**СОДЕРЖАНИЕ**

[Введение 4](#_Toc58983534)

[1 Обзор и сравнение программных средств для игрового приложения «Танки» 5](#_Toc58983535)

[1.1 Обзор существующих реализаций игры «Танки» 5](#_Toc58983536)

[1.2 Обзор графических библиотек *OpenGL* и *DirectX* 7](#_Toc58983537)

[1.3 Сравнение графических средств *OpenGL* и *DirectX* 11](#_Toc58983538)

[1.4 Обзор технологии *Windows* *Sockets* 14](#_Toc58983539)

[2 Архитектура и программная реализация игрового приложения «Танки» 15](#_Toc58983540)

[2.1 Определение основных этапов в разработке игрового приложения 15](#_Toc58983541)

[2.2 Программная реализация игрового приложения 20](#_Toc58983542)

[3 Верификация и анализ результатов 25](#_Toc58983543)

[3.1 Эксплуатация сетевого игрового приложения «Танки» 25](#_Toc58983544)

[3.2 Верификация работы приложения «Танки» 29](#_Toc58983545)

[3.3 Модульное тестирование 31](#_Toc58983546)

[Заключение 32](#_Toc58983547)

[Список использованных источников 33](#_Toc58983548)

[Приложение А. Листинг программы 34](#_Toc58983549)

[Приложение Б. Схема иерархии классов 63](#_Toc58983550)

[Приложение В. Руководство пользователя 64](#_Toc58983551)

[Приложение Г. Руководство программиста 66](#_Toc58983552)

[Приложение Д. Руководство системного программиста 67](#_Toc58983553)

# **ВВЕДЕНИЕ**

В настоящее время особенно широкое распространение получили различные компьютерные игры. Они привлекают обширную аудиторию, в которую входят представители всех возрастов и слоёв населения.

Наблюдается стремительное развитие игрового программного обеспечения, постоянное усовершенствование методов реализации игрового процесса, графических представлений, разработка новых методов взаимодействию с приложением. Они оказали столь существенное влияние на общество, что в информационных технологиях отмечена устойчивая тенденция к геймификации для неигрового прикладного программного обеспечения.

Создание компьютерной игры является способом выражения своих идей и создания своего уникального проекта.

Целью курсового проекта является разработка сетевого игрового приложения *Windows* *Form* «Танки» для игры на одном экране двух пользователей друг против друга.

Выполнение курсового проекта потребовало решения следующих задач:

– выполнить аналитический обзор игр жанра «Танки»;

– разработать основные классы;

– использовать шаблоны проектирования «фабричный метод» и «декоратор»;

– разработать программный код игры;

– организовать сетевые коммуникации между пользователями;

– провести тестирование разработанного приложения.

# **1 ОБЗОР И СРАВНЕНИЕ ПРОГРАММНЫХ СРЕДСТВ ДЛЯ ИГРОВОГО ПРИЛОЖЕНИЯ «ТАНКИ»**

## **1.1 Обзор существующих реализаций игры «Танки»**

В игре «Танки» принимают участие два игрока, каждый из которых управляет своим танком с помощью клавиатуры. В начале игры танки имеют одинаковые характеристики. Танки перемещаются по игровому полю с неоднородной территорией. Пули из пулемёта достигают цели моментально, у снарядов скорость ограниченна, так что танки могут уклоняться от выстрелов. Победителем считается игрок, уничтоживший танк соперника.

Далее жанр будет рассмотрен более подробно на примере нескольких аналогичных игр.

Одна из самых известных игр жанра «Танки» – *Battle* *City* (рисунок 1.1), выпущенная в 1985 году фирмой *Namco*.



Рисунок 1.1 – Геймплей игры *Battle City*

Игровой полигон действий в игре представлен в виде сверху. Игрок управляет своим танком и должен уничтожить все вражеские танки на уровне, которые постепенно появляются на игровом поле. Вражеские танки пытаются уничтожить штаб игрока, который представлен внизу игрового поля в виде орла, и его танк. На каждом уровне нужно уничтожить 20 вражеских танков. Если противник сможет разрушить штаб или лишит игрока всех жизней – игра будет окончена.

