариев), которые часто предназначались для того, чтобы подготовить данные для обработки более длинными программами, написанными на статических языка[4, с. 36].

*Python* интуитивно понятен и легко читаем, что позволяет разработчикам быстро осваивать язык и сосредотачиваться на решении задач, а не на сложности кода. Это делает *Python* идеальным выбором для начинающих программистов и образовательных учреждений. Данный язык программирования используется в различных областях, включая веб-разработку, научные вычисления, анализ данных, искусственный интеллект, автоматизацию и многое другое. Это делает его одним из наиболее универсальных языков программирования.

*Python* поставляется с обширной стандартной библиотекой, которая включает в себя модули для работы с сетью, файлами, базами данных, текстом и многими другими задачами. Это позволяет разработчикам использовать готовые решения, ускоряя процесс разработки.

*Python* поддерживается на различных операционных системах, включая *Windows*, *macOS* и *Linux*, что позволяет разработчикам создавать кроссплатформенные приложения.

Рассматриваемый язык программирования имеет обширные библиотеки для машинного обучения, такие как *TensorFlow*, *Keras* и *PyTorch*, делают *Python* одним из наиболее популярных языков для разработки моделей искусственного интеллекта и машинного обучения. *Python* часто используется для написания скриптов, которые автоматизируют повседневные задачи, такие как обработка данных, управление файлами и взаимодействие с *API*.

## **1.2 Выбор технологии работы с графикой**

*OpenGL* поддерживается в различных операционных системах, включая *Windows*, *Mac* и *Linux*, что означает то, что разработчики могут написать код один раз и запустить его на нескольких платформах без существенных изменений. Так же данное *API* существует и для мобильных устройств. *OpenGL* для встроенных систем (*OpenGL* *ES*) – это *API* для продвинутой 3*D*-графики, рассчитанный на мобильные и встроенные устройства[3, c. 23].

*OpenGL* хорошо настраивается, и разработчики имеют контроль над конвейером рендеринга, что позволяет им создавать узкоспециализированные эффекты и оптимизировать производительность для конкретного оборудования.

Графическое *API OpenGL* разработан для максимального использования аппаратного ускорения, что означает, что он может эффективно использовать вычислительную мощность видеокарты для быстрого рендеринга сложных сцен. *OpenGL* предлагает широкий спектр функций, включая 2*D* и 3*D* рендеринг, наложение текстур, освещение и затенение, среди прочего. *OpenGL* существует уже несколько десятилетий, и существует обширное сообщество разработчиков и доступных ресурсов, что облегчает поиск помощи и документации в случае необходимости. Для математики используется специальная библиотека. Одним из таких решений является библиотека *GLM*, написанная Кристофом Риччио[2, c. 25]. Она разработана на основе спецификации *GLSL*, поэтому синтаксис ее функций очень похож на поддержку математических операций в *GLSL* [2, c. 25].

Данная библиотека может быть интегрирован с другими языками программирования и библиотеками, включая *C*++, *Java*, *Python*, *С*# и другие, что делает его универсальным выбором для разработчиков.

*API OpenGL* используется в *CAD*-приложениях, требующих высококачественного рендеринга сложных 3*D*-моделей. *OpenGL* используется в приложениях виртуальной и дополненной реальности для визуализации реалистичных 3*D*-окружений, с которыми могут взаимодействовать пользователи. *OpenGL* используется в приложениях научной визуализации для создания интерактивных и информативных визуализаций сложных наборов данных. *OpenGL* может использоваться для разработки мультимедийных приложений, таких как видеоплееры и редакторы изображений.

Хотя *OpenGL* имеет много преимуществ как инструмент разработки, есть и некоторые недостатки, которые следует учитывать. Данное *API* имеет высокий порог вхождения для обучения, особенно для разработчиков, которые являются новичками в программировании графики. Может потребоваться время, чтобы освоить *API* и научиться эффективно его использовать. *OpenGL* является низкоуровневым *API*, что означает, что для создания сложных приложений разработчикам необходимо писать больше кода и хорошо разбираться в концепциях графического программирования.

Отсутствие встроенных функций. Хотя *OpenGL* предоставляет широкий спектр функциональных возможностей, он не содержит многих встроенных функций, таких как обнаружение столкновений или моделирование физики, что означает, что разработчикам необходимо реализовывать эти функции самостоятельно или использовать сторонние библиотеки.

*OpenGL* является кроссплатформенным, могут возникнуть некоторые проблемы, связанные с конкретной платформой, например, несовместимость драйверов, что может быть трудно отлаживать. *OpenGL* предназначен для высокопроизводительного рендеринга графики, в некоторых ситуациях он может работать не так хорошо, как другие графические *API*, например, при рендеринге большого количества объектов или сложных сцен.

*DirectX* – это набор интерфейсов прикладного программирования (*API*), которые призваны помочь разработчикам создавать мультимедийные приложения, включая видеоигры, для *Microsoft* *Windows*. Впервые он был представлен в 1995 году и с тех пор стал ключевым инструментом в разработке игр и других мультимедийных приложений для операционной системы *Windows*. *DirectX* предоставляет *API*, позволяющих разработчикам получать доступ к аппаратным средствам компьютера, таким как видеокарта, звуковая карта и устройства ввода. Это позволяет разработчикам создавать высокопроизводительные игры и другие мультимедийные приложения, которые в полной мере используют возможности аппаратного обеспечения компьютера.

Одним из главных преимуществ *DirectX* как инструмента разработки является его кросс-совместимость. Разработчики могут использовать одни и те же *API* для широкого спектра аппаратных платформ и операционных систем, что позволяет им создавать игры и другие мультимедийные приложения, которые можно воспроизводить на различных устройствах.

*DirectX* также включает ряд библиотек, которые помогают разработчикам решать такие задачи, как рендеринг графики, обработка ввода и воспроизведение звука. Эти библиотеки предоставляют готовые функциональные возможности, которые можно легко включить в игры и другие мультимедийные приложения, сокращая время и усилия, необходимые для разработки этих приложений с нуля.

Помимо кроссплатформенных возможностей и готовых библиотек, *Di-rectX* постоянно развивается, чтобы соответствовать новейшим технологиям аппаратного и программного обеспечения. Это означает, что разработчики могут воспользоваться последними достижениями в области компьютерного оборудования и программного обеспечения для создания еще более захватывающих и высокопроизводительных игр и мультимедийных приложений.

В целом, *DirectX* – это гибкий инструмент разработки, который стал неотъемлемым компонентом разработки игр для *Windows*. Его кроссплатформенные возможности, готовые библиотеки и постоянное развитие делают его популярным выбором для разработчиков игр, которые хотят создавать высокопроизводительные игры, в полной мере использующие возможности современного компьютерного оборудования.

Графический конвейер *DirectX* – это ряд шагов, которые использует для вывода графики на экран. Оптимизация этого конвейера может помочь повысить производительность приложений *DirectX*. Разработчики могут использовать инструменты профилирования для выявления узких мест и соответствующей оптимизации конвейера.

Приложениям *DirectX* часто требуется большой объем памяти для хранения текстур, моделей и других активов. Внедрение эффективных методов управления памятью, таких как кэширование и объединение ресурсов, может помочь сократить использование памяти и повысить производительность.

*HLSL* шейдеры – это инструмент для создания продвинутых визуальных эффектов в приложениях *DirectX*. Разработчики должны потратить время на изучение и освоение *HLSL*, чтобы создавать различные эффекты, графику в своих играх и приложениях.

*DirectX* 12 *Ultimate* является последней версией *DirectX* и предоставляет дополнительные возможности, такие как затенение с переменной скоростью и трассировка лучей. Разработчикам следует рассмотреть возможность использования *DirectX* 12 *Ultimate*, чтобы воспользоваться преимуществами этих новых функций и создать более захватывающие игры и приложения.

Хотя на первый взгляд *DirectX* представляется идеальным инструментом для разработки игр и мультимедийных приложений, у него всё же имеются некоторые потенциальные недостатки. *DirectX* – это специфичная технология для *Microsoft* *Windows*, что означает то, что игры и приложения, разработанные с ее помощью, будут работать только на устройствах на базе *Windows*. Это может ограничить потенциальную аудиторию игры или приложения.

*DirectX* может быть сложным и трудным для изучения новичками. Он требует глубокого понимания компьютерной графики и программирования, и для новичков в этих областях кривая обучения может оказаться крутой. *DirectX* зависит от оборудования, а это значит, что некоторые старые или менее мощные аппаратные средства не смогут запускать игры или приложения, разработанные с использованием последней версии *DirectX*. Это может ограничить потенциальную аудиторию игры или приложения. *DirectX* в первую очередь предназначен для настольных компьютеров и ноутбуков и имеет ограниченную поддержку мобильных устройств, таких как смартфоны и планшеты. Это может ограничить потенциальную аудиторию игр или приложений, предназначенных для мобильных платформ. Хотя сам *DirectX* является бесплатным для использования, некоторые программные инструменты и библиотеки, которые обычно используются вместе с ним, например *Microsoft* *Visual* *Studio*, могут иметь лицензионные расходы, связанные с ними. Это может увеличить общую стоимость разработки игр и приложений с *DirectX*.

В целом, хотя *DirectX* является мощным и гибким инструментом разработки, существуют некоторые потенциальные недостатки работы с ним. Разработчикам следует внимательно изучить эти ограничения, прежде чем принимать решение об использовании *DirectX* в своих проектах.

## **1.3 Перечень используемых технологий**

При рассмотрении предлагаемых программных решений применимо задаче, выделяется ряд инструментов, наиболее подходящих к использованию в игровом приложении.

Список предлагаемых инструментов включал в себя:

– графические библиотеки: *DirectX* и *OpenGL*;

– языки программирования: *C*# .*NET*, *Java*, *С*++.

В течении всего аналитического обзора сравнивались языки программирования и графические библиотеки.

В качестве графики был выбран *OpenGL* из-за простоты нахождения информации работы с данной библиотекой и кроссплатформенностью. Языком программирования был выбран C# за его высокую производительность, наличие четкой и отличной документации и высокая популярность языка.

Список выбранных инструментов включает в себя:

*– OpenGL* *API* (*OpenTK*) главный инструмент для работы с графикой;

*– C*# .*NET* как язык программирования, на котором будет разработана основная часть игрового приложения.

# **2 ПРОЕКТИРОВАНИЕ АРХИТЕКТУРЫ И АЛГОРИТМОВ ДЛЯ СЕТЕВОГО ПРИЛОЖЕНИЯ «ТАНКОВАЯ ДУЭЛЬ»**

## **2.1 Архитектура приложения**

Архитектура сетевого приложения включает в себя *front*-*end* и *back*-*end*. *Front*-*end* предоставляет пользовательский интерфейс для взаимодействия с программным продуктом. Эта часть выполняет функции отображения и управления, предоставляя доступ к функциям и данным приложения. Также может обрабатываться часть бизнес-логики, если это необходимо для улучшения пользовательского опыта. *Back*-*end* отвечает за обработку данных, хранение информации и выполнение бизнес-логики. В этой части приложения расположены серверы и базы данных, а также реализуются функции безопасности и аутентификации. Приложения бывают разных типов:

– клиент-серверные;

– одноранговые;

– облачные;

– микросервисные.

В клиент-серверных приложениях обрабатываются запросы клиентов, например, почтовыми серверами, веб-серверами, файловыми серверами. Клиентское устройство отправляет запросы на сервер и получает от него данные. Существует два вида клиент-серверных архитектур: двухуровневая и многоуровневая. В двухуровневой архитектуре клиентское приложение взаимодействует напрямую с сервером, который обрабатывает запросы и отправляет данные обратно клиенту. В многоуровневой архитектуре между клиентом и сервером могут быть добавлены дополнительные промежуточные уровни, такие как уровень приложений, базы данных и уровень бизнес-логики, что способствует распределению задач обработки и улучшению масштабируемости и производительности приложения.

В одноранговой (*P*2*P*) архитектуре все узлы сети являются равноправными и могут как отправлять, так и получать данные. Такие системы эффективны для обмена файлами и распределенных вычислений, например, системы обмена файлами (*BitTorrent*) или распределенные базы данных. В этой модели узлы сети являются равноправными и могут выполнять функции отправки и получения данных.

Облачные приложения размещаются на удаленных серверах и предоставляются пользователям через интернет, обеспечивая высокую доступность и возможность работы с данными и файлами независимо от устройства.

Микросервисные приложения делятся на множество мелких сервисов, задачи которых выполняются и взаимодействуют с другими сервисами по сети. Каждый микросервис может разрабатываться и разворачиваться независимо, что упрощает масштабирование и улучшает устойчивость к сбоям. Приложение также может классифицироваться по назначению, что напрямую влияет на выбор архитектуры. В зависимости от назначения приложения выделяются следующие типы:

– онлайн редакторы;

– корпоративные системы;

– мессенджеры и средства связи;

– игровые приложения;

– файловые хранилища и обменники.

Способ связи играет важнейшую роль, определяя взаимодействие приложения и пользователя. Данный аспект влияет на то, каким будет *front*-*end* и часть *back*-*end*. Выделяются три основных способа связи. Синхронный способ требует немедленного ответа от сервера клиенту, асинхронный способ не требует мгновенного отклика от сервера, приложения реального времени и кэшируемые приложения. Приложения реального времени предоставляют информацию, приходящую с минимальной задержкой и практически в реальном времени. Кэшируемые или офлайн-доступные приложения предназначены для хранения и обмена файлами.

В качестве общей архитектуры приложения выбрана клиент-серверная модель, в которой функции распределяются между клиентом и сервером. На рисунке 2.1 изображен пример алгоритма работы игрового приложения без сетевой составляющей.



Рисунок 2.1 ­ Пример алгоритма работы игрового приложения

В данном случае реализуется архитектура для онлайн-игры, предполагающая взаимодействие клиента с сервером, обрабатывающим игровые события и обеспечивающим синхронизацию между игроками. Используется синхронный способ связи.

На рисунке 2.1 представлена инициализация объектов. Под инициализацией подразумевается загрузка объектов и определение типов игровых объектов. Следующий этап включает загрузку объектов в игровой движок и определение объектов, которые необходимо рендерить. Третий этап включает инициализацию скриптов, представляющих собой компоненты, прикрепляемые к игровым объектам для управления ими или взаимодействия с другими игровыми объектами. После завершения трех этапов запускается игровой цикл, состоящий из двух этапов. Цикл состоит из двух этапов. На первом этапе игрового цикла проверяются коллизии и триггеры объектов, а также происходит просчет физики для правильного определения столкновений игровых объектов. После коллизий и триггеров выполняются скрипты объектов и обрабатывается пользовательский ввод. Процесс продолжается до прекращения цикла.

Комбинация двух подходов – клиент-серверного и архитектуры обычной однопользовательской игры – позволяет масштабировать приложение и добавлять новую функциональность и модули. На рисунке 2.2 представлен доработанный алгоритм работы игрового приложения.



Рисунок 2.2 – Алгоритм работы разрабатываемого приложения

В общих чертах представляется обычное игровое приложение. Процесс включает инициализацию, проверку коллизий, выполнение скриптов. Добавляются два новых модуля:

– прием состояний клиентов (игроков);

– отправка состояний от сервера.

Данные поступающие от игроков обрабатываются модулем. Модуль может включать информацию о текущем положении объекта, действиях игрока (прыжки, удары, движение) или других игровых событиях. После обработки сообщения клиента происходит обновление состояния игры. Уже после обновления состояния игры сервер отправляет это состояние всем подключенным игрокам.

Также следует понимать, что игровой цикл и состояние игроков обновляются асинхронно. Обработка входящих данных от клиентов осуществляется независимо от основного игрового цикла. Это позволяет обрабатывать каждое сообщение игрока сразу после его получения, не ожидая завершения текущего этапа игрового цикла. Обновления игрового состояния могут поступать клиентам без ожидания полного завершения обработки на сервере. Это уменьшает сетевые задержки и делает игровой процесс более плавным.

## **2.2 Модели синхронизации пользователей и сервера в игровых приложениях**

Синхронизация состояний представляет собой важную часть онлайн-игр, от которой зависит как игровое восприятие, так и игровой баланс. Используются два концептуальных подхода для синхронизации клиента и сервера:

* *Client*-*Side* *Prediction*;
* *Server*-*Authoritative* *Model*.

*Client*-*Side* *Prediction* представляет собой способ предсказания на клиенте части игровых событий, таких как анимации, передвижения игрока, взаимодействия объектов. На рисунке 2.3 изображен пример интерполяции.

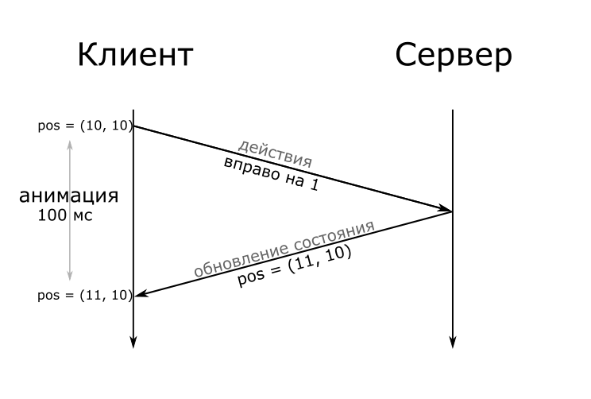


Рисунок 2.3 – Пример интерполяции передвижения игрока

Этот подход позволяет разгрузить часть задач на сервере. Обеспечивается более плавное отображение картинки, так как клиенту требуется лишь визуализировать состояние, полученное от сервера, без необходимости выполнять сложные вычисления, такие как физика игрового мира и другие ресурсоемкие операции. Сервер обрабатывает и отправляет ограниченное количество состояний в единицу времени, что характеризуется параметром, называемым тикрейтом. Чтобы игровой процесс не происходил с малым количеством кадров, состояния интерполируются.

Как показано на рисунке 2.3, вычисляются промежуточные клиентские состояния между отправками сервера. Когда сервер отправляет конечный результат, сравниваются состояния клиента и сервера. Если разница невелика, состояние клиента не меняется, если разница превышает пороговое значение, применяется серверное состояние. Такой подход имеет как плюсы, так и минусы.

*Server*-*Authoritative* *Model* – это архитектурный подход в разработке многопользовательских игровых приложений, при котором полностью контролируется все аспекты игрового процесса сервером. В этой модели выполняются роль «тонких клиентов» устройства, которые только отправляют запросы серверу и отображают полученные данные, но не принимают ключевых решений о состоянии игрового мира.

Происходит монополизация всех событий, связанных с миром. Состояние игрового мира не может изменяться клиентами самостоятельно. Все изменения обрабатываются через сервер. Исключается возможность использования вредоносных программ, основанных на манипуляциях с локальными данными. Все игровые расчеты выполняются на сервере. Это включает:

* физика;
* проверку правил игры (например, доступность атак или действий);
* сохранение прогресса (например, инвентарь, позиции игроков).

Взаимодействие клиента и сервера осуществляется асинхронно. Команды отправляются на сервер, который обрабатывает их и возвращает результат. Состояние обновляется на основе полученных данных. Этапы взаимодействия:

* отправка команды от клиента;
* проверка действия сервером;
* обновление состояния мира;
* отправка обновленного состояния клиентам.

Команда отправляется на сервер для выполнения действий(например, перемещение персонажа, выстрел, использование предмета). Команда проверяется сервером на соответствие правилам игры и текущим состоянием мира. Команда перемещения проверяется на возможность перемещения, команда взаимодействия проверяется на возможность использования тех или иных ресурсов игрока. Если команда валидна, состояние сервера обновляется. После чего отправляется новое состояние игры клиентам.

По итогу обзора моделей синхронизации, для игры «танковая дуэль», была выбрана модель *Server*-*Authoritative* *Model*. Она является более справедливой в отношении к игрокам.

## **2.3 Алгоритмы обработки коллизий**

Существуют различные методы обнаружения коллизий между объектами. Наиболее распространенными являются *AABB* (*Axis*-*Aligned* *Bounding* *Box*), коллизия окружности или сферы и *OBB* (*Oriented* *Bounding* *Box*). Наиболее простыми в реализации являются *AABB* и сферическая коллизия. У *AABB* возникают проблемы при повороте объекта в пространстве, когда, например, при вращении объекта вокруг своей оси, коллизия меняет свой размер и не соответствует исходному объекту. На рисунке 2.4 изображена эта особенность.

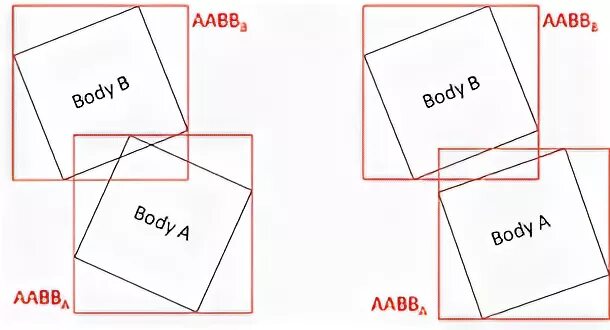


Рисунок 2.4 – Метод обнаружения коллизия *AABB*

Алгоритм *OBB* лишен данного недостатка. Для расчета используется теорема о разделяющей оси. Теорема гласит, что если существует ось, на которую проекции двух объектов не пересекаются, то эти объекты не пересекаются. Если же проекции пересекаются на всех возможных осях, то объекты сталкиваются. На рисунке 2.5 изображен пример.

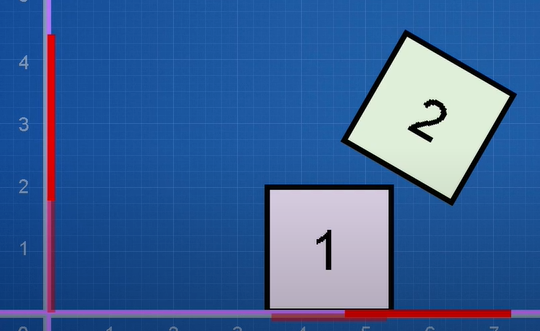


Рисунок 2.5 – Пример теоремы о разделяющей оси

Данный подход является более трудоемким в вычислениях, чем *AABB* и сферические коллайдеры, но более точно определяет коллизии для не сферических объектов.

Помимо определения коллизий в играх существуют технические нюансы, которые могут вызывать критические ошибки. Для минимизации таких ошибок используются специальные алгоритмы. В расчетах движения объектов применяется дельта времени – интервал между предыдущим и текущим кадрами. Это значение позволяет плавно перемещать объекты. Однако метод эффективен только при умеренных скоростях и схожих размерах объектов. При высокой скорости или значительных различиях в размерах объекты могут проходить сквозь друг друга. Решение этой проблемы заключается в бросании луча вдоль вектора движения объекта или создании цилиндров, соответствующих дельте перемещения, для проверки наличия пересечений. После этих проверок объект корректно перемещается на заданное расстояние.

## **2.4 Архитектура сети**

Архитектура сети зависит от архитектуры разрабатываемого приложения. Поскольку за основу была взята клиент-серверная архитектура, взаимодействие клиента с сервером может быть двухуровневым, когда клиент отображает то, что сервер прислал, а сервер обрабатывает команды и решает, применять их или нет, или многоуровневым, когда клиент обращается к узлу, а тот уже обращается к серверу или выполняет другие операции и возвращает ответ клиенту.

Для многопользовательских онлайн-игр характерна многоуровневая архитектура сети, так как для таких приложений важнейшей задачей является масштабируемость. Такой подход также облегчает распределение нагрузки между несколькими серверами. Для данной задачи избыточной будет такая архитектура, так как нет необходимости распределять нагрузку на разные серверы для двух игроков. Следовательно, многоуровневая архитектура не подходит. На рисунке 2.6 изображена архитектура сети приложения «Танковая дуэль».

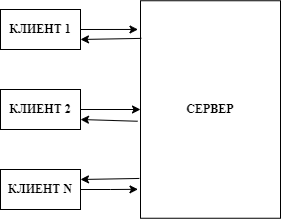


Рисунок 2.6 – Архитектура сети

Как видно из рисунка 2.6, отправляется сообщение от клиента серверу, и сервер возвращает ответ. Сервер может находиться как в локальной сети с клиентами, так и в глобальной сети, где клиенты могут обращаться по «белому» *IP*-адресу. В данном случае ответом от сервера является состояние игры. Несмотря на то, что игра рассчитана на двух игроков, такая архитектура позволяет обрабатывать столько пользователей, сколько позволяет производительность сервера и пропускная способность сетевого оборудования.

## **2.5 Выбранные архитектурные и алгоритмические решения**

В процессе анализа были тщательно изучены различные архитектурные подходы и алгоритмы, применимые к созданию сетевого игрового приложения и его ключевых функциональных компонентов.

Был сделан выбор в пользу клиент-серверной архитектуры, адаптированной для нужд игрового приложения. Этот подход обеспечивает разделение ответственности между клиентом и сервером, что упрощает управление состояниями игроков и синхронизацию игрового мира.

Для синхронизации игрового приложения выбран алгоритм *Server*-*Authoritative* *Model*, который гарантирует выполнение всех ключевых вычислений и проверок состояния игры на стороне сервера. Это минимизирует возможность нечестной игры и обеспечивает единообразие игрового процесса для всех участников.

В качестве метода обработки коллизий выбран алгоритм *AABB* (*Axis*-*Aligned* *Bounding* *Box*), который представляет собой простой и эффективный способ проверки пересечений между объектами. Дополнительные проверки столкновений не добавляются, так как *AABB* полностью удовлетворяет требованиям текущего проекта.

Такое сочетание архитектуры и алгоритмов обеспечивает высокую производительность, стабильность и удобство в разработке игрового приложения.

# **3 ОПИСАНИЕ ПРОГРАММЫ И РАЗРАБОТАННЫХ АЛГОРИТМОВ ДЛЯ ИГРЫ «ТАНКОВАЯ ДУЭЛЬ»**

## **3.1 Описание главных клиентских классов**

Основными клиентскими классами являются те, что рисуют объекты и интерфейс, обрабатывают клиентских ввод, обеспечивают связь с сервером. Основные классы:

* *ConectionManager*;
* *RenderObjects*;
* *GameClient*;
* *SingeltonEngine*;

В классе *ConectionManager* из пространства имен *NetWorkConnector* осуществляется управление соединением между клиентом и сервером в сетевой игре. Основные аспекты включают установку соединения, отправку и получение данных, а также обновление состояния игры. Конструктор класса создает экземпляр сокета (*socketPlayer*) и инициализирует объект *Server* для работы с сервером. Методы класса *ConectionManager*:

* *SetConnection*(*IPEndPoint* *hostEndPoint*);
* *Player*;
* *AddState*;
* *SendState*.

Соединение устанавливается с сервером через указанный *IPEndPoint* (адрес и порт сервера) при помощи метода *SetConnection*. Запускается новый поток для выполнения метода *Player*.

Метод *Player* выполняет обработку игрового процесса. Получает начальное состояние игры от сервера через *server*.*ReciveOnStart*(). В бесконечном цикле выполняется проверка доступности данных от сервера, обновление состояния игры на основе полученных данных. Вызываются методы обновления состояния клиента, рендера экрана и обработки ввода игрока. Ответственность за отправку текущего состояния на сервер через метод *SendState*. Используется задержка *Task*.*Delay*(10) для ограничения скорости выполнения. Объект состояния добавляется при помощи метода *AddState*(*StateObject* *stateObject*), чтобы он был отправлен на сервер. Список состояний очищается в методе *SendState*.

Сетевое взаимодействие управляется при помощи класса *ConectionManager*. Входящие данные обрабатываются от сервера (например, обновления состояния игры) и интерпретируются для приложения. Что бы отправить состояние, оно должно быть добавлено в очередь. Состояния добавляются при помощи классов тип *MonoBehaivour*, к примеру *PlayerTankScript*. Затем принимается решение менеджером об отправке или не отправке состояния из очереди.

Класс *TimeDeltaHelper* используется для управления временем в игровой логике, расчета времени между кадрами (*DeltaT*), регулировки скорости времени (*TimeSpeed*) и измерения *FPS* (кадров в секунду).

Методы класса:

* *Start*;
* *Update*.

Класс *GameClient* используется для управление состоянием игрового клиента, обработку игровых объектов, обновление игрового состояния и взаимодействие с событиями игрового цикла. Методы для добавления, удаления и получения игровых объектов, а также для обновления состояния игры предоставляются данным классом.

Методы класса:

– *GameState*();

– *UpdateState*(List<*StateObject*> *stateObjects*);

– *GetObjectByID*(*int* *ID*);

– *GetGameObjectByName*(*string* *name*);

– *RemoveListeners*(*MonoBehaivour* *monoBehaivour*);

– *AddGameObject*(*GameObject* *gameObject*);

– *RemoveGameObject*(*GameObject* gameObject);

– *AddListenerToUpdate*(*MonoBehaivour* *monoBehaivour*);

– *ClearObjects*().

*GameState*() используется для инициализации игрового состояния, вызова обработчиков запуска и обновления, а также очистки объектов. *UpdateState*(*List*<*StateObject*> *stateObjects*) выполняет обновление состояния игры на основе переданных объектов состояния. *GetObjectByID*(*int* *ID*) позволяет получить игровой объект по его идентификатору. *GetGameObjectByName*(*string* *name*) предназначен для получения игрового объекта по его имени. *RemoveListeners*(*MonoBehaivour* *monoBehaivour*) удаляет слушателей событий для указанного объекта. *AddGameObject*(*GameObject* *gameObject*) добавляет новый объект в игровой цикл. *RemoveGameObject*(*GameObject* *gameObject*) помечает объект для удаления. *AddListenerToUpdate*(*MonoBehaivour* *monoBehaivour*) добавляет слушателей обновлений и запусков для указанного объекта. *ClearObjects*() очищает объекты, помеченные для удаления, и удаляет соответствующих слушателей.

Класс *RenderObjects* используется для управления рендерингом игровых объектов и интерфейса пользователя в 2*D* с использованием *OpenTK*. Классом обрабатывается отображение объектов на экране, их текстуры, а также события, связанные с изменением размера окна и обновлением экрана.

Методы класса:

– *RenderObjects*(*GLControl* *gLControl*, *Form* *form*);

– *AddToRender*2*DUI*(*UserInterface*2*D* *interface*2*D*);

– *RemoveFromRender*2*DUI*(*UserInterface*2*D* *userInterface*2*D*);

– *RenderCicle*(*object* *sender*, *PaintEventArgs* *eventPaint*);

– *RenderUI*2*D*(*UserInterface*2*D* *userInterface*2*D*);

– *UpdateTextTexture*(*string* *text*, *Font* *font*, *int* *textureId*, *int* *width*, *int* *height*);

– *RenderObject*(*GameObject* *gameObject*);

– *ScreenPosition*();

– *Control*\_*Resize*(*object* *sender*, *EventArgs* *e*);

– *UpdateScreen*();

– *OnOfRender*(*bool* *isWork*).

*RenderObjects*(*GLControl* *gLControl*, *Form* *form*) используется для инициализации объекта рендеринга с использованием *OpenGL*-контрола и формы, так же привязываются события для рендеринга и изменения размера окна. *AddToRender*2*DUI*(*UserInterface*2*D* *interface*2*D*) добавляет элемент *interface*2*D* в список для рендеринга. *RemoveFromRender*2*DUI*(*UserInterface*2*D* *userInterface*2*D*) удаляет элемент *userInterface*2*D* из списка рендеринга. В методе *RenderCycle*(*object* *sender*, *PaintEventArgs* *eventPaint*) выполняются четыре этапа: первый этап очищает экран, второй устанавливает проекцию и матрицы для 2*D*-рендеринга, третий рендерит все игровые объекты из списка \_*gameObjects*, последний рендерит все элементы *UI* из списка \_*userInterface*2*D*.

Класс *SingeltonEngine* используется для управления и инициализацией различных компонентов игрового движка. Классом реализуется шаблон синглтоном. Благодаря этом предоставляется глобальный и единый доступ к объектам, таким как загрузчики, менеджеры соединений, рендеринг и игровая логика.

Методы класса:

* *SetGameClient*(*GameClient* *client*);
* *SetLoaderTextures*(*LoaderTexture* *loaderTexture*);
* *SetObjectLoader*(*ObjectLoader* *loader*);
* *SetRender*(*RenderObjects* *render*);
* *SetConectionMenager*(*ConectionManager* *conectionManager*).

*SetGameClient* (*GameClient* *client*) – в методе инициализируется игровой клиент, если он еще не был установлен, устанавливается флаг *isSetGameClient* в *true*. *SetLoaderTextures* (*LoaderTexture* *loaderTexture*) – инициализирует загрузчик текстур, если он еще не был установлен, устанавливает флаг \_*isSetLoadTextures* в *true*. *SetObjectLoader* (*ObjectLoader* *loader*) – в методе происходит инициализация загрузчика объектов, если он еще не был установлен, устанавливается флаг \_*isSetLoader* в *true*. *SetRender* (*RenderObjects* *render*) – в методе инициализируется рендерер объектов, если он еще не был установлен, устанавливается флаг \_*isSetRendering* в *true*. *SetConectionMenager* (*ConectionManager* *conectionManager*) – в методе устанавливается менеджер соединений.

## **3.2 Описание главных серверных классов**

Для сервера основное внимание уделяется классам, отвечающим за логику игры, физические расчёты, обработку коллизий и выполнение скриптов. Также значимую роль играют классы, обеспечивающие интернет-соединение и обмен данными.

* *GameEngine*;
* *Collision*;
* *SimpleCollision*;
* *GameState*;
* *MonoBehaivour*.

*GameEngine* класс реализует движок игры, отвечающий за управление игровыми объектами, обработку коллизий, события игрового цикла, а также управление многопоточностью для обеспечения производительности.

Методы класса:

* *GameEngine*(*ObjectLoader objectLoader*);
* *GetGameObjectByID*(*int* *ID*);
* *GetGameObjectByName*(*string* *name*);
* *AddObject*(*GameObject* *gameObject*);
* *RemoveListener*(*MonoBehaivour* *monoBehaivour*);
* *AddListener*(*MonoBehaivour* *monoBehaivour*);
* *GetDataFromUser*(*object* *data*, *EventArgs* *eventArgs*);
* *Restart*();
* *WorkCycle*();
* *ProcessCollisionAsync*(*GameObject* *gameObject*, *List*<*GameObject*> *colList*);
* *CheckCol*(*GameObject* *gameObject*, *List*<*GameObject*> *gameObjects*);
* *ClearObjects*();
* *AddObjectToDelete*(*GameObject* *gameObject*).

Конструктор *GameEngine*(*ObjectLoader* *objectLoader*) используется для инициализации игрового движка и запуска игрового цикла. Метод *GetGameObjectByID*(*int* *ID*) применяется для получения игрового объекта по его *ID*. Метод *GetGameObjectByName*(*string* *name*) используется для возвращения игрового объекта по его имени. Метод *AddObject*(*GameObject* *gameObject*) служит для добавления объекта в игру и связывания его событий. Метод *RemoveListener*(*MonoBehaivour* *monoBehaivour*) используется для удаления событий, связанных с данным скриптом. Метод *AddListener*(*MonoBehaivour* *monoBehaivour*) применяется для добавления событий, связанных с объектом. Метод *GetDataFromUser*(*object* *data*, *EventArgs* eventArgs) предназначен для обработки данных, поступающих от пользователя, и применения изменений к игровым объектам. Метод *Restart*() выполняет перезапуск игры с удалением всех текущих объектов. Асинхронный метод *WorkCycle*() отвечает за выполнение игрового цикла, включая обработку событий, коллизий и обновление объектов. Метод *ProcessCollisionAsync*(*GameObject* *gameObject*, *List*<*GameObject*> *colList*) применяется для асинхронной проверки коллизий указанного объекта. Метод *CheckCol*(*GameObject* *gameObject*, *List*<*GameObject*> *gameObjects*) используется для выполнения проверки коллизий заданного объекта. Метод *ClearObjects*() предназначен для удаления из игры объектов, помеченных для удаления. Метод *AddObjectToDelete*(*GameObject* *gameObject*) используется для пометки объекта для последующего удаления.