Игра состоит из 35 основных уровней. Игровое поле представляет собой квадратное поле из 169 секторов (13×13). После прохождения 35 уровней они начинаются заново, но становятся сложнее из-за того, что изменяется набор вражеских танков (набор 30-го уровня), а также из-за более быстрого респауна врагов.

Каждая карта содержит различные типы местности и препятствий:

* кирпичные стены, которые могут быть разрушены от одного или нескольких выстрелов танка;
* стальные стены, которые игрок может разрушить, если он собрал три или более усиленных звезд;
* кусты, снижающие видимость танков и снарядов;
* лёд, который затрудняет управляемость движения танка;
* вода, блокирующая передвижение, но пропускающая снаряды.

Игра содержит четыре типа танков противника, которые различаются скоростью и прочностью. На более поздних уровнях игра становится сложнее, поскольку вражеские танки могут действовать как приманка, чтобы отвлечь игроков от своей базы, чтобы другой танк мог ее уничтожить. Кроме того, мигающие танки могут быть уничтожены для бонусов.

*Tank Battalion* (рисунок 1.2) –игра, выпущенная фирмой *Namco* в 1980*.* Игрок, управляющий танком, должен уничтожать 20 вражеских танков на каждом уровне. Вражеские танки пытаются уничтожить базу игрока (изображенную на карте как орел), а также сам танк игрока. Уровень завершается, когда игрок уничтожает все 20 вражеских танков, но игра заканчивается, если база игрока уничтожена или у него кончились жизни.



Рисунок 1.2 – Геймплей игры *Tank Battalion*

В 1990 году компанией *Namco* была выпущена игра *Tank Force* (рисунок 1.3),сиквелигры *Battle City*.



Рисунок 1.3 – Геймплей игры *Tank Force*

Геймплей очень похож на *Tank Battalion*, за исключением того, что на этот раз одновременно могут играть до двух (на вертикальной модели) или четырех (на коктейле) игроков, появились семь новых («обычных») типов вражеских танков. В игре есть 15 типов бонусов, которые позволяют игрокам увеличить огневую мощь своих танков и повысить свой счет. В отличии от *Tank Battalion* и *Battle City*, каждый уровень имеет размер 17x13 блоков вместо 13x13. Игроки не могут разрушить свои собственные стены штаб-квартиры, поэтому, когда один игрок стреляет в другого, они будут отталкивается назад. У игры есть окончание, которое будет видно после прохождения всех тридцати шести раундов.

## **1.2 Обзор графических библиотек *OpenGL* и *DirectX***

Для решения поставленной задачи существуют графические библиотеки *OpenGL* и *DirectX*. Далее будет рассмотрена характеристика каждой библиотеки и проведен аналитический обзор.

*OpenGL* (*Open* *Graphics* *Library* – открытая графическая библиотека, графическое *API*) – это программный интерфейс, применяемый для получения программно-генерируемых изображений и приложений, использующих двух- или трёхмерные объекты, а также растровые изображения [1].

*OpenGL* появился в результате переработки специализированного *3D*-интерфейса этой фирмы *IrisGL* на универсальный, независимый от платформы. Данный интерфейс позволяет создавать трехмерные графические изображения независимо от используемых оконных интерфейсов, операционной системы и графических устройств.

В основе *OpenGL* лежит библиотека графических функций, предоставляющих разработчику все необходимое для создания и отображения плоских (*2D*) и пространственных (*3D*) моделей и анимации. Эта библиотека, содержащая более 250 функций, обеспечивает средства для управления графическими примитивами, выполнения матричных преобразований, задания источников света, теней, управления цветам.

Основной принцип работы *OpenGL* – получение наборов векторных графических примитивов в виде точек, линий и многоугольников с последующей математической обработкой полученных данных и построением растровой картинки на экране и/или в памяти. Векторные трансформации и растеризация выполняются графическим конвейером, который по сути представляет собой дискретный автомат.

Пример рендеринга треугольника в *OpenGL* представлен на рисунке 1.4.