Класс *Collision* используется для обработки коллизий игровых объектов, включая определение их границ, обработку событий столкновений и триггеров. Методы поля:

* *Collision*(*GameObject* *gameObject*);
* *FoundMinPos*();
* *FoundMaxPos*();
* *MoveCollision*();
* *AddListener*(*MonoBehaivour* *monoBehaivour*);
* *RemoveListener*(*MonoBehaivour* *monoBehaivour*);
* *RemoveListener*();
* *CheckCollision*(*GameObject* *gameObject*);
* *Intersect*(*Vector*3 *min, Vector*3 *max, Vector*3 *selfMin*, *Vector*3 *selfMax*);
* *Invoke*(*GameObject* *gameObject*).

Конструктор *Collision*(*GameObject* *gameObject*) применяется для инициализации коллизии для объекта. Метод *FoundMinPos*() предназначен для вычисления минимальной границы объекта в 3*D*-пространстве. Метод *FoundMaxPos*() используется для вычисления максимальной границы объекта в 3*D*-пространстве. Метод *MoveCollision*() выполняет пересчет границ коллизии. Метод *AddListener*(*MonoBehaivour* *monoBehaivour*) добавляет обработчики событий столкновения и триггера. Метод *RemoveListener*(*MonoBehaivour* *monoBehaivour*) удаляет обработчики событий для указанного поведения. Метод *RemoveListener*() применяется для удаления всех обработчиков событий для текущего объекта. Метод *CheckCollision*(*GameObject* *gameObject*) используется для проверки пересечения границ текущего объекта и указанного объекта. Метод *Intersect*(*Vector*3 *min*, *Vector*3 *max*, *Vector*3 *selfMin*, *Vector*3 *selfMax*) определяет, пересекаются ли границы двух объектов. Метод *Invoke*(*GameObject* *gameObject*) вызывает события коллизии или триггера для текущего и указанного объектов.

Класс *SimpleCollision* используется для обработки простых коллизий игрового объекта с препятствиями. Обеспечивается проверка и корректировка позиции объекта при столкновении, а также управление его направлением и движением.

Методы класса:

– *TurnOfOn*();

– *SimpleCollision*(*GameObject gameObjects*);

– *AddImpact*(float velocity, Vector3 direction);

– *CorrectPosition*(*Vector*3*selfMin*, *Vector*3*selfMax*, *Vector*3*wallMin*, *Vector*3*wallMax*).

Конструктор *SimpleCollision*(*GameObject* *gameObjects*) применяется для инициализации обработки коллизий для указанного игрового объекта. Метод *TurnOfOn*() позволяет включить или выключить обработку коллизий. Метод *AddImpact*(*float* *velocity*, *Vector*3 *direction*) используется для добавления движения объекту с учетом заданной скорости и направления. Метод *CorrectPosition*(*Vector*3 *selfMin*, *Vector*3 *selfMax*, *Vector*3 *wallMin*, *Vector*3 *wallMax*) корректирует позицию объекта при столкновении с препятствием.

Класс *MonoBehaivour* используется для инициализации игрового поведения объекта, обработки событий, управления состоянием ввода и выполнения сериализации.

Основные методы класса:

* *MonoBehaivour*(*GameObject mesh*);
* *Start*(*object* *sender*, *EventArgs* *e*);
* *Update*(*object sender*, *EventArgs e*);
* *Colision*(*object sender*, *EventArgs e*);
* *Trigger*(*object sender*, *EventArgs e*);
* *IsCopingScript*();
* *SateNewInputState*(*PlayerInputController* *playerInputController*);
* *SereilizeScript*(*Type* *type*);
* *ReturnSatate*(*Type* *type*);
* *SateState*(*List*<*object*> *list*, *PlayerInputController* *playerInputController*);
* *RemoveScript*().

Конструктор *MonoBehaivour*(*GameObject* *mesh*) применяется для инициализации игрового поведения для указанного объекта. Метод *Start*(*object* *sender*, *EventArgs* *e*) вызывается при старте игрового объекта. Метод *Update*(*object* *sender*, *EventArgs* *e*) используется для выполнения операций в каждом кадре обновления. Метод *Collision*(*object* *sender*, *EventArgs* *e*) обрабатывает события столкновений. Метод *Trigger*(*object* *sender*, *EventArgs* *e*) отвечает за обработку событий триггеров. Метод *IsCopingScript*() проверяет, разрешено ли копирование полей скрипта. Метод *SetNewInputState*(*PlayerInputController* *playerInputController*) устанавливает новое состояние ввода для контроллера игрока. Метод *SerializeScript*(*Type* *type*) выполняет сохранение состояния скрипта путем сериализации его публичных полей. Метод *ReturnState*(*Type* *type*) возвращает сериализованное состояние скрипта. Метод *SetState*(*List*<*object*> *list*, *PlayerInputController* *playerInputController*) восстанавливает состояние скрипта из списка сохраненных значений. Метод *RemoveScript*() используется для удаления скрипта из списка поведения объекта и движка.

Классы *GameState*, *StateObject* и *StateScript* предназначены для сериализации и десериализации объектов, которые передаются по сети. Выполняется инкапсуляция полей передаваемых объектов. После отправки и приема данных выполняется обработка *JSON*-парсером.

## **3.3 Апробация игрового приложения «Танковая дуэль»**

После того, как все необходимые элементы игрового поля загружены, начинается игровой процесс. Игроки используют клавиатуру для управления своими танками на поле, уклоняются от снарядов и обходят препятствия, собирают различные бонусы, которые могут как помочь, так и навредить в битве с другим игроком. Что бы одержать победу необходимо что бы у игрока кончилось здоровье или же топливо. На рисунке Б.1 показан внешний вид танков, игрового поля. Управление осуществляется при помощи клавиатуры.

В игровом пространстве присутствуют различные бонусы: здоровье, скорость, урон, топливо, броня. Данные объекты являются бонусами. При наезди на бонус произойдет изменение характеристик машины игрока. Игрок поднявший данный предмет может получить как улучшение характери­стик, так и их ухудшение. Данные изменения бывают положительными или отрицательными. Интерфейс игроков представлен текстом, в котором отображается здоровье, топливо, урон. Текст меняется при нанесении урона, передвижении машины игрока, подборе бонусов.

## **3.4 Нагрузочные тесты для сервера**

Для нагрузки сервера будут использованы отправленные с клиента команды и обработанные на сервере. Тестовые параметры будут представлены количеством обработанных запросов и пропускной способностью. На рисунке 3.1 изображен часть графика, где точки – это принятые данные в конкретное время, на рисунке 3.3 изображены число принятых и обработанных сообщений.

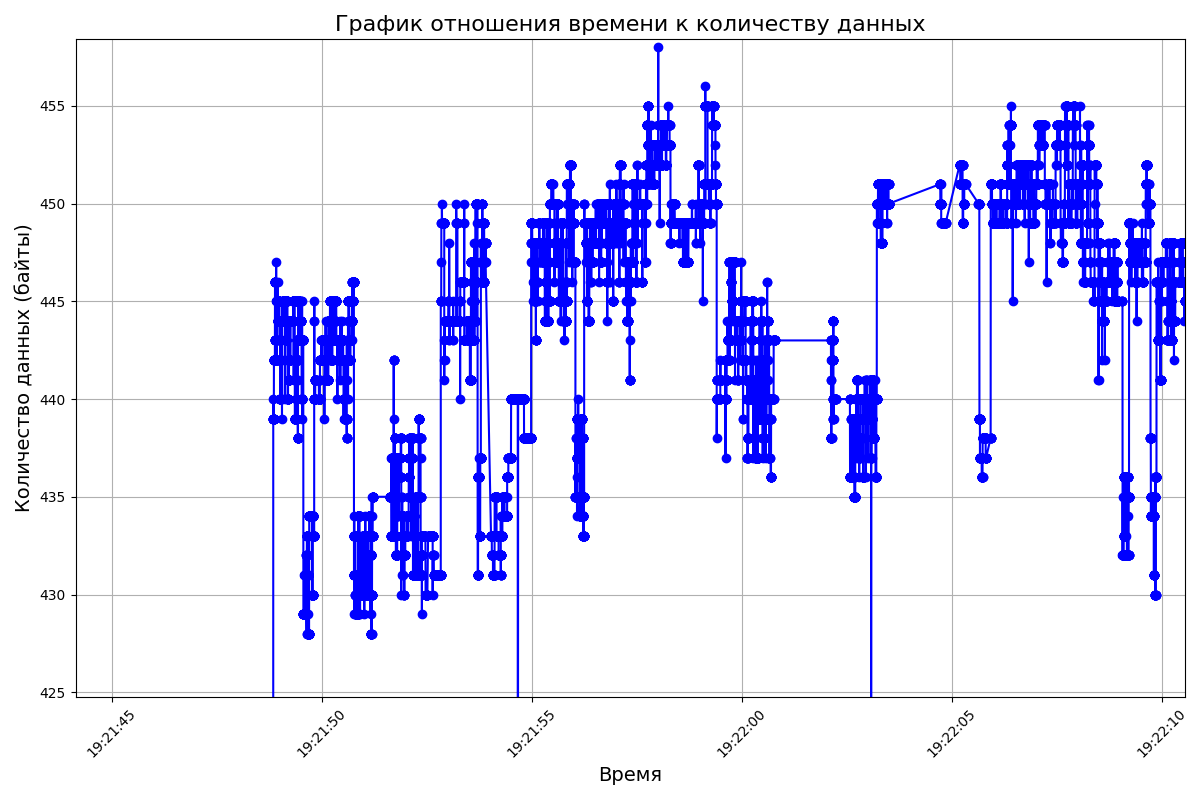


Рисунок 3.1 – Часть графика приема сообщений

Как видно из рисунка 3.1 данные варьируются от 430 и выше 450 байт. Зная начальное и конечное время можно посчитать пропускную способность сервера. На рисунке 3.2 представляются проведенные подсчеты.



Рисунок 3.2 – Пропускная способность сервера

Стоит отметить, что сервер имеет ограничение в количестве принимаемых сообщений. Это необходимо для меньшей нагрузки на сервер. Т.е. если команд от одного клиента было отправлено больше, чем *n*, где *n* это максимальное количество команд на одного игрока, то сервер будет игнорировать часть команд пока очередь не освободится. На рисунке 3.3 изображен вывод из консоли.



Рисунок 3.3 – Число принятых и обработанных сообщений

Согласно части изображения на рисунке 3.3, сервер обрабатывает около половины запросов от клиента. Размер очереди установлен на 30 сообщений, после чего вводится штраф по времени, пропорциональный размеру очереди (очередь делится на 1 секунду). Этот параметр зависит от производительности сервера. Текущая производительность позволяет обрабатывать 200–240 состояний в секунду, что оставляет значительный запас. В настоящее время сервер обрабатывает 120–140 команд от клиентов, что соответствует тикрейту 30–60. Увеличение очереди может повысить отзывчивость, но при текущей нагрузке это не требуется. Конфигурации сервера: процессор *I*7 7700*HQ*, 16 Gb *DDR*4. При данной конфигурации не было замечено зависаний и потерь управления на клиентах.

## **3.5 Модульные тесты**

Для тестирования приложения следует использовать модульные тесты. В среде *Visual* *Studio* имеется библиотека *UnitTests*. При помощи данной библио­теки можно тестировать классы и методы программы на их валидность.

На рисунке 3.2 изображены пройденные тесты.

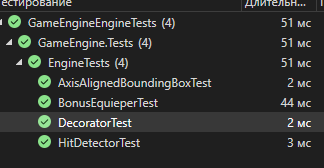


Рисунок 3.2 – Пройденные тесты для валидации методов в классе *GameEngine*

Данные тесты проверяют работоспособность методов коллизии танка с объектом стены, снаряда и стены или же танка, бонуса и танка.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы был разработан проект с целью создания кооперативной онлайн-игры «Танковая дуэль». В процессе реализации применялись современные технологии разработки, включая язык программирования *C*#, графическую библиотеку *OpenTK* и сетевые технологии для взаимодействия между клиентом и сервером с использованием *winsock* *API*.

Созданный проект демонстрирует успешный пример разработки многопользовательской игры с кооперативным режимом. В ходе работы достигнуты цели, связанные с реализацией функционала для взаимодействия игроков через сеть, обработкой данных от клиентов и обеспечением передачи игровых состояний в реальном времени. Выполнена оптимизация работы с сжатыми данными и их эффективной передачи по сети, что подтвердило значимость данного аспекта для улучшения игрового опыта пользовтаеля.

Разработка многопользовательских игр рассматривается как востребованное направление в сфере программного обеспечения. Представленный проект обладает высокой практической значимостью и может быть использован для создания игр различной сложности и функциональности. Полученные результаты могут применяться как для образовательных целей (обучение разработке игр и сетевых технологий), так и для создания коммерческих продуктов в игровой индустрии.

В перспективе проект может быть расширен за счет добавления новых возможностей, таких как улучшенная физика, поддержка большего количества игроков и дополнительные варианты серверных настроек. Также представляется целесообразным проведение дальнейшей работы над повышением производительности игры для повышения её конкурентоспособности на рынке онлайн-игр.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Frame work 2.0 на языке C#. Мастер класс / Дж. Рихтер. – СПб. Питер , 2008. – 656 с.

2. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов / Вольф Д. – Москва.: ДМК Пресс, 2015. – 368 с.

3. Гинсбург Д., Пурномо Б. OpenGL ES 3.0. Руководство разработчика / Гинсбург Д., Пурномо Б. – Москва. ДМК Пресс, 2015. – 448 с.

4. Любанович Б., Простой Python. Современный стиль программиро-вания. / Любанович, Б. – СПб.: Питер, 2016. – 480 с

5. Берд, Барри. Jаvа для чайников / Берд Барри. – 5-е изд. – 2013. – 368 с.

# **ПРИЛОЖЕНИЕ А**

**Листинг приложения**

(обязательное)

**Файл Programm.cs:**

using Game;

using NetWorkConnector;

using Newtonsoft.Json;

using OutputWindow.Game;

using System.Net;

namespace Server

{

public class Programm

{

public static void Main(string[] args)

{

Console.WriteLine("Server Start");

string pathToConfServer = AppDomain.CurrentDomain.BaseDirectory + "ConfServer.json";

string pathToMap = AppDomain.CurrentDomain.BaseDirectory + "CurentMap.json";

string cfgFileMap = File.ReadAllText(pathToMap);

string cfgFileServer = File.ReadAllText(pathToConfServer);

var dictonaryMap = JsonConvert.DeserializeObject<Dictionary<string, string>>(cfgFileMap);

var dictonaryServer = JsonConvert.DeserializeObject<Dictionary<string, string>>(cfgFileServer);

IPAddress ipAddress = IPAddress.Parse(dictonaryServer["IPADRES"]);

IPEndPoint iPEndPoint = new IPEndPoint(ipAddress, int.Parse(dictonaryServer["PORT"]));

string map = AppDomain.CurrentDomain.BaseDirectory + "MetaData\\Map's\\" + dictonaryMap["Map"];

ConectionManager conectionManager = new ConectionManager();

SingeltonEngine.SetConectionMenager(conectionManager);

conectionManager.SetHost(true, iPEndPoint);

LoaderTexture loaderTexture = new LoaderTexture();

loaderTexture.LoadTextures();

ObjectLoader objectLoader = new ObjectLoader(map, loaderTexture.Textures);

GameEngine gameEngine = new GameEngine(objectLoader);

}

}

}

**Файл PlayerInputController.cs:**

using OpenTK.Input;

namespace Server.GameLogic.InputController

{

public class PlayerInputController

{

public Dictionary<string, bool> KeyValuePairsPush => \_keyValuePairsPush;

private Dictionary<string, bool> \_keyValuePairsPush = new Dictionary<string, bool>();

public Dictionary<string, bool> KeyValuePairsUp => \_keyValuePairsUp;

private Dictionary<string, bool> \_keyValuePairsUp = new Dictionary<string, bool>();

public Dictionary<string, bool> KeyValuePairsDown => \_keyValuePairsDown;

private Dictionary<string, bool> \_keyValuePairsDown = new Dictionary<string, bool>();

private int \_playerToken;

public int PlayerToken => \_playerToken;

public PlayerInputController()

{

foreach (Key key in Enum.GetValues(typeof(Key)))

{

\_keyValuePairsPush[key.ToString()] = false;

\_keyValuePairsUp[key.ToString()] = false;

\_keyValuePairsDown[key.ToString()] = false;

}

}

public PlayerInputController(int token)

{

\_playerToken = token;

foreach (Key key in Enum.GetValues(typeof(Key)))

{

\_keyValuePairsPush[key.ToString()] = false;

\_keyValuePairsUp[key.ToString()] = false;

\_keyValuePairsDown[key.ToString()] = false;

}

}

public bool PushedButton(Key key)

{

var t = key.ToString();

return \_keyValuePairsPush[key.ToString()];

}

public bool UpButton(Key key)

{

var v = \_keyValuePairsUp[key.ToString()];

return \_keyValuePairsUp[key.ToString()];

}

public bool DownButton(Key key)

{

return \_keyValuePairsDown[key.ToString()];

}

public void UpdateInput()

{

// Сбрасываем состояния Up и Down на каждом кадре

foreach (var key in \_keyValuePairsUp.Keys.ToList())

{

\_keyValuePairsUp[key] = false;

}

foreach (var key in \_keyValuePairsDown.Keys.ToList())

{

\_keyValuePairsDown[key] = false;

}

foreach (var key in \_keyValuePairsPush.Keys.ToList())

{

\_keyValuePairsPush[key] = false;

}

}

}

}

**Файл GameEngine.cs:**

using Behaivourus;

using GameObjects;

using NetWorkConnector;

using OutputWindow.Game;

using OutputWindow.Network;

using Server.MonoBehaivourus;

using Server.Network;

namespace Game

{

public class GameEngine

{

List<GameObject> \_gameObjects = new List<GameObject>();

List<GameObject> \_gameObjectsToDelete = new List<GameObject>();

List<GameObject> \_gameObjectsToAdd = new List<GameObject>();

public List<GameObject> GameObjects => \_gameObjects;

EventHandler \_handlerUpdate;

EventHandler \_handlerStart;

public EventHandler DeleteEventOfObject;

int threadCountMaxSize => 24;

int threadCount = 0;

bool \_start = false;

bool \_objectLoaded = false;

bool \_restart = false;

public GameEngine(ObjectLoader objectLoader)

{

TimeDeltaHelper.Start();

SingeltonEngine.SetEngine(this);

SingeltonEngine.SetObjectLoader(objectLoader);

Thread thread = new Thread(WorkCycle);

thread.Start();

}

public GameObject GetGameObjectByID(int ID)

{

foreach (GameObject gameObject in \_gameObjects)

{

if (ID == gameObject.GameObjectID)

{

return gameObject;

}

}

return null;

}

public GameObject GetGameObjectByName(string name)

{

foreach (GameObject gameObject in \_gameObjects)

{

if (name == gameObject.NameObject)

{

return gameObject;

}

}

return null;

}

public void AddObject(GameObject gameObject)

{

lock (\_gameObjects)

{

\_gameObjects.Add(gameObject);

}

foreach (var monoBehaivour in gameObject.MonoBehaivours)

{

\_handlerUpdate += monoBehaivour.Update;

\_handlerStart += monoBehaivour.Start;

}

}

public void RemoveListener(MonoBehaivour monoBehaivour)

{

\_handlerUpdate -= monoBehaivour.Update;

\_handlerStart -= monoBehaivour.Start;

}

public void AddListener(MonoBehaivour monoBehaivour)

{

\_handlerUpdate += monoBehaivour.Update;

\_handlerStart += monoBehaivour.Start;

}

private void AddListeners()

{

foreach (GameObject obj in \_gameObjects)

{

foreach (var monoBehaivour in obj.MonoBehaivours)

{

\_handlerUpdate += monoBehaivour.Update;

\_handlerStart += monoBehaivour.Start;

}

}

}

public void UpdateStatet()

{

var list = SingeltonEngine.ConectionManager.GetDataFromUsers();

foreach (var gameState in list)

{

foreach (var mesage in gameState.GameStates)

{

GameObject gameObject = GetGameObjectByID(mesage.ObjectID);

if (gameObject != null)

{

if (mesage.Type == StateObject.TypeOfMesage.Read)

{

mesage.ReturnObject(gameObject, gameState.Player.InputController);

}

else if (mesage.Type == StateObject.TypeOfMesage.Create)

{

mesage.CreateObject(gameState.Player.InputController);

}

else

{

SingeltonEngine.GameEngine.AddObjectToDelete(gameObject);

}

}

}

}

}

public void GetDataFromUser(object data, EventArgs eventArgs)

{

var gameState = (GameState)data;

foreach (var mesage in gameState.GameStates)

{

GameObject gameObject = GetGameObjectByID(mesage.ObjectID);

if (gameObject != null)

{

if (mesage.Type == StateObject.TypeOfMesage.Read)

{

mesage.ReturnObject(gameObject, gameState.Player.InputController);

}

else if (mesage.Type == StateObject.TypeOfMesage.Create)

{

mesage.CreateObject(gameState.Player.InputController);

}

else

{

SingeltonEngine.GameEngine.AddObjectToDelete(gameObject);

}

}

}

}

public void Restart()

{

foreach (GameObject obj in \_gameObjects)

{

SingeltonEngine.ConectionManager.AddObjectToSend(obj, StateObject.TypeOfMesage.Delete);

AddObjectToDelete(obj);

}

SingeltonEngine.ConectionManager.SendStateFromServer();

ClearObjects();

\_restart = true;

\_objectLoaded = false;

\_start = false;

}

public void Start()

{

if (!\_objectLoaded)

{

var gmOBJ = SingeltonEngine.ObjectLoader.Load();

foreach (var item in gmOBJ)

{

AddObject(item);

}

\_objectLoaded = true;

}

\_start = false;

}

int FPS\_COUNTER = 0;

float \_time = 0;

private object \_monitor = new object();

SemaphoreSlim \_semaphore = new SemaphoreSlim(8);

private const float TARGET\_FPS = 90;

private const float FRAME\_TIME = 1000 / TARGET\_FPS; // миллисекунды на кадр

//-------------------------------------------------

private async void WorkCycle()

{

while (true)

{

var startTime = DateTime.Now;

var newColGameObj = new List<GameObject>(\_gameObjects);

List<Task> tasks = new List<Task>();

int count = 0;

int cicle\_count = 0;

UpdateStatet();

// Обработка коллизий

foreach (var gameObject in newColGameObj)

{

gameObject.Collision.MoveCollision();

if (gameObject.Collision.isTurn && !gameObject.IsDestroyed)

{

var task = ProcessCollisionAsync(gameObject, newColGameObj, cicle\_count + 1);

tasks.Add(task);

count++;

}

cicle\_count++;

}

// Ожидаем завершения всех задач проверки коллизий

if (tasks.Count > 0)

{

await Task.WhenAll(tasks);

}

\_handlerUpdate?.Invoke(this, null);

//Сигнал на отправку пакета

SingeltonEngine.ConectionManager.SendStateFromServer();

SingeltonEngine.ConectionManager.EndOfState();

if (!\_start)

{

Start();

\_handlerStart.Invoke(this, null);

SingeltonEngine.ConectionManager.SetPlaybelObjects();

if (\_restart)

{

SingeltonEngine.ConectionManager.SendDataOnRestart();

\_restart = false;

}

\_start = true;

}

ClearObjects();

if (\_time < 1.0)

{

\_time = \_time + TimeDeltaHelper.DeltaT;

FPS\_COUNTER++;

}

else

{

Console.WriteLine("COUNT FPS " + FPS\_COUNTER);

Console.WriteLine("COUNT " + count + " COUNT OBJ " + newColGameObj.Count + " CYCLE COUNT " + cicle\_count);

FPS\_COUNTER = 0;

\_time = 0;

}

var processingTime = (DateTime.Now - startTime).TotalMilliseconds;

var sleepTime = FRAME\_TIME - processingTime;

if (sleepTime > 0)

{

Thread.Sleep((int)sleepTime);

}

TimeDeltaHelper.Update();

}

}

private async Task ProcessCollisionAsync(GameObject gameObject, List<GameObject> colList, int index\_start)

{

await \_semaphore.WaitAsync();

try

{

await Task.Run(() => CheckCol(gameObject, colList, index\_start));

}

finally

{

\_semaphore.Release();

}

}

private void CheckCol(GameObject gameObject, List<GameObject> gameObjects, int index\_start)

{

foreach (var nextObject in gameObjects)

{

if (nextObject.Collision.isTurn && nextObject.GameObjectID != gameObject.GameObjectID)

{

nextObject.Collision.MoveCollision();

gameObject.Collision.CheckCollision(nextObject);

}

}

threadCount--;

}

private void ClearObjects()

{

if (\_gameObjectsToDelete.Count > 0)

{

foreach (var gameObject in \_gameObjectsToDelete)

{

foreach (var monoBehaivour in gameObject.MonoBehaivours)

{

\_handlerUpdate -= monoBehaivour.Update;

\_handlerStart -= monoBehaivour.Start;

gameObject.Collision.RemoveListener(monoBehaivour);

lock (\_gameObjects)

{

\_gameObjects.Remove(gameObject);

}

DeleteEventOfObject?.Invoke(gameObject, null);

}

}

\_gameObjectsToDelete.Clear();

}

}

public void AddObjectToDelete(GameObject gameObject)

{

lock (\_gameObjectsToDelete)

{

\_gameObjectsToDelete.Add(gameObject);

}

}

}

}

**Файл LoaderTexture.cs:**

using GameObjects;

namespace Game

{

public class LoaderTexture

{

public List<Texture> Textures => \_textures;

private List<Texture> \_textures = new List<Texture>();

public void LoadTextures()

{

string mainDir = AppDomain.CurrentDomain.BaseDirectory + "MetaData\\Texture's\\";

string[] files = Directory.GetFiles(mainDir);

int index = 0;

foreach (string file in files)

{

Texture texture = new Texture(file, index);

\_textures.Add(texture);

index++;

}

}

}

}

**Файл ObjectLoader.cs:**

using GameObjects;

using OpenTK;

using OutputWindow.Game;

using OutputWindow.Network;

using OutputWindow.Scripts.BonusScript;

using OutputWindow.Scripts.TankScripts;

using Server.Script\_s.GameScript;

using System.Drawing;

namespace Game

{

public class ObjectLoader

{

string \_pathToMap;

private List<Texture> \_textures;

private List<string> \_nameOfObjects = new List<string>();

public List<string> NameOfObjects => \_nameOfObjects;

public ObjectLoader(string file, List<Texture> textures)

{

\_pathToMap = file;

\_textures = textures;

foreach (Texture tex in textures)

{

\_nameOfObjects.Add(tex.Name);

}

}

public Texture GetTexturByID(int ID)

{

foreach (var item in \_textures)

{

if (item.Index == ID)

{

return item;

}

}

return null;

}

public List<GameObject> Load()

{

List<GameObject> gameObjects = new List<GameObject>();

using (Bitmap map = new Bitmap(\_pathToMap))

{

float stepX = 2 / (float)map.Width;

float stepY = 2 / (float)map.Width;

Color tank\_1 = Color.FromArgb(0, 128, 0);

Color tank\_2 = Color.FromArgb(0, 0, 128);

Color gas = Color.FromArgb(255, 0, 0);

Color wall = Color.FromArgb(0, 0, 0);

Color bonus = Color.FromArgb(255, 255, 0);

Color armor = Color.FromArgb(192, 192, 192);

Color ammo = Color.FromArgb(255, 0, 255);

Mesh mesh1 = GetMesh(Vector3.Zero);

mesh1.ScaleTo(1f);

Texture texture = null;

foreach (var item in \_textures)

{

if (item.Name.Contains("mapTex"))

{

texture = item;

break;

}

}

GameObject gameObject = new GameObject(mesh1, texture);

gameObject.Collision.isTurn = false;

GameScript gameScript = new GameScript(gameObject);

gameObject.AddBehaivour(gameScript);

gameObjects.Add(gameObject);

for (int i = 0; i < map.Width; i++)

{

for (int j = 0; j < map.Height; j++)

{

Vector3 position = GetCenterOfObject(stepX, stepY, i, j);

if (map.GetPixel(i, j) == tank\_1)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("TankT72\_Green"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "TankT72\_Green";

gameObject.Mesh.ScaleTo(gameObject.Mesh.Scale - gameObject.Mesh.Scale \* 0.15f);

gameObject.MonoBehaivours.Add(new PlayerTankScript(gameObject));

break;

}

}

}

else if (map.GetPixel(i, j) == tank\_2)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("TankT72\_Red"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "TankT72\_Red";

gameObject.Mesh.ScaleTo(gameObject.Mesh.Scale - gameObject.Mesh.Scale \* 0.15f);

gameObject.MonoBehaivours.Add(new PlayerTankScript(gameObject));

break;

}

}

}

else if (map.GetPixel(i, j) == wall)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("wall"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "Wall";

break;

}

}

}

else if (map.GetPixel(i, j) == bonus)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("BonusBox"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "BonusBox";

gameObject.MonoBehaivours.Add(new BonusSpawnerScript(gameObject));

break;

}

}

}

else if (map.GetPixel(i, j) == gas)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("Fuel\_Can"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "Fuel\_Can";

gameObject.MonoBehaivours.Add(new FullGasBehaivour(gameObject));

break;

}

}

}

else if (map.GetPixel(i, j) == ammo)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("AmmoBox"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "AmmoBox";

gameObject.MonoBehaivours.Add(new FullGasBehaivour(gameObject));

break;

}

}

}

else if (map.GetPixel(i, j) == armor)

{

foreach (var listText in \_textures)

{

if (listText.Name.Contains("shield"))

{

Mesh mesh = GetMesh(position);

mesh.ScaleTo(1.0f / map.Width);

gameObject = new GameObject(mesh, listText);

gameObjects.Add(gameObject);

gameObject.NameObject = "shield";

gameObject.MonoBehaivours.Add(new ArmorBehaivour(gameObject));

break;

}

}

}

}

}

}

return gameObjects;

}

public GameObject CreateObject(string nameOfObject)

{

Vector3 pos = Vector3.Zero;

Mesh mesh = GetMesh(pos);

Texture texture = null;

foreach (var listText in \_textures)

{

if (listText.Name == nameOfObject)

{

texture = listText;

break;

}

}

GameObject gameObject = new GameObject(mesh, texture);

SingeltonEngine.GameEngine.AddObject(gameObject);

return gameObject;

}

public GameObject CreateObject(string nameOfObject, StateObject.TypeOfMesage netType)

{

Vector3 pos = Vector3.Zero;

Mesh mesh = GetMesh(pos);

Texture texture = null;

foreach (var listText in \_textures)

{

if (listText.Name == nameOfObject)

{

texture = listText;

break;

}

}

GameObject gameObject = new GameObject(mesh, texture);

if (StateObject.TypeOfMesage.Create == netType)

{

SingeltonEngine.GameEngine.AddObject(gameObject);

}

return gameObject;

}

public GameObject CreateObject(string nameOfObject, List<Type> TypeOfScripts)

{

Vector3 pos = Vector3.Zero;

Mesh mesh = GetMesh(pos);

Texture texture = null;

foreach (var listText in \_textures)

{

if (listText.Name == nameOfObject)

{

texture = listText;

break;

}

}

GameObject gameObject = new GameObject(mesh, texture);

object[] objects = { gameObject };

foreach (var type in TypeOfScripts)

{

gameObject.AddBehaivour((Behaivourus.MonoBehaivour)Activator.CreateInstance(type, gameObject));

}

SingeltonEngine.GameEngine.AddObject(gameObject);

return gameObject;

}

private Vector3 GetCenterOfObject(float stepX, float stepY, float x, float y)

{

Vector3 objectPos = Vector3.Zero;

objectPos.X = -1 + (x + 0.5f) \* stepX;

objectPos.Y = -1 + (y + 0.5f) \* stepY;

return objectPos;

}

private Mesh GetMesh(Vector3 pos)

{

Vector3[] vercites = new Vector3[4];

//First Face

vercites[0] = new Vector3(1f, 1f, 0f);

vercites[1] = new Vector3(1f, -1f, 0f);

vercites[2] = new Vector3(-1f, -1f, 0f);

vercites[3] = new Vector3(-1f, 1f, 0f);

return new Mesh(pos, vercites);

}

// метод для вычисления координат для объекта

private Vector2[] GetPosition(Vector2 firstPoint, float coefX, float coefY, float x, float y)

{

firstPoint = new Vector2(x, y);

Vector2[] objectVercites = new Vector2[]

{

new Vector2(firstPoint.X - coefX, firstPoint.Y),

new Vector2(firstPoint.X,firstPoint.Y),

new Vector2(firstPoint.X - coefY,firstPoint.Y + coefY),

new Vector2(firstPoint.X ,firstPoint.Y + coefY)

};

return objectVercites;

}

}

}

**Файл SingeltonEngine.cs:**

using Game;

using NetWorkConnector;

namespace OutputWindow.Game

{

public static class SingeltonEngine

{

public static GameEngine GameEngine => \_gameEngine;

static GameEngine \_gameEngine;

public static ObjectLoader ObjectLoader => \_objectLoader;

private static ObjectLoader \_objectLoader;

private static ConectionManager \_conectionManager;

public static ConectionManager ConectionManager => \_conectionManager;

static bool \_isSetEngine = false;

static bool \_isSetLoader = false;

public static void SetEngine(GameEngine gameEngine)

{

if (!\_isSetEngine)

{

\_isSetEngine = true;

\_gameEngine = gameEngine;

}

}

public static void SetObjectLoader(ObjectLoader loader)

{

if (!\_isSetLoader)

{

\_isSetLoader = true;

\_objectLoader = loader;

}

}

public static void SetConectionMenager(ConectionManager conectionManager)

{

\_conectionManager = conectionManager;

}

}

}

**Файл TimeDeltaHelper.cs:**

using System.Diagnostics;

namespace OutputWindow.Game

{

public static class TimeDeltaHelper

{

private static bool \_isStart = false;

public static bool IsStart => \_isStart;

private static Stopwatch \_stopwatch = new Stopwatch();

private static int \_framesCounter = 0;

private static int \_fps = 0;

public static int FPS { get => \_fps; }

private static long \_previousFPSMeasurementTime;

private static long \_previousTicks;

public static float TimeSpeed = 1f;

private static float \_time;

public static float Time { get => \_time; }

private static float \_deltaT;

public static float DeltaT { get => \_deltaT; }

public static void Start()

{

\_stopwatch.Reset();

\_framesCounter = 0;

\_fps = 0;

\_stopwatch.Start();

\_previousFPSMeasurementTime = \_stopwatch.ElapsedMilliseconds;

\_previousTicks = \_stopwatch.Elapsed.Ticks;

\_isStart = true;

}

public static void Update()

{

long ticks = \_stopwatch.Elapsed.Ticks;

\_time = (float)ticks / TimeSpan.TicksPerSecond;

\_deltaT = (float)(ticks - \_previousTicks) / TimeSpan.TicksPerSecond;

\_deltaT = \_deltaT \* TimeSpeed;

\_previousTicks = ticks;

\_framesCounter++;

if (\_stopwatch.ElapsedMilliseconds - \_previousFPSMeasurementTime >= 1000)

{

\_fps = \_framesCounter;

\_framesCounter = 0;

\_previousFPSMeasurementTime = \_stopwatch.ElapsedMilliseconds;

}

}

}

}

**Файл Collision.cs:**

using Behaivourus;

using OpenTK;

namespace GameObjects

{

public class Collision

{

public bool Detected = false;

private GameObject \_gameObject;

public GameObject gameObject => \_gameObject;

private Vector3 \_min;

private Vector3 \_max;

public Vector3 Min => \_min;

public Vector3 Max => \_max;

private EventHandler \_colider;

public EventHandler Colider => \_colider;

private EventHandler \_triger;

public EventHandler Trigger => \_triger;

public bool isTurn = true;

public bool isTriger = false;

public Collision(GameObject gameObject)

{

\_gameObject = gameObject;

FoundMaxPos();

FoundMinPos();

AddListener();

}

private void FoundMinPos()

{

List<Vector3> ver = new List<Vector3>();

foreach (var item in gameObject.Mesh.Vercites)

{

ver.Add(Vector3.TransformPosition(item, gameObject.Matrix));

}

var firstPos = ver[0];

Vector3 minPos = new Vector3(firstPos.X, firstPos.Y, firstPos.Z);

foreach (var pos in ver)