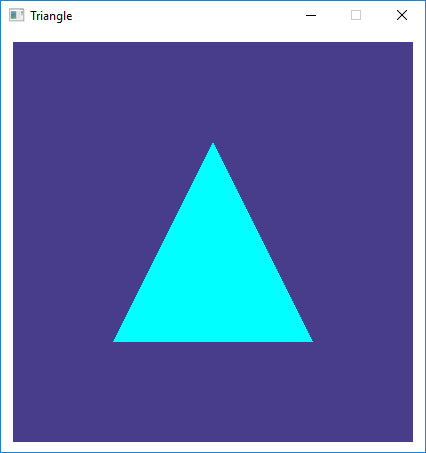


Рисунок 1.4 – Пример рендеринга треугольника в *OpenGL*

*OpenGL* является низкоуровневым процедурным *API*, что вынуждает программиста диктовать точную последовательность шагов, чтобы построить результирующую растровую графику (императивный подход). Это является основным отличием от дескрипторных подходов, когда вся сцена передается в виде структуры данных, которая обрабатывается и строится на экране. С одной стороны, императивный подход требует от программиста глубокого знания законов трёхмерной графики и математических моделей, с другой стороны – даёт свободу внедрения различных инноваций.

*OpenGL* ориентируется на следующие две задачи:

* скрыть сложности адаптации различных 3D-ускорителей*,* предоставляя разработчику единый *API*;
* скрыть различия в возможностях аппаратных платформ, требуя реализации недостающей функциональности с помощью программной эмуляции.

Для подтверждения независимости от языка программирования были разработаны различные варианты привязки (*binding*) функций *OpenGL* или полностью перенесены на другие языки. Например, библиотека *Java 3D*, которая может использовать аппаратное ускорение *OpenGL*. Прямая привязка функций *OpenGL* для *Java* реализована в *Lightweight Java Game Library*. Имеются также варианты привязки *OpenGL* для языков *C++* и *C#*. Для связи *OpenGL* c *C#* используются библиотеки *Tao Framework, OpenTK и SharpGL.*

Основные функции и возможности *OpenGL*:

* геометрические примитивы позволяют задавать математическое описание объектов;
* цветное кодирование в формате *RGBA* или в режиме индексированных цветов;
* просмотр и моделирование позволяют размещать трехмерные объекты на сцене, передвигать камеры вокруг сцены и выбирать точку обзора, для которого выполняется рендеринг;
* текстуры помогают привнести реализм в модели за счет имитации реальных поверхностей (их изображения накладываются поверх «скелета» модели, составленного из многоугольников);
* освещение, которое задаётся функцией описания свойства материала и источников света для сцены;
* двойная буферизация позволяет устранить мерцание при анимации;
* антиалиасинг сглаживает «ступенчатые» края цветовых переходов путём изменения цвета и интенсивности точек вблизи линии резкого перехода;
* затенение по методу Гуро закрашивает *3D*-объектов, при которой сначала рассчитывается освещение вдоль ребер треугольников на основании данных об освещенности вершин, а затем на основании этих данных и направления градиентов изменения цвета закрашивается внутренняя поверхность треугольников;
* *Z*-буферизация используется для сохранения информации о «глубине» трехмерного объекта (*Z*-координаты);
* атмосферные эффекты туман или дымка делают компьютерные изображения более реалистичными;
* альфа-смешение использует значение Альфа-буфера (информации о прозрачности пикселов) как составляющую расширенного цветового кода растрового изображения в формате *RGBA*, позволяя комбинировать цвет обрабатываемого фрагмента с цветом точек, которые уже хранятся в буфере изображения;
* трафаретные планы ограничивают отрисовку на некоторых областях экрана по маске (трафарету);
* cписки отображения позволяют сохранять команды отрисовки в некотором списке для дальнейшего рендеринга;
* полиномные вычисления используются для поддержки *NURBS*-сплайнов, c помощью которых можно рисовать плавные кривые через небольшой набор опорных точек, что исключает необходимость сохранять все промежуточные значения.
* обратная связь, выделение и выбор – функции которые дают возможность создавать приложения, позволяющие пользователю выбирать область экрана или отдельный объект, изображенный на экране;
* растровые примитивы: растровые изображения (прямоугольные картинки, состоящие из точек);
* операции с пикселами;
* различные преобразования – вращение, масштабирование, перемещение, перспективные искажения и т.д. [2, c. 14]