{

if (minPos.X > pos.X)

{

minPos.X = pos.X;

}

else if (minPos.Y > pos.Y)

{

minPos.Y = pos.Y;

}

else if (minPos.Z > pos.Z)

{

minPos.Z = pos.Z;

}

}

\_min = minPos;

}

private void FoundMaxPos()

{

List<Vector3> ver = new List<Vector3>();

foreach (var item in gameObject.Mesh.Vercites)

{

ver.Add(Vector3.TransformPosition(item, gameObject.Matrix));

}

var firstPos = ver[0];

Vector3 maxPos = new Vector3(firstPos.X, firstPos.Y, firstPos.Z);

foreach (var pos in ver)

{

if (maxPos.X < pos.X)

{

maxPos.X = pos.X;

}

else if (maxPos.Y < pos.Y)

{

maxPos.Y = pos.Y;

}

else if (maxPos.Z < pos.Z)

{

maxPos.Z = pos.Z;

}

}

\_max = maxPos;

}

public void MoveCollision()

{

FoundMaxPos();

FoundMinPos();

}

public void AddListener(MonoBehaivour monoBehaivour)

{

\_colider += monoBehaivour.Colision;

\_triger += monoBehaivour.Colision;

}

private void AddListener()

{

foreach (var behaivour in this.gameObject.MonoBehaivours)

{

\_colider += behaivour.Colision;

\_triger += behaivour.Trigger;

}

}

public void RemoveListener(MonoBehaivour monoBehaivour)

{

\_colider -= monoBehaivour.Colision;

\_triger -= monoBehaivour.Colision;

}

public void RemoveListener()

{

foreach (var monoBehaivour in \_gameObject.MonoBehaivours)

{

\_colider -= monoBehaivour.Colision;

\_triger -= monoBehaivour.Colision;

}

}

public void CheckCollision(GameObject gameObject)

{

Vector3 selfMin = \_min;

Vector3 selMax = \_max;

Vector3 min = gameObject.Collision.Min;

Vector3 max = gameObject.Collision.Max;

bool isIntersect = Intersect(min, max, selfMin, selMax);

if (isIntersect)

{

if (\_gameObject.SimpleCollision.IsTurn && !\_gameObject.Collision.isTriger && !gameObject.Collision.isTriger)

{

\_gameObject.SimpleCollision.CorrectPosition(selfMin, selMax, min, max);

}

this.Invoke(gameObject);

Detected = true;

}

}

private bool Intersect(Vector3 min, Vector3 max, Vector3 selfMin, Vector3 selfMax)

{

return selfMin.X < max.X && selfMax.X > min.X &&

selfMin.Y < max.Y && selfMax.Y > min.Y;

}

private void Invoke(GameObject gameObject)

{

if (!isTriger)

{

this.\_colider?.Invoke(gameObject.Collision, null);

}

else

{

this.\_triger?.Invoke(gameObject.Collision, null);

}

if (!gameObject.Collision.isTriger)

{

gameObject.Collision.Colider?.Invoke(gameObject.Collision, null);

}

else

{

gameObject.Collision.Trigger?.Invoke(gameObject.Collision, null);

}

}

}

}

**Файл GameObject.cs:**

using Behaivourus;

using Newtonsoft.Json;

using OpenTK;

using OutputWindow.Game;

using OutputWindow.GameObjects;

namespace GameObjects

{

public class GameObject

{

[JsonIgnore] private Texture \_texture;

public Texture Texture => \_texture;

[JsonProperty] private Mesh \_mesh;

public Mesh Mesh => \_mesh;

[JsonProperty] private List<MonoBehaivour> \_monoBehaivours = new List<MonoBehaivour>();

public List<MonoBehaivour> MonoBehaivours => \_monoBehaivours;

[JsonIgnore] private Collision \_collision;

public Collision Collision => \_collision;

[JsonIgnore] private SimpleCollision \_simpleCollision;

public SimpleCollision SimpleCollision => \_simpleCollision;

public Vector3 Position => \_mesh.Position;

public Matrix4 Matrix => \_mesh.GetWorldMatrix();

[JsonProperty] public string NameObject = "Object";

[JsonProperty] public int GameObjectID;

public bool IsDestroyed => \_isDestroyed;

private bool \_isDestroyed = false;

public bool IsVisible = true;

public GameObject(Mesh mesh, Texture texture)

{

\_texture = texture;

\_mesh = mesh;

\_collision = new Collision(this);

\_monoBehaivours.Add(new MonoBehaivour(this));

GameObjectID = this.GetHashCode();

\_simpleCollision = new SimpleCollision(this);

\_simpleCollision.TurnOfOn();

}

public GameObject(Mesh mesh, Texture texture, MonoBehaivour monoBehaivour)

{

\_texture = texture;

\_mesh = mesh;

\_collision = new Collision(this);

\_monoBehaivours.Add(monoBehaivour);

\_simpleCollision = new SimpleCollision(this);

\_simpleCollision.TurnOfOn();

}

public void SetTexture(Texture texture)

{

\_texture = texture;

}

public void AddBehaivour(MonoBehaivour monoBehaivour)

{

\_monoBehaivours.Add(monoBehaivour);

\_collision.AddListener(monoBehaivour);

SingeltonEngine.GameEngine.AddListener(monoBehaivour);

}

public void DestroyObject()

{

\_isDestroyed = true;

}

}

}

**Файл Mesh.cs:**

using Newtonsoft.Json;

using OpenTK;

namespace GameObjects

{

public class Mesh

{

[JsonProperty] private float \_scale = 1f;

public float Scale { get => \_scale; set => \_scale = value; }

private float \_scaleX = 1f;

private float \_scaleY = 1f;

private float \_scaleZ = 1f;

[JsonProperty] private Vector3 \_position;

public Vector3 Position { get => \_position; set => \_position = value; }

[JsonProperty] private float \_yaw;

public float Yaw { get => \_yaw; set => \_yaw = value; }

[JsonProperty] private float \_pitch;

public float Pitch { get => \_pitch; set => \_pitch = value; }

[JsonProperty] private float \_roll;

public float Roll { get => \_roll; set => \_roll = value; }

[JsonProperty] private Vector3[] \_vercites;

public Vector3[] Vercites => \_vercites;

public Mesh(Vector3 position, Vector3[] vercites, float yaw = 0.0f, float pitch = 0.0f, float roll = 0.0f)

{

\_position = position;

\_vercites = vercites;

\_yaw = yaw;

\_pitch = pitch;

\_roll = roll;

}

public void SetPosition(Vector3 pos)

{

\_position = pos;

}

private void LimitAngleByPlusMinusPi(ref float angle)

{

if (angle > MathHelper.Pi) angle -= MathHelper.TwoPi;

else if (angle < -MathHelper.Pi) angle += MathHelper.TwoPi;

}

public virtual void YawBy(float deltaYaw)

{

\_yaw += deltaYaw;

LimitAngleByPlusMinusPi(ref \_yaw);

}

public virtual void PitchBy(float deltaPitch)

{

\_pitch += deltaPitch;

LimitAngleByPlusMinusPi(ref \_pitch);

}

public virtual void RollBy(float deltaRoll)

{

\_roll += deltaRoll;

LimitAngleByPlusMinusPi(ref \_roll);

}

public virtual void MoveBy(float deltaX, float deltaY, float deltaZ)

{

\_position.X += deltaX;

\_position.Y += deltaY;

\_position.Z += deltaZ;

}

public virtual void MoveBy(Vector4 direction)

{

\_position.X += direction.X;

\_position.Y += direction.Y;

\_position.Z += direction.Z;

}

public virtual void MoveTo(float x, float y, float z)

{

\_position.X = x;

\_position.Y = y;

\_position.Z = z;

}

public virtual void ScaleTo(float scale)

{

\_scale = scale;

\_scaleX = scale;

\_scaleY = scale;

\_scaleZ = scale;

}

public void ScaleOnX\_Axis(float scale)

{

\_scale = scale;

\_scaleX = scale;

\_scaleY = scale;

\_scaleZ = scale;

}

public void ScaleOnY\_Axis(float scale)

{

\_scaleY = scale;

}

public void ScaleOnZ\_Axis(float scale)

{

\_scaleZ = scale;

}

public void SetSclae(Vector3 vector)

{

\_scaleX = vector.X;

\_scaleY = vector.Y;

\_scaleZ = vector.Z;

}

public Matrix4 GetWorldMatrix()

{

return Matrix4.Identity \*

Matrix4.CreateScale(new Vector3(\_scaleX, \_scaleY, \_scaleZ)) \*

Matrix4.CreateRotationY(\_yaw) \*

Matrix4.CreateRotationX(\_pitch) \*

Matrix4.CreateRotationZ(\_roll) \*

Matrix4.CreateTranslation(\_position);

}

}

}

**Файл SimpleCollision.cs:**

using GameObjects;

using OpenTK;

using OutputWindow.Game;

namespace OutputWindow.GameObjects

{

public class SimpleCollision

{

public bool IsTurn => \_isTurn;

private bool \_isTurn = true;

Vector3 \_direction = Vector3.Zero;

GameObject gameObjects;

public void TurnOfOn()

{

\_isTurn = !\_isTurn;

}

public SimpleCollision(GameObject gameObjects)

{

this.gameObjects = gameObjects;

}

public void AddImpact(float veilocity, Vector3 direction)

{

Vector3 move = direction \* veilocity \* TimeDeltaHelper.DeltaT;

\_direction = direction;

gameObjects.Mesh.MoveBy(move.X, move.Y, move.Z);

}

public void CorrectPosition(Vector3 selfMin, Vector3 selfMax, Vector3 wallMin, Vector3 wallMax)

{

if (\_isTurn)

{

Vector3 correctedPosition = gameObjects.Position;

// Проверка пересечения по X

if (selfMax.X > wallMin.X && selfMin.X < wallMin.X && \_direction.X > 0)

{

// Двигаемся влево, отодвигаем вправо

correctedPosition.X = wallMin.X - (selfMax.X - selfMin.X) / 2;

}

else if (selfMin.X < wallMax.X && selfMax.X > wallMax.X && \_direction.X < 0)

{

// Двигаемся вправо, отодвигаем влево

correctedPosition.X = wallMax.X + (selfMax.X - selfMin.X) / 2;

}

// Проверка пересечения по Y

if (selfMax.Y > wallMin.Y && selfMin.Y < wallMin.Y && \_direction.Y > 0)

{

// Двигаемся вверх, отодвигаем вниз

correctedPosition.Y = wallMin.Y - (selfMax.Y - selfMin.Y) / 2;

}

else if (selfMin.Y < wallMax.Y && selfMax.Y > wallMax.Y && \_direction.Y < 0)

{

// Двигаемся вниз, отодвигаем вверх

correctedPosition.Y = wallMax.Y + (selfMax.Y - selfMin.Y) / 2;

}

// Проверка пересечения по Z

if (selfMax.Z > wallMin.Z && selfMin.Z < wallMin.Z && \_direction.Z > 0)

{

// Двигаемся вперёд, отодвигаем назад

correctedPosition.Z = wallMin.Z - (selfMax.Z - selfMin.Z) / 2;

}

else if (selfMin.Z < wallMax.Z && selfMax.Z > wallMax.Z && \_direction.Z < 0)

{

// Двигаемся назад, отодвигаем вперёд

correctedPosition.Z = wallMax.Z + (selfMax.Z - selfMin.Z) / 2;

}

gameObjects.Mesh.SetPosition(correctedPosition);

}

}

}

}

**Файл Texture.cs:**

using OpenTK;

using System.Drawing;

using System.Drawing.Imaging;

using System.Runtime.InteropServices;

namespace GameObjects

{

public class Texture

{

public int Handle;

public string Name => name;

private Vector2[] \_textureCords;

public Vector2[] TextureCords => \_textureCords;

public int Index;

byte[] data;

int height;

int width;

string name;

// чтение растрого изображения

public Texture(string path, int index)

{

Index = index;

string[] folders = path.Split("\\");

using (Bitmap bitmap = new Bitmap(path))

{

name = folders[folders.Length - 1].Split(".")[0];

height = bitmap.Height;

width = bitmap.Width;

BitmapData wallData = bitmap.LockBits(new Rectangle(0, 0, width, height), ImageLockMode.ReadOnly

, System.Drawing.Imaging.PixelFormat.Format32bppArgb);

IntPtr intPtr = wallData.Scan0;

int bytes = wallData.Stride \* wallData.Height;

data = new byte[bytes];

Marshal.Copy(intPtr, data, 0, bytes);

bitmap.UnlockBits(wallData);

bitmap.Dispose();

\_textureCords = new Vector2[4];

\_textureCords[0] = new Vector2(1f, 1f);

\_textureCords[1] = new Vector2(1f, 0f);

\_textureCords[2] = new Vector2(0f, 0f);

\_textureCords[3] = new Vector2(0f, 1f);

}

}

// получения массива байт с текстурой

public byte[] GetData() { return data; }

// получение высоты

public int GetHeight() { return height; }

// получение длинны

public int GetWidth() { return width; }

}

}

**Файл MonoBehaivour.cs:**

using GameObjects;

using Newtonsoft.Json;

using OpenTK;

using OutputWindow.Game;

using OutputWindow.Network;

using Server.GameLogic.InputController;

using System.Reflection;

namespace Behaivourus

{

[AttributeUsage(AttributeTargets.Field | AttributeTargets.Property)]

public class IgnorePropertyAttribute : Attribute { }

public class MonoBehaivour

{

[JsonIgnore]

[IgnoreProperty]

public GameObject GameObject

{

get { return gameObject; }

set { gameObject = value; }

}

[IgnoreProperty]

protected GameObject gameObject;

[IgnoreProperty]

protected Vector3 possion => gameObject.Position;

[IgnoreProperty]

protected List<object> \_attributes = new List<object>();

protected PlayerInputController PlayerInputController;

protected bool \_copyScript = true;

public MonoBehaivour(GameObject mesh)

{

gameObject = mesh;

gameObject.Collision.AddListener(this);

PlayerInputController = new PlayerInputController();

}

public virtual void Start(object sender, EventArgs e)

{

}

public virtual void Update(object sender, EventArgs e)

{

}

public virtual void Colision(object sender, EventArgs e)

{

}

public virtual void Trigger(object sender, EventArgs e)

{

}

public bool IsCopingScript()

{

return \_copyScript;

}

public void SateNewInputState(PlayerInputController playerInputController)

{

PlayerInputController = playerInputController;

}

protected void SereilizeScript(Type type)

{

\_attributes.Clear();

foreach (FieldInfo field in type.GetFields(BindingFlags.Public | BindingFlags.Instance))

{

if (!(field.GetValue(this) is MonoBehaivour))

{

var fieldValue = field.GetValue(this); // Получаем значение поля

\_attributes.Add(fieldValue); // Добавляем в коллекцию

}

}

}

public List<object> ReturnSatate(Type type)

{

SereilizeScript(type);

return \_attributes;

}

public void SateState(List<object> list, PlayerInputController playerInputController)

{

Type type = this.GetType();

int index = 0;

PlayerInputController = playerInputController;

foreach (FieldInfo field in type.GetFields(BindingFlags.Public | BindingFlags.Instance))

{

var fieldType = field.FieldType;

var value = list[index];

if (value is Vector3)

{

field.SetValue(this, list[index]); // Получаем значение поля

index++;

}

else

{

var valueConv = Convert.ChangeType(list[index], fieldType);

field.SetValue(this, valueConv); // Получаем значение поля

index++;

}

}

}

protected void RemoveScript()

{

GameObject.MonoBehaivours.Remove(this);

SingeltonEngine.GameEngine.RemoveListener(this);

}

protected void Delete()

{

SingeltonEngine.GameEngine.AddObjectToDelete(GameObject);

}

public virtual MonoBehaivour Clone()

{

return new MonoBehaivour(GameObject);

}

}

}

**Файл OnlinePlaybleBehaivour.cs:**

namespace Server.MonoBehaivourus

{

public interface OnlinePlaybleBehaivour

{

public void SetToken(int token);

public int GetToken();

}

}

**Файл ConectionManager.cs:**

using GameObjects;

using OutputWindow.Game;

using OutputWindow.Network;

using OutputWindow.Scripts.TankScripts;

using Server.Network;

using System.Net;

using System.Net.Sockets;

namespace NetWorkConnector

{

public class ConectionManager

{

public EventHandler \_eventSend;

public EventHandler \_eventReceive;

private List<Player> \_players = new List<Player>();

private List<StateObject> stateObjects = new List<StateObject>();

private List<PlayerTankScript> \_playbelObjects = new List<PlayerTankScript>();

Socket \_socketHost;

bool \_isHost = false;

public ConectionManager()

{

\_socketHost = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

\_socketHost.NoDelay = true;

\_socketHost.SendBufferSize = 24 \* 1024;

\_socketHost.ReceiveBufferSize = 24 \* 1024;

}

public void SetHost(bool IHost, IPEndPoint endPoint)

{

\_isHost = IHost;

\_socketHost.Bind(endPoint);

Console.WriteLine("Server start on IP/PORT: " + \_socketHost.LocalEndPoint);

\_socketHost.Listen(10);

Thread thread = new Thread(Host);

thread.Start();

}

public void SetPlaybelObjects()

{

foreach (GameObject gameObject in SingeltonEngine.GameEngine.GameObjects)

{

foreach (var item in gameObject.MonoBehaivours)

{

if (item is PlayerTankScript playerTank)

{

\_playbelObjects.Add(playerTank);

break;

}

}

}

}

public void Host()

{

while (true)

{

Socket playerSoc = \_socketHost.Accept();

var player = new Player(playerSoc);

\_players.Add(player);

\_playbelObjects[0].SetToken(player.PlayerToken);

\_playbelObjects.RemoveAt(0);

SendDataOnStart(player);

if (\_players.Count > 2)

{

break;

}

}

}

public void SendDataOnRestart()

{

\_playbelObjects.Clear();

foreach (var player in \_players)

{

List<StateObject> stateObjects = new List<StateObject>();

var gmOBJ = SingeltonEngine.GameEngine.GameObjects;

foreach (var obj in gmOBJ)

{

AddObjectToSend(obj, StateObject.TypeOfMesage.Create);

}

if (\_playbelObjects.Count == 0)

{

SetPlaybelObjects();

}

foreach (var obj in \_playbelObjects)

{

obj.SetToken(player.PlayerToken);

break;

}

if (\_playbelObjects.Count > 0)

{

\_playbelObjects.Remove(\_playbelObjects[0]);

}

}

\_playbelObjects.Clear();

foreach (var player in \_players)

{

player.SendData(stateObjects);

player.SendToken();

}

}

//ОТправка состояния на старте

private void SendDataOnStart(Player player)

{

List<StateObject> stateObjects = new List<StateObject>();

var gmOBJ = SingeltonEngine.GameEngine.GameObjects;

foreach (var obj in gmOBJ)

{

stateObjects.Add(new StateObject(obj, StateObject.TypeOfMesage.Create));

}

player.SendData(stateObjects);

player.SendToken();

}

public List<GameState> GetDataFromUsers()

{

List<GameState> gameStates = new List<GameState>();

foreach (var player in \_players)

{

var statet = player?.GetGameStatet();

if (statet != null)

{

gameStates.Add(statet);

}

}

return gameStates;

}

//Отправка состояния

public void SendStateFromServer()

{

if (\_players.Count > 0 && stateObjects.Count > 0)

{

Console.WriteLine(Thread.CurrentThread.Name);

foreach (var player in \_players)

{

player.SendData(stateObjects);

player.SendToken();

}

lock (stateObjects)

{

stateObjects.Clear();

}

}

}

public void AddObjectToSend(GameObject gameObject, StateObject.TypeOfMesage typeOfMesage)

{

bool canAdd = true;

foreach (var stateObject in stateObjects)

{

if (stateObject.ObjectID == gameObject.GameObjectID)

{

canAdd = false;

}

}

if (canAdd)

{

stateObjects.Add(new StateObject(gameObject, typeOfMesage));

}

}

public void EndOfState()

{

foreach (var player in \_players)

{

player.EndUpdate();

}

}

}

}

**Файл GameState.cs:**

using OutputWindow.Network;

namespace Server.Network

{

public class GameState

{

public Dictionary<string, bool> PushKeys = new Dictionary<string, bool>();

public Dictionary<string, bool> DownKeys = new Dictionary<string, bool>();

public Dictionary<string, bool> UpKeys = new Dictionary<string, bool>();

public List<StateObject> GameStates;

private Player \_player;

public Player Player => \_player;

public int PlayerToken;

public GameState() { }

public void SatePlayer(Player player)

{

\_player = player;

}

}

}

**Файл Player.cs:**

using Newtonsoft.Json;

using OutputWindow.Game;

using OutputWindow.Network;

using Server.GameLogic.InputController;

using System.IO.Compression;

using System.Net.Sockets;

using System.Text;

namespace Server.Network

{

public class Player

{

public PlayerInputController InputController;

private Queue<GameState> \_gameStateQueue = new Queue<GameState>();

Socket \_playerSocket;

EventHandler \_reciveHandler;

int playerToken;

public int PlayerToken => playerToken;

public Player(Socket socket)

{

\_playerSocket = socket;

//\_reciveHandler += SingeltonEngine.GameEngine.GetDataFromUser;

playerToken = GetHashCode();

InputController = new PlayerInputController(playerToken);

Thread thread = new Thread(ReciveData);

thread.Start();

}

public void EndUpdate()

{

InputController.UpdateInput();

}

public void SendToken()

{

byte[] data = BitConverter.GetBytes(playerToken);

\_playerSocket.Send(data);

}

public void SendData(List<StateObject> mesages)

{

string mesage = JsonConvert.SerializeObject(mesages);

byte[] dataSend = Encoding.UTF8.GetBytes(mesage);

using (var outputStream = new MemoryStream())

{

using (var gzipStream = new GZipStream(outputStream, CompressionMode.Compress))

{

gzipStream.Write(dataSend, 0, dataSend.Length);

}

dataSend = outputStream.ToArray();

}

// Сначала отправляем размер данных

byte[] sizeBytes = BitConverter.GetBytes(dataSend.Length);

\_playerSocket.Send(sizeBytes);

// Затем отправляем сами данные

int totalSent = 0;

while (totalSent < dataSend.Length)

{

int sent = \_playerSocket.Send(dataSend, totalSent,

dataSend.Length - totalSent, SocketFlags.None);

totalSent += sent;

}

Console.WriteLine("Sended data bytes " + dataSend.Length + "\n Sended count object's " + mesages.Count + "\n");

}

//Прием данных от пользователя

private void ReciveData()

{

while (true)

{

if (\_playerSocket.Available > 0)

{

// Получаем размер данных

byte[] sizeBuffer = new byte[4];

int totalRead = 0;

while (totalRead < 4)

{

int read = \_playerSocket.Receive(sizeBuffer, totalRead, 4 - totalRead, SocketFlags.None);

if (read == 0) break;

totalRead += read;

}

int expectedSize = BitConverter.ToInt32(sizeBuffer, 0);

// Получаем сами данные

using (MemoryStream memoryStream = new MemoryStream())

{

byte[] buffer = new byte[expectedSize]; // больший размер буфера для эффективности

totalRead = 0;

while (totalRead < expectedSize)

{

int read = \_playerSocket.Receive(buffer, 0,

Math.Min(buffer.Length, expectedSize - totalRead),

SocketFlags.None);

if (read == 0) break; // соединение закрыто

memoryStream.Write(buffer, 0, read);

totalRead += read;

}

GameState gameStatet;

using (var inputStream = new MemoryStream(memoryStream.ToArray()))

using (var gzipStream = new GZipStream(inputStream, CompressionMode.Decompress))

using (var outputStream = new MemoryStream())

{

gzipStream.CopyTo(outputStream);

byte[] decompressedData = outputStream.ToArray();

// Преобразуем обратно в строку JSON

string json = Encoding.UTF8.GetString(decompressedData);

// Десериализуем обратно в объект

gameStatet = JsonConvert.DeserializeObject<GameState>(json);

}

foreach (var item in gameStatet.PushKeys)

{

InputController.KeyValuePairsPush[item.Key] = item.Value;

}

foreach (var item in gameStatet.DownKeys)

{

InputController.KeyValuePairsDown[item.Key] = item.Value;

}

foreach (var item in gameStatet.UpKeys)

{

InputController.KeyValuePairsUp[item.Key] = item.Value;

}

gameStatet.SatePlayer(this);

lock (\_gameStateQueue)

{

if (\_gameStateQueue.Count < \_count)

{

if (\_timerToRecive > \_timeToWait)

{

\_gameStateQueue.Enqueue(gameStatet);

\_timerToRecive = 0;

}

else

{

\_timerToRecive = \_timerToRecive + TimeDeltaHelper.DeltaT;

}

}

}

Console.WriteLine("Recive data bytes " + expectedSize + "\n Recive count object's " + gameStatet.GameStates.Count + "\n");

}

}

}

}

private int \_count = 5;

private float \_timerToRecive = 0f;

private int \_timeToWait => \_count / 1000;

public GameState GetGameStatet()

{

GameState gameState = null;

if (\_gameStateQueue.Count > 0)

{

gameState = \_gameStateQueue.Dequeue();

Console.WriteLine("--------------------------");

Console.WriteLine("Player token " + playerToken);

Console.WriteLine(\_gameStateQueue.Count);

Console.WriteLine("--------------------------");

}

return gameState;

}

}

}

**Файл StateObject.cs:**

using GameObjects;

using OutputWindow.Game;

using Server.GameLogic.InputController;

using Server.MonoBehaivourus;

namespace OutputWindow.Network

{

public class StateObject

{

public enum TypeOfMesage

{ Create, Read, Delete }

public TypeOfMesage Type;

public string TypeOfObject;

public string NameOfGameObject;

public string NameOfTexture;

public bool isAlive = true;

public int ObjectID;

public float Roll;

public float Yaw;

public float Pitch;

public float[] Position;

public float Scale;

public int TextureID;

public bool Visible;

public List<StateScript> Behaivours = new List<StateScript>();

public StateObject()

{

}

public StateObject(GameObject gameObject, TypeOfMesage type)

{

this.Type = type;

TypeOfObject = gameObject.Texture.Name.Split(".")[0];

NameOfGameObject = gameObject.NameObject;

NameOfTexture = gameObject.Texture.Name;

TextureID = gameObject.Texture.Index;

ObjectID = gameObject.GameObjectID;

Roll = gameObject.Mesh.Roll;

Yaw = gameObject.Mesh.Yaw;

Pitch = gameObject.Mesh.Pitch;

Position = new float[3];

Position[0] = gameObject.Position.X;

Position[1] = gameObject.Position.Y;

Position[2] = gameObject.Position.Z;

Visible = gameObject.IsVisible;

Scale = gameObject.Mesh.Scale;

foreach (var item in gameObject.MonoBehaivours)

{

if (item.IsCopingScript())

{

Behaivours.Add(new StateScript(item));

}

}

}

//Для обновления объекта

public void ReturnObject(GameObject gameObject, PlayerInputController playerInputController)

{

if (TypeOfMesage.Create == Type)

gameObject.SetTexture(SingeltonEngine.ObjectLoader.GetTexturByID(TextureID));

int index = 0;

foreach (var item in gameObject.MonoBehaivours)

{

if (item.IsCopingScript())

{

if (item is OnlinePlaybleBehaivour online)

{

if (online.GetToken() == playerInputController.PlayerToken)

{

item.SateState(Behaivours[index].StatementOfScript, playerInputController);

}

}

else

{

item.SateState(Behaivours[index].StatementOfScript, playerInputController);

}

}

index++;

}

}

public void CreateObject(PlayerInputController playerInputController)

{

GameObject gameObject = SingeltonEngine.ObjectLoader.CreateObject(NameOfGameObject);

gameObject.NameObject = NameOfGameObject;

gameObject.GameObjectID = ObjectID;

gameObject.Mesh.SetPosition(new OpenTK.Vector3(Position[0], Position[1], Position[2]));

gameObject.Mesh.Roll = Roll;

gameObject.Mesh.Yaw = Yaw;

gameObject.Mesh.Pitch = Pitch;

gameObject.Mesh.Scale = Scale;

gameObject.IsVisible = Visible;

if (TypeOfMesage.Create == Type)

gameObject.SetTexture(SingeltonEngine.ObjectLoader.GetTexturByID(TextureID));

int index = 0;

foreach (var item in gameObject.MonoBehaivours)

{

item.SateState(Behaivours[index].StatementOfScript, playerInputController);

index++;

}

SingeltonEngine.GameEngine.AddObject(gameObject);

}

}

}

**Файл StateScript.cs:**

using Behaivourus;

namespace OutputWindow.Network

{

public class StateScript

{

public int ObjectID;

public List<object> StatementOfScript;

public string TypeOfScript;

public StateScript()

{

}

public StateScript(MonoBehaivour monoBehaivour)

{

ObjectID = monoBehaivour.GameObject.GameObjectID;

StatementOfScript = monoBehaivour.ReturnSatate(monoBehaivour.GetType());

TypeOfScript = monoBehaivour.GetType().ToString();

}

}

}

**Файл ArmorBehaivour.cs:**

using GameObjects;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class ArmorBehaivour : BonusBehaivour

{

public ArmorBehaivour(GameObject mesh) : base(mesh)

{

}

public override void SetBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetDeltaArmmor(10f);

}

}

public override void RemoveBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetDeltaArmmor(10f);

}

}

}

}

**Файл BonusBehaivour.cs:**

using Behaivourus;

using GameObjects;

using OutputWindow.Game;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class BonusBehaivour : MonoBehaivour

{

protected MonoBehaivour \_target;

protected float \_currentTime = 0;

protected float \_maxTime = 10;

protected bool \_active = false;

protected bool \_hasTimer = true;

public BonusBehaivour(GameObject mesh) : base(mesh)

{

\_copyScript = false;

}

public void SetTarget(PlayerTankScript playerTank)

{

\_target = playerTank;

\_active = true;

SetBonus();

}

public virtual void SetBonus()

{

}

public virtual void RemoveBonus()

{

}

public override void Update(object sender, EventArgs e)

{

if (\_active)

{

if (\_currentTime < \_maxTime && \_hasTimer)

{

\_currentTime += TimeDeltaHelper.DeltaT;

}

else

{

RemoveBonus();

RemoveScript();

}

}

}

}

}

**Файл BonusSpawnerScript.cs:**

using GameObjects;

using OutputWindow.Game;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

internal class BonusSpawnerScript : BonusBehaivour

{

private float \_maxTime = 10;

private float \_curentTime = 0;

private bool \_active = false;

public BonusSpawnerScript(GameObject mesh) : base(mesh)

{

GameObject.Collision.isTriger = true;

\_copyScript = false;

}

public override void Colision(object sender, EventArgs e)

{

if (sender is Collision collision)

{

if (collision.gameObject.NameObject.Contains("Tank"))

{

if (!\_active)

{

Timer();

foreach (var script in collision.gameObject.MonoBehaivours)

{

if (script is PlayerTankScript player)

{

AbsBonus(player);

GameObject.IsVisible = false;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

SingeltonEngine.ConectionManager.AddObjectToSend(player.GameObject, Network.StateObject.TypeOfMesage.Read);

break;

}

}

}

}

}

}

public override void Update(object sender, EventArgs e)

{

if (\_active)

{

if (\_curentTime > \_maxTime)

{

\_active = false;

GameObject.IsVisible = true;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

}

\_curentTime += TimeDeltaHelper.DeltaT;

}

}

private void Timer()

{

\_curentTime = 0;

\_active = true;

}

private BonusBehaivour AbsBonus(PlayerTankScript gameObject)

{

BonusBehaivour bonus = null;

Random random = new Random();

int index = random.Next(0, 6);

switch (index)

{

case 0:

bonus = new ArmorBehaivour(gameObject.GameObject);

break;

case 1:

bonus = new DownDamageBehaivour(gameObject.GameObject);

break;

case 2:

bonus = new UPHPBehaivour(gameObject.GameObject);

break;

case 3:

bonus = new DownHPBehaivour(gameObject.GameObject);

break;

case 4:

bonus = new UpDamageBehaivour(gameObject.GameObject);

break;

case 5:

bonus = new FullGasBehaivour(gameObject.GameObject);

break;

}

Console.WriteLine(bonus.GetType());

gameObject.GameObject.AddBehaivour(bonus);

bonus.SetTarget(gameObject);

return bonus;

}

}

}

**Файл DownDamageBehaivour.cs:**

using GameObjects;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class DownDamageBehaivour : BonusBehaivour

{

public DownDamageBehaivour(GameObject mesh) : base(mesh)

{

}

public override void SetBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetDeltaDamage(10f);

}

}

public override void RemoveBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetDeltaDamage(-10f);

}

}

}

}

**Файл DownHPBehaivour.cs:**

using GameObjects;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class DownHPBehaivour : BonusBehaivour

{

public DownHPBehaivour(GameObject mesh) : base(mesh)

{

}

public override void SetBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetNewHP(-10f);

}

}

public override void RemoveBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetNewHP(10f);

}

}

}

}

**Файл FullGasBehaivour.cs:**

using GameObjects;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class FullGasBehaivour : BonusBehaivour

{

public FullGasBehaivour(GameObject mesh) : base(mesh)

{

\_hasTimer = false;

}

public override void SetBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetNewGas(50f);

}

}

}

}

**Файл UpDamageBehaivour.cs:**

using GameObjects;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class UpDamageBehaivour : BonusBehaivour

{

public UpDamageBehaivour(GameObject mesh) : base(mesh)

{

}

public override void SetBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetDeltaDamage(10f);

}

}

public override void RemoveBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetDeltaDamage(-10f);

}

}

}

}

**Файл UPHPBehaivour.cs:**

using GameObjects;

using OutputWindow.Scripts.TankScripts;

namespace OutputWindow.Scripts.BonusScript

{

public class UPHPBehaivour : BonusBehaivour

{

public UPHPBehaivour(GameObject mesh) : base(mesh)

{

}

public override void SetBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetNewHP(10f);

}

}

public override void RemoveBonus()

{

if (\_target is PlayerTankScript player)

{

player.SetNewHP(-10f);

}

}

}

}

**Файл GameScript.cs:**

using Behaivourus;

using GameObjects;

using OutputWindow.Game;

using OutputWindow.Scripts.TankScripts;

namespace Server.Script\_s.GameScript

{

public class GameScript : MonoBehaivour

{

private List<PlayerTankScript> \_players = new List<PlayerTankScript>();

private bool \_gameIsStoped = false;

public GameScript(GameObject mesh) : base(mesh)

{

\_copyScript = false;

}

public override void Start(object sender, EventArgs e)

{

var list = SingeltonEngine.GameEngine.GameObjects;

foreach (var gameObject in list)

{

foreach (var item in gameObject.MonoBehaivours)

{

if (item is PlayerTankScript script)

{

\_players.Add(script);

break;

}

}

}

}

public override void Update(object sender, EventArgs e)

{

if (!\_gameIsStoped)

{

foreach (var item in \_players)

{

if (item.CurentGas <= 0 || item.CurrentHealth <= 0)

{

StopGame();

SingeltonEngine.GameEngine.Restart();

break;

}

}

}

}

private void StopGame()

{

\_gameIsStoped = true;

foreach (var player in \_players)

{

player.TankDestroy();

}

\_players.Clear();

}

}

}

**Файл ShellBehaivour.cs:**

using Behaivourus;

using GameObjects;

using OpenTK;

using OutputWindow.Game;

namespace OutputWindow.Scripts.ShellScript

{

public class ShellBehaivour : MonoBehaivour

{

public int Target;

private Vector3 Direction;

public float SizeOfDamage;

public float VeilocityOfShell;

public ShellBehaivour(GameObject mesh) : base(mesh)

{

GameObject.Collision.isTriger = true;

GameObject.SimpleCollision.TurnOfOn();

\_copyScript = false;

}

public void InitScript(Vector3 direction, int target, float damage, float veilocity)

{

Target = target;

Direction = direction;

SizeOfDamage = damage;

VeilocityOfShell = veilocity;

}

public override void Colision(object sender, EventArgs e)

{

if (sender is Collision collision)

{

if (collision.gameObject.NameObject == "Wall")

{

this.Delete();

SingeltonEngine.ConectionManager.AddObjectToSend(this.GameObject, Network.StateObject.TypeOfMesage.Delete);