Библиотека *OpenGL* является одной из самых перспективных библиотек трехмерной графики, имеющихся в распоряжении разработчиков. Основное преимущество данной библиотеки – аппаратная и платформерная независимость, позволяющая экономить силы разработчиков при создании приложений, работающих на различных платформах.

Следующее графическое средство, на которое будет сделан обзор – *DirectX*.

*DirectX* – это набор *API*, разработанных для решения задач, связанных с программированием под *Microsoft Windows*. Наиболее широко используется при написании компьютерных игр. Пакет средств разработки *DirectX* под *Microsoft* *Windows* бесплатно доступен на сайте *Microsoft*. Зачастую обновлённые версии *DirectX* поставляются вместе с игровыми приложениями [3].

Сам *DirectX* состоит из следующих модулей:

* *DirectX Graphics*: набор интерфейсов, ранее (до версии 8.0) делившихся на *DirectDraw*:интерфейс вывода растровой графики, и *Direct3D*:интерфейс вывода трёхмерных примитивов;
* *DirectInput*: интерфейс, используемый для обработки данных, поступающих с клавиатуры, мыши, джойстика и прочих игровых контроллеров;
* *DirectPlay*: интерфейс сетевой коммуникации игр;
* *DirectSound*: интерфейс низкоуровневой работы со звуком;
* [*DirectMusic*](https://ru.wikipedia.org/w/index.php?title=DirectMusic&action=edit&redlink=1): интерфейс воспроизведения музыки в форматах *Microsoft*;
* *DirectShow*: интерфейс, используемый для ввода/вывода аудио и/или видео данных;
* *DirectSetup*: часть, ответственная за установку *DirectX*;
* *DirectX* *Media Objects*: реализует функциональную поддержку потоковых объектов;
* *Direct2D*: интерфейс вывода двухмерной графики.

Пример рендеринга куба в *DirectX* представлен на рисунке 1.5.

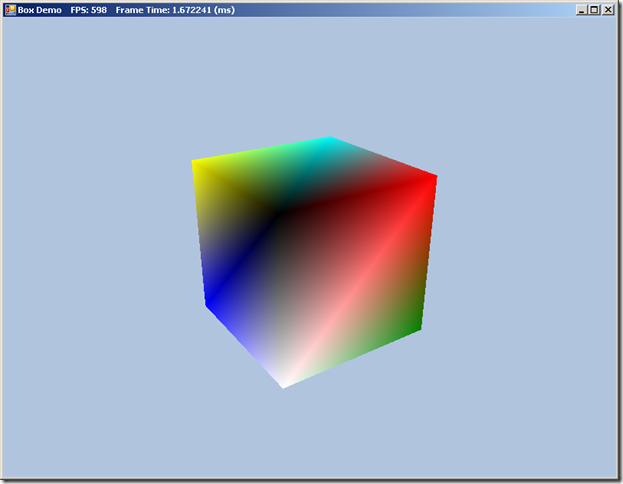


Рисунок 1.5 – Рендеринга куба в *DirectX*

*DirectX* появился именно в помощь разработчикам компьютерных игр и мультимедийных приложений и позволяет создавать программы с очень реалистичной графикой и звуками. Поскольку *DirectX* «заточен» под игры, он имеет инструменты, позволяющие обрабатывать данные, получаемые с устройств управления – с клавиатуры, мыши или игрового джойстика. Изначально нацеленный на разработку видеоигр, *DirectX* стал популярен и в других областях разработки программного обеспечения. К примеру, *DirectX*, наряду с [*OpenGL*](https://ru.wikipedia.org/wiki/OpenGL), получил очень широкое распространение в инженерном/математическом ПО