}

else if (Target == collision.gameObject.GameObjectID)

{

this.Delete();

SingeltonEngine.ConectionManager.AddObjectToSend(this.GameObject, Network.StateObject.TypeOfMesage.Delete);

}

}

}

int \_every = 2;

int \_count = 0;

public override void Update(object sender, EventArgs e)

{

GameObject.SimpleCollision.AddImpact(VeilocityOfShell, Direction);

if (\_count == 3)

{

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

\_count = 0;

}

\_count++;

}

public override MonoBehaivour Clone()

{

return base.Clone();

}

}

}

**Файл PlayerTankScript.cs:**

using Behaivourus;

using GameObjects;

using OpenTK;

using OutputWindow.Game;

using Server.GameLogic.InputController;

using Server.MonoBehaivourus;

using Vector3 = OpenTK.Vector3;

namespace OutputWindow.Scripts.TankScripts

{

public class PlayerTankScript : MonoBehaivour, OnlinePlaybleBehaivour

{

public float CurrentArmor = 0f;

public float MaxHealth = 100;

public float CurrentHealth = 100;

public float CurrentDamage = 10f;

public float MaxGas = 300;

public float CurentGas = 150;

public float Veilocity = 0.5f;

public float DirX;

public float DirY;

public float DirZ;

public int Target;

private Vector3 DirectionOfMove => new Vector3(DirX, DirY, DirZ);

private float \_timerToFire = 0;

private bool \_canFire = true;

private bool \_canMove = true;

private int \_playerToken = 0;

public PlayerTankScript(GameObject mesh) : base(mesh)

{

GameObject.SimpleCollision.TurnOfOn();

}

public void SetToken(int token)

{

\_playerToken = token;

}

public int GetToken()

{

return \_playerToken;

}

public void TankDestroy()

{

\_canFire = false;

\_canMove = false;

}

public override void Start(object sender, EventArgs e)

{

if (GameObject.NameObject.Contains("TankT72\_Green"))

{

GameObject gameObject = SingeltonEngine.GameEngine.GetGameObjectByName("TankT72\_Red");

if (gameObject != null)

{

Target = gameObject.GameObjectID;

}

}

else

{

GameObject gameObject = SingeltonEngine.GameEngine.GetGameObjectByName("TankT72\_Green");

if (gameObject != null)

{

Target = gameObject.GameObjectID;

}

}

}

public override void Colision(object sender, EventArgs e)

{

if (sender is Collision collision)

{

foreach (var monoBehaivour in collision.gameObject.MonoBehaivours)

{

if (monoBehaivour is ShellScript.ShellBehaivour shellBehaivour)

{

if (shellBehaivour.Target != Target)

{

CurrentHealth = CurrentHealth - shellBehaivour.SizeOfDamage;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

}

}

}

}

}

public override void Update(object sender, EventArgs e)

{

if (CurentGas > 0)

{

}

else

{

CurentGas = 0;

}

if (PlayerInputController.PlayerToken == \_playerToken && \_canMove)

{

if (PlayerInputController.PushedButton(OpenTK.Input.Key.A))

{

DirX = -1;

DirY = 0;

DirZ = 0;

GameObject.Mesh.Roll = -MathHelper.Pi;

GameObject.SimpleCollision.AddImpact(Veilocity, DirectionOfMove);

CurentGas = CurentGas - 0.1f;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

}

else if (PlayerInputController.PushedButton(OpenTK.Input.Key.D))

{

DirX = 1;

DirY = 0;

DirZ = 0;

GameObject.Mesh.Roll = 0;

GameObject.SimpleCollision.AddImpact(Veilocity, DirectionOfMove);

CurentGas = CurentGas - 0.1f;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

}

else if (PlayerInputController.PushedButton(OpenTK.Input.Key.S))

{

DirX = 0;

DirY = -1;

DirZ = 0;

GameObject.Mesh.Roll = 3 \* MathHelper.Pi / 2;

GameObject.SimpleCollision.AddImpact(Veilocity, DirectionOfMove);

CurentGas = CurentGas - 0.1f;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

}

else if (PlayerInputController.PushedButton(OpenTK.Input.Key.W))

{

DirX = 0;

DirY = 1;

DirZ = 0;

GameObject.Mesh.Roll = MathHelper.Pi / 2;

GameObject.SimpleCollision.AddImpact(Veilocity, DirectionOfMove);

CurentGas = CurentGas - 0.1f;

SingeltonEngine.ConectionManager.AddObjectToSend(GameObject, Network.StateObject.TypeOfMesage.Read);

}

if (PlayerInputController.PushedButton(OpenTK.Input.Key.Space) && \_canFire)

{

Fire();

}

}

DelayOfFire();

}

private float \_countOfDelay = 0;

private float \_delayOfFire = 0.25f;

private void DelayOfFire()

{

if (!\_canFire)

{

\_countOfDelay = \_countOfDelay + TimeDeltaHelper.DeltaT;

if (\_countOfDelay > \_delayOfFire)

{

\_canFire = true;

\_countOfDelay = 0;

}

}

}

private void Fire()

{

\_canFire = false;

var shell = CreateShell();

shell.Mesh.Roll = GameObject.Mesh.Roll;

shell.AddBehaivour(new ShellScript.ShellBehaivour(shell));

shell.Mesh.ScaleTo(GameObject.Mesh.Scale / 3);

shell.Mesh.SetPosition(GameObject.Position);

foreach (var monoBehaivour in shell.MonoBehaivours)

{

if (monoBehaivour is ShellScript.ShellBehaivour shellBehaivour)

{

shellBehaivour.InitScript(DirectionOfMove, Target, CurrentDamage, Veilocity \* 1.5f);

}

}

SingeltonEngine.ConectionManager.AddObjectToSend(shell, Network.StateObject.TypeOfMesage.Create);

}

public GameObject CreateShell()

{

string nameObject = "AP";

foreach (var item in SingeltonEngine.ObjectLoader.NameOfObjects)

{

if (item.Contains(nameObject))

{

nameObject = item;

break;

}

}

return SingeltonEngine.ObjectLoader.CreateObject(nameObject);

}

public void SetNewHP(float deltaHP)

{

MaxHealth += deltaHP;

}

public void SetNewGas(float deltaGas)

{

CurentGas += deltaGas;

}

public void SetDeltaArmmor(float deltaArmmor)

{

CurrentArmor += deltaArmmor;

}

public void SetDeltaDamage(float deltaDameg)

{

CurrentDamage += deltaDameg;

}

public override MonoBehaivour Clone()

{

return base.Clone();

}

}

}

**Файл GameClient.cs:**

using Behaivourus;

using GameLoader;

using GameObjects;

using OutputWindow.Network;

namespace Client.Game

{

public class GameClient

{

bool gameStart = false;

EventHandler \_updateHandler;

EventHandler \_deleteHandler;

EventHandler \_startHandler;

private List<GameObject> \_gameObjects = new List<GameObject>();

public List<GameObject> GameObjects => \_gameObjects;

private List<GameObject> \_removeGameObjects = new List<GameObject>();

public void GameState()

{

if (!gameStart)

{

\_startHandler?.Invoke(this, null);

gameStart = true;

}

SetStatet();

\_updateHandler?.Invoke(this, null);

ClearObjects();

TimeDeltaHelper.Update();

}

private Queue<List<StateObject>> \_gameStatet = new Queue<List<StateObject>>();

private void SetStatet()

{

if (\_gameStatet.Count > 0)

{

List<StateObject> stateObjects = \_gameStatet.Dequeue();

GameObject gameObject;

lock (\_gameObjects)

{

foreach (var state in stateObjects)

{

gameObject = GetObjectByID(state.ObjectID);

if (state.Type == StateObject.TypeOfMesage.Create)

{

if (gameObject == null)

{

gameObject = state.CreateObject();

AddGameObject(gameObject);

}

}

else if (state.Type == StateObject.TypeOfMesage.Read && gameObject != null)

{

state.SetStateObject(gameObject);

}

else if (state.Type == StateObject.TypeOfMesage.Delete && gameObject != null)

{

RemoveGameObject(gameObject);

}

}

}

}

}

public void UpdateState(List<StateObject> stateObjects)

{

lock (\_gameObjects)

{

\_gameStatet.Enqueue(stateObjects);

}

}

public GameObject GetObjectByID(int ID)

{

foreach (var gameObject in \_gameObjects)

{

if (gameObject.GameObjectID == ID)

{

return gameObject;

}

}

return null;

}

public GameObject GetGameObjectByName(string name)

{

foreach (GameObject gameObject in \_gameObjects)

{

if (name == gameObject.NameObject)

{

return gameObject;

}

}

return null;

}

public void RemoveListeners(MonoBehaivour monoBehaivour)

{

\_startHandler -= monoBehaivour.Start;

\_updateHandler -= monoBehaivour.Update;

}

//Добавление объекта в игровой цикл

public void AddGameObject(GameObject gameObject)

{

lock (\_gameObjects)

{

\_gameObjects.Add(gameObject);

}

foreach (var item in gameObject.MonoBehaivours)

{

AddListenerToUpdate(item);

}

}

public void RemoveGameObject(GameObject gameObject)

{

lock (\_removeGameObjects)

{

\_removeGameObjects.Add(gameObject);

}

}

public void AddListenerToUpdate(MonoBehaivour monoBehaivour)

{

\_updateHandler += monoBehaivour.Update;

\_startHandler += monoBehaivour.Start;

}

public void ClearObjects()

{

lock (\_removeGameObjects)

{

lock (\_gameObjects)

{

if (\_removeGameObjects.Count > 0)

{

foreach (var gameObject in \_removeGameObjects)

{

foreach (var monoBehaivour in gameObject.MonoBehaivours)

{

\_updateHandler -= monoBehaivour.Update;

\_startHandler -= monoBehaivour.Start;

gameObject.Collision.RemoveListener(monoBehaivour);

\_gameObjects.Remove(gameObject);

monoBehaivour.Delete();

}

}

\_removeGameObjects.Clear();

}

}

}

}

}

}

**Файл UserInterface2D.cs:**

using GameLoader;

namespace Server.GameObject.Interface

{

public class UserInterface2D

{

public string Text = "";

public int X\_Position = 0;

public int Y\_Position = 0;

public UserInterface2D()

{

SingeltonEngine.Render.AddToRender2DUI(this);

}

}

}

**Файл InputControlller.cs:**

using OpenTK;

using OpenTK.Input;

namespace IputController

{

public static class InputControlller

{

public static Dictionary<string, bool> KeyValuePairsPush => \_keyValuePairsPush;

private static Dictionary<string, bool> \_keyValuePairsPush = new Dictionary<string, bool>();

public static Dictionary<string, bool> KeyValuePairsUp => \_keyValuePairsUp;

private static Dictionary<string, bool> \_keyValuePairsUp = new Dictionary<string, bool>();

public static Dictionary<string, bool> KeyValuePairsDown => \_keyValuePairsDown;

private static Dictionary<string, bool> \_keyValuePairsDown = new Dictionary<string, bool>();

static GLControl \_gLControl;

public static void InitInputConroller(GLControl gLControl)

{

\_gLControl = gLControl;

foreach (Key key in Enum.GetValues(typeof(Key)))

{

\_keyValuePairsPush[key.ToString()] = false;

\_keyValuePairsUp[key.ToString()] = false;

\_keyValuePairsDown[key.ToString()] = false;

}

\_gLControl.KeyDown += \_gLControl\_KeyDown;

\_gLControl.KeyUp += \_gLControl\_KeyUp;

\_gLControl.KeyPress += \_gLControl\_KeyPress;

}

private static void \_gLControl\_KeyPress(object? sender, System.Windows.Forms.KeyPressEventArgs e)

{

string keyPress = e.KeyChar.ToString().ToUpper();

\_keyValuePairsPush[keyPress] = true;

}

private static void \_gLControl\_KeyUp(object? sender, KeyEventArgs e)

{

string keyUp = e.KeyCode.ToString();

\_keyValuePairsPush[keyUp] = false;

\_keyValuePairsUp[keyUp] = true;

}

private static void \_gLControl\_KeyDown(object? sender, KeyEventArgs e)

{

string keyDown = e.KeyData.ToString();

\_keyValuePairsPush[keyDown] = true;

\_keyValuePairsDown[keyDown] = true;

}

public static bool PushedButton(Key key)

{

var t = key.ToString();

return \_keyValuePairsPush[key.ToString()];

}

public static bool UpButton(Key key)

{

var v = \_keyValuePairsUp[key.ToString()];

return \_keyValuePairsUp[key.ToString()];

}

public static bool DownButton(Key key)

{

return \_keyValuePairsDown[key.ToString()];

}

public static void UpdateInput()

{

// Сбрасываем состояния Up и Down на каждом кадре

foreach (var key in \_keyValuePairsUp.Keys.ToList())

{

\_keyValuePairsUp[key] = false;

\_keyValuePairsDown[key] = false;

}

}

}

}

**Файл RenderObjects.cs:**

using GameLoader;

using GameObjects;

using OpenTK;

using OpenTK.Graphics.OpenGL;

using Server.GameObject.Interface;

using System.Drawing.Imaging;

using PixelFormat = OpenTK.Graphics.OpenGL.PixelFormat;

namespace RenderObject

{

public class RenderObjects

{

List<UserInterface2D> \_userInterface2D = new List<UserInterface2D>();

GLControl \_glControl;

Form \_form;

public RenderObjects(GLControl gLControl, Form form)

{

\_glControl = gLControl;

\_form = form;

\_glControl.Paint += RenderCicle;

\_glControl.Load += Control\_Resize;

\_glControl.Resize += Control\_Resize;

\_glControl.Dock = DockStyle.Fill;

\_form.Controls.Add(\_glControl);

}

public void AddToRender2DUI(UserInterface2D interface2D)

{

\_userInterface2D.Add(interface2D);

}

public void RemoveFromRender2DUI(UserInterface2D userInterface2D)

{

lock (\_userInterface2D)

{

\_userInterface2D.Remove(userInterface2D);

}

}

public void RenderCicle(object sender, PaintEventArgs eventPaint)

{

\_glControl.MakeCurrent();

GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

GL.ClearColor(0.1f, 0.1f, 0.1f, 1);

GL.PushMatrix();

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(0, \_form.Width, \_form.Height, 0, 1, 1);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadIdentity();

var curDraw = new List<GameObject>(SingeltonEngine.GameClient.GameObjects);

foreach (GameObject gameObject in curDraw)

{

if (gameObject.IsVisible)

RenderObject(gameObject);

}

GL.PopMatrix();

lock (\_userInterface2D)

{

foreach (var UI in \_userInterface2D)

{

RenderUI2D(UI);

}

}

\_glControl.SwapBuffers();

}

private void RenderUI2D(UserInterface2D userInterface2D)

{

Font font = new Font("Arial", 14);

int textureId;

// Сохраняем текущую матрицу

GL.PushMatrix();

//// Переключаемся в режим 2D

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(0, \_form.Width, \_form.Height, 0, -1, 1);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadIdentity();

// Создаем текстуру

GL.GenTextures(1, out textureId);

// Измеряем размер текста

Bitmap tempBitmap = new Bitmap(1, 1);

Graphics tempGraphics = Graphics.FromImage(tempBitmap);

SizeF textSize = tempGraphics.MeasureString(userInterface2D.Text, font);

tempGraphics.Dispose();

tempBitmap.Dispose();

// Создаем текстуру с текстом

int textureWidth = (int)Math.Ceiling(textSize.Width);

int textureHeight = (int)Math.Ceiling(textSize.Height);

UpdateTextTexture(userInterface2D.Text, font, textureId, textureWidth, textureHeight);

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Blend);

GL.BlendFunc(BlendingFactor.SrcAlpha, BlendingFactor.OneMinusSrcAlpha);

GL.BindTexture(TextureTarget.Texture2D, textureId);

float x = userInterface2D.X\_Position;

float y = userInterface2D.Y\_Position;

GL.Begin(PrimitiveType.Quads);

GL.Color4(1.0f, 1.0f, 1.0f, 1.0f); // Устанавливаем белый цвет с полной непрозрачностью

GL.TexCoord2(0, 0); GL.Vertex2(x, y);

GL.TexCoord2(1, 0); GL.Vertex2(x + textureWidth, y);

GL.TexCoord2(1, 1); GL.Vertex2(x + textureWidth, y + textureHeight);

GL.TexCoord2(0, 1); GL.Vertex2(x, y + textureHeight);

GL.End();

GL.Disable(EnableCap.Blend);

GL.Disable(EnableCap.Texture2D);

GL.PopMatrix();

GL.DeleteTexture(textureId);

}

private void UpdateTextTexture(string text, Font font, int textureId, int width, int height)

{

// Создаем bitmap с текстом

if (width != 0 && height != 0)

using (Bitmap bitmap = new Bitmap(width, height))

using (Graphics g = Graphics.FromImage(bitmap))

{

g.Clear(Color.Transparent);

g.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAlias;

g.DrawString(text, font, Brushes.Aqua, new PointF(0, 0));

BitmapData data = bitmap.LockBits(

new Rectangle(0, 0, bitmap.Width, bitmap.Height),

ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

GL.BindTexture(TextureTarget.Texture2D, textureId);

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba,

data.Width, data.Height, 0,

OpenTK.Graphics.OpenGL.PixelFormat.Bgra,

PixelType.UnsignedByte, data.Scan0);

bitmap.UnlockBits(data);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Linear);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureWrapS, (int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureWrapT, (int)TextureWrapMode.Clamp);

}

}

private void RenderObject(GameObject gameObject)

{

int texture = GL.GenTexture();

GL.BindTexture(TextureTarget.Texture2D, texture);

GL.PixelStore(PixelStoreParameter.UnpackAlignment, 1);

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, gameObject.Texture.GetWidth(), gameObject.Texture.GetHeight(),

0, PixelFormat.Bgra, PixelType.UnsignedByte, gameObject.Texture.GetData());

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapS, (int)TextureWrapMode.Repeat);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureWrapT, (int)TextureWrapMode.Repeat);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter, (int)TextureMinFilter.Nearest);

GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter, (int)TextureMagFilter.Nearest);

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Blend);

GL.BlendFunc((BlendingFactor)BlendingFactorSrc.SrcAlpha, (BlendingFactor)BlendingFactorDest.OneMinusSrcAlpha);

GL.Begin(PrimitiveType.Quads);

Matrix4 transformMT = gameObject.Matrix;

for (int i = 0; i < gameObject.Mesh.Vercites.Length / 1; i++)

{

Vector3 vercite = Vector3.TransformPosition(gameObject.Mesh.Vercites[i], transformMT);

GL.Vertex3(vercite);

GL.TexCoord2(gameObject.Texture.TextureCords[i]);

}

GL.End();

GL.DeleteTexture(texture);

GL.Begin(PrimitiveType.Lines); // Начало отрисовки линии

if (gameObject.Collision.Detected)

{

var verctMax = Vector3.TransformPosition(gameObject.Collision.Max, gameObject.Matrix);

var verctMin = Vector3.TransformPosition(gameObject.Collision.Min, gameObject.Matrix);

GL.Color3(1.0f, 1.0f, 1.0f); // Установка цвета линии (красный)

GL.Vertex2(verctMax.X, verctMax.Y); // Начальная точка линии

GL.Vertex2(verctMin.X, verctMin.Y); // Конечная точка линии

}

GL.End();

}

// Расположение окна

private void ScreenPosition()

{

// Получаем размеры экрана

Screen screen = Screen.PrimaryScreen;

double screenWidth = screen.Bounds.Width;

double screenHeight = screen.Bounds.Height;

// Получаем размеры окна

double windowWidth = \_form.Width;

double windowHeight = \_form.Height;

// Вычисляем координаты окна

double x = (screenWidth - windowWidth) / 2;

double y = (screenHeight - windowHeight) / 2;

// Устанавливаем новые координаты окна

\_form.Left = (int)x;

\_form.Top = (int)y;

}

// отвечает за размер объектов от размеров окна

private void Control\_Resize(object sender, EventArgs e)

{

if (\_glControl.Width <= 1080)

{

\_form.Width = 600;

\_form.Height = 600;

GL.Viewport(0, 0, \_glControl.Width, \_glControl.Height);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(-1, 1, -1, 1, -1, 1);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadIdentity();

}

else

{

GL.Viewport(\_glControl.Width / 4, 0, 1080, 1020);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(-1, 1, -1, 1, -1, 1);

GL.MatrixMode(MatrixMode.Modelview);

GL.LoadIdentity();

}

}

public void UpdateScreen()

{

\_glControl.Invalidate();

}

public void OnOfRender(bool isWork)

{

\_glControl.Visible = isWork;

}

}

}

**Файл UIScript.cs:**

using Behaivourus;

using GameObjects;

using OutputWindow.Scripts.TankScripts;

using Server.GameObject.Interface;

namespace Client.Scripts.UI\_Script

{

public class UIScript : MonoBehaivour

{

private UserInterface2D \_userInterface2D;

private PlayerTankScript \_playerTankScript;

public UIScript(GameObject mesh) : base(mesh)

{

}

public void SetUserInterface(UserInterface2D userInterface2D, PlayerTankScript playerTankScript)

{

\_userInterface2D = userInterface2D;

\_playerTankScript = playerTankScript;

}

public override void Update(object sender, EventArgs e)

{

\_userInterface2D.Text = "";

\_userInterface2D.Text = "HP: " + \_playerTankScript.CurrentHealth + "\n";

\_userInterface2D.Text = \_userInterface2D.Text + "Fuel Gas: " + \_playerTankScript.CurentGas + "\n";

}

}

}

**Файл TankBattle.cs:**

using IputController;

using NetWorkConnector;

using OpenTK;

using GameLoader;

using RenderObject;

using System.Net;

using Client.Game;

namespace Client

{

public partial class TankBattle : Form

{

GLControl \_glControl;

RenderObjects \_render;

//GameEngine \_engine;

ConectionManager \_conectionManager;

Panel \_mainMenu;

Panel \_secondMenu;

Panel \_thirdMenu;

public bool IsHost = true;

public bool Stop = false;

public TankBattle()

{

InitializeComponent();

this.Name = "Tank Battle";

object renderMonitor = new object();

\_glControl = new GLControl();

InputControlller.InitInputConroller(\_glControl);

\_render = new RenderObjects(\_glControl, this);

LoaderTexture loaderTexture = new LoaderTexture();

loaderTexture.LoadTextures();

string pathToMap = AppDomain.CurrentDomain.BaseDirectory + "MetaData\\Map's\\Map1.bmp";

ObjectLoader objectLoader = new ObjectLoader(loaderTexture.Textures);

GameClient gameClient = new GameClient();

\_conectionManager = new ConectionManager();

SingeltonEngine.SetConectionMenager(\_conectionManager);

SingeltonEngine.SetRender(\_render);

SingeltonEngine.SetObjectLoader(objectLoader);

SingeltonEngine.SetLoaderTextures(loaderTexture);

SingeltonEngine.SetGameClient(gameClient);

\_glControl.Visible = false;

InitMenu();

InitSecondMenu();

InitThirdMenu();

}

public Panel InitMenu()

{

\_mainMenu = new Panel();

var buttonStart = new Button();

buttonStart.Text = "Start Game";

buttonStart.Name = "StartGameButton";

buttonStart.Size = new Size(100, 40);

buttonStart.Location = new Point(

(this.Width - buttonStart.Width) / 2,

(this.Height / 2) - buttonStart.Height / 2 + this.Height / 10

);

buttonStart.Click += ButtonStart\_Click;

var buttonEnd = new Button();

buttonEnd.Text = "Exit";

buttonEnd.Name = "ExitButton";

buttonEnd.Size = new Size(100, 40);

buttonEnd.Location = new Point(

(this.Width - buttonEnd.Width) / 2,

(this.Height / 2) - buttonEnd.Height / 2 - this.Height / 10

);

buttonEnd.Click += ButtonEnd\_Click;

\_mainMenu.Dock = DockStyle.Fill;

\_mainMenu.Controls.Add(buttonStart);

\_mainMenu.Controls.Add(buttonEnd);

this.Controls.Add(\_mainMenu);

return \_mainMenu;

}

public Panel InitSecondMenu()

{

\_secondMenu = new Panel();

var buttonNotHost = new Button();

buttonNotHost.Text = "Not a host";

buttonNotHost.Name = "INotHOSTButton";

buttonNotHost.Size = new Size(100, 40);

buttonNotHost.Location = new Point(

(this.Width - buttonNotHost.Width) / 2,

(this.Height / 2) - buttonNotHost.Height / 2 - this.Height / 10

);

buttonNotHost.Click += ButtonNotHost\_Click;

\_secondMenu.Dock = DockStyle.Fill;

\_secondMenu.Controls.Add(buttonNotHost);

\_secondMenu.Visible = false;

this.Controls.Add(\_secondMenu);

return \_secondMenu;

}

public Panel InitThirdMenu()

{

\_thirdMenu = new Panel();

Label labelIP = new Label();

labelIP.Text = "Input IP Address";

labelIP.AutoSize = true;

labelIP.Location = new Point((this.Width - labelIP.Width) / 2, (this.Height / 2) - this.Height / 5);

\_textBoxIP = new TextBox();

\_textBoxIP.Size = new Size(120, 25);

\_textBoxIP.Location = new Point((this.Width - \_textBoxIP.Width) / 2, (this.Height / 2) - this.Height / 10);

\_textBoxIP.Text = "127.0.0.1";

Label labelPORT = new Label();

labelPORT.Text = "Input PORT";

labelPORT.AutoSize = true;

labelPORT.Location = new Point((this.Width - labelPORT.Width) / 2, (this.Height / 2));

\_textBoxPort = new TextBox();

\_textBoxPort.Size = new Size(120, 25);

\_textBoxPort.Location = new Point((this.Width - \_textBoxPort.Width) / 2, (this.Height / 2) + this.Height / 10);

\_textBoxPort.Text = "2000";

\_button = new Button();

\_button.Text = "Connect to Server";

\_button.Size = new Size(100, 40);

\_button.Location = new Point((this.Width - \_button.Width) / 2, (this.Height / 2) + this.Height / 5);

\_thirdMenu.Dock = DockStyle.Fill;

\_thirdMenu.Controls.Add(\_button);

\_thirdMenu.Controls.Add(labelIP);

\_thirdMenu.Controls.Add(labelPORT);

\_thirdMenu.Controls.Add(\_textBoxIP);

\_thirdMenu.Controls.Add(\_textBoxPort);

this.Controls.Add(\_thirdMenu);

return \_thirdMenu;

}

private void ButtonEnd\_Click(object? sender, EventArgs e)

{

throw new NotImplementedException();

}

TextBox \_textBoxIP;

TextBox \_textBoxPort;

Button \_button;

string IP => \_textBoxIP.Text;

string PORT => \_textBoxPort.Text;

private void Button\_Click\_ConnectToHost(object? sender, EventArgs e)

{

IPAddress ip = IPAddress.Parse(IP);

IPEndPoint iPEndPoint = new IPEndPoint(ip, int.Parse(PORT));

SingeltonEngine.ConectionManager.SetConnection(iPEndPoint);

\_glControl.Visible = true;

}

private void ButtonNotHost\_Click(object? sender, EventArgs e)

{

IsHost = false;

\_mainMenu.Visible = false;

\_secondMenu.Visible = false;

\_thirdMenu.Visible = true;

\_button.Text = "Connect";

\_button.Click += Button\_Click\_ConnectToHost;

}

private void ButtonStart\_Click(object? sender, EventArgs e)

{

\_mainMenu.Visible = false;

\_secondMenu.Visible = true;

\_thirdMenu.Visible = false;

}

}

}

# **ПРИЛОЖЕНИЕ Б**

(обязательное)

**UI и графические ресурсы игры**

На рисунке Б.1 игровое поле игры «Танковая Дуэль».

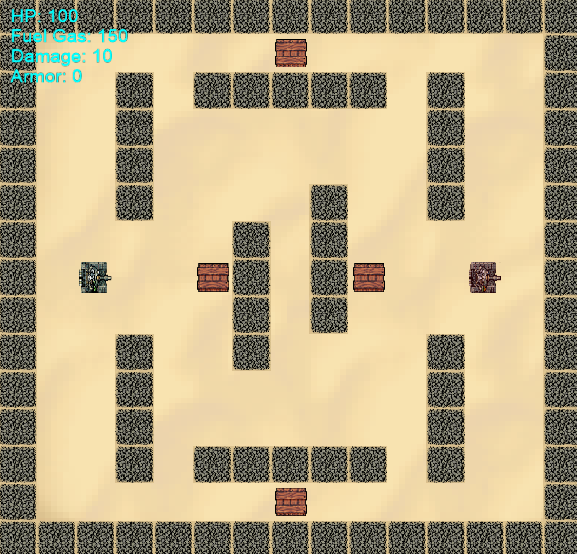


Рисунок Б.1 – Игровое поле игры

На рисунке Б.3 изображена текстура снаряда

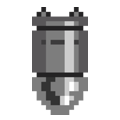


Рисунок Б.2 – текстура снаряда

На рисунке Б. 3 изображена текстура ящика с бонусом

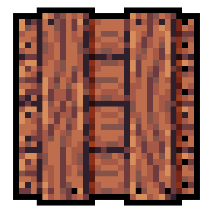


Рисунок Б.3 – Текстура ящика

На рисунке Б. 4 изображен изображена текстура земли.



Рисунок Б.4 – Текстура земли

На рисунке Б. 5 изображена текстура танка номер 1.



Рисунок Б.5 – Текстура танка номер 1

На рисунке Б.6 изображена текстура танка номер 2.



Рисунок Б.6 – Текстура танка номер 2

На рисунке Б.7 изображен текстура стены

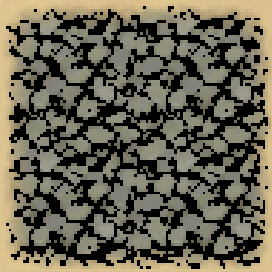


Рисунок Б.7 – Текстура стены

# **ПРИЛОЖЕНИЕ В**

(обязательное)

**Инструкция пользователя**

1. Введение.

Это руководство пользователя предназначено для обеспечения базовых навыков использования игрового приложения «Танковая Дуэль».

1. Назначение и условия применения.

Приложение разработано для игры в танковую дуэль на двух компьютерах для двух игроков. Функционал включает в себя игровое поле в виде пустыни с каменными стенами, перемещение игроков по игровому полю и генерацию призов.

Для корректной работы приложения необходимо удовлетворение следую­щих требований:

* операционная система *Windows* *XP* и выше;
* стандартная клавиатура, мышь в качестве устройств ввода, а также монитор в качестве устройства вывода;
* подключение к локальной сети.

1. Подготовка к работе.

Для установки приложения необходимо загрузить готовую программу с игрой и установить библиотеку *OpenGL* последней версии. В случае непредвиденных ошибок рекомендуется завершить процесс в диспетчере задач и запустить программу заново. Для того что бы запустить саму игру необходимо запустить сервер. Предварительно в конфигурационном файле *ConfServer*.*json* вписать *IP* и *PORT* сервера.

1. Описание операций.

Игровое поле представляет собой пустыню с каменными стенами. Игроки управляют танками и могут перемещаться по игровому полю с помощью клавиш *W*, *S*, *A*, *D*. Стрельба снаря­дами осуществляется при нажатии клавиши *Space*. В игре будут случайно генерироваться положительные и штраф­ные призы, такие как боеприпасы, здоровье, топливо, ускорение, усиление вы­стрела, броня и понижение скорости. Топливо тратится при передвижении, и если игрок не восполнит его, то танк больше не сможет ездить.

1. Аварийные ситуации.

При возникновении аварийных ситуаций следует заскринить код ошибки или же причину ошибки написанную текстом. После этого описать после чего произошла данная ошибка, какие действия ее создали. После двух этих пунктов отправить условный отчет об ошибке. Так же если ошибка возникает при старте игры или вообще ничего не происходит следует попробовать переустановить или же установить все пакет *Microsoft* *Visual* *C*++. На сайте Microsoft есть сборка данных компонентов. Следует так же обновить *OpenGL* до последней или предпоследний версии. Если данные манипуляции не привили к успех следует переустановить приложение. Если даже это не помогло то следует сделать то что описано в первом абзаце.

# **ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство программиста**

1. Общие сведения о программе. Описание структуры и компонентов приложения.

Приложение состоит из четырех компонентов:

* *UI*;
* рендер;
* игрового движка;
* игровых объектов;
* модуля связи.

*UI* представлен *WFA*. Рендер представлен *OpenGL*, библиотекой *OpenTK*. Игровой движок принимает решения об всех действиях, связанных с физикой и коллизиями. Скрипты управляют объектами, посылая команды движку или меняя данные объекта. Технологии и фреймворки, информация о технологиях и фреймворках, используемых в приложении. Данное приложение использует стек технологий таких как:

* язык программирования *с*#;
* .*NET* *Core 6.0.0*;
* библиотеку *opentk*;
* библиотеку *GLControl*;
* *winsocet*;

1. Разработка. Описание процесса разработки, включая рекомендации по написанию кода, тестированию.

Процесс разработки при использовании объектно-ориентированного подхода начинается с рассмотрения предметной области. Первый шаг создание класса, который управляет всеми объектами. После данного класса нужен объект, который сможет отобразить игровые объекты, т.е. какой-то рендер, в данном случае это класс, который использует функции из библиотеки *OpenTK*. Для сетевого взаимодействия разработан модуль, который использует протокол *TCP*/*IP*. Так же для данного модуля созданы контейнеры для передачи данных по сети.

При дополнении функционала в данном приложении это делать через классы типа *MonoBehaivour*. Это облегчит тестирование прило­жения т.к. данный класс является скриптом, который отвечает за логику отдельного игрового объекта.

*API*. Описание интерфейсов программирования приложения и инструкции по их использованию. Интерфейс представлен игровым полем. Меню отсутствует в следствии простоты приложения. При его усложнении его можно сделать при помощи средств *Visual* *Studio*. Данный интерфейс можно добавить в уже готовую *WFA* форму.

# **ПРИЛОЖЕНИЕ Д**

(обязательное)

**Руководство системного программиста**

1. Архитектура игры. Описание структуры игры, используемые компоненты и связи между ними.

Приложение состоит из четырех компонентов:

* *UI*;
* рендер;
* игрового движка;
* игровых объектов;
* модуля связи.

*UI* представлен *WFA*. Рендер представлен *OpenGL*, библиотекой *OpenTK*. Игровой движок принимает решения об всех действиях, связанных с физикой и коллизиями. Скрипты управляют объектами, посылая команды движку или меняя данные объекта. Технологии и фреймворки, информация о технологиях и фреймворках, используемых в приложении. Данное приложение использует стек технологий таких как:

1. Технологии и инструменты. Информация о технологиях и инструментах, используемых при разработке игры.

При разработке использовалась среда *Visual* *Studio* 2022. В качестве языка программирования был выбран *C*#, инструмент для отображения картинки был выбран *OpenGL* и его библиотеки *OpenTK* и *GLControl*. В качестве основы при­ложения был выбран *WFA*. В следствии этого была выбрана библиотека *GLCon­trol* которая может управлять *OpenTK* внутри *WFA*. В процессе разработки был создан собственный движок игры. Соединение обеспечивается при помощи *winsocket*, которые предоставляются библиотекой *Socket* в .*NET* *Core* 6.

1. Системные требования. Информация о необходимых системных требова­ниях для запуска игры:

* системные требования для игры «Танковая Дуэль»:
* графика не ниже *UHD* 620;
* процессор *Athlon* II 640 для *AMD* для *Intel* *Core* 2 *Duo*;
* оперативная память 512 *Mb*;
* место свободного на диске 512 *Mb*;
* 10 мегабит скорости интернета.

1. Оптимизация. Рекомендации по оптимизации.

Для оптимизации игры следует использовать текстуры более низкого разре­шения. Распараллелить вычисления на процессоре на несколько потоков. При усложнении графики или же появления обилия визуальных эффектов следует до­бавить вершинный и фрагментный шейдеры и использовать *Vertex Array Object* (*VAO*) и *Vertex Buffer Object* (*VBO*) по средствам внутренних функций *OpenTK*.

# **ПРИЛОЖЕНИЕ Е**

(обязательное)

**Архитектура приложения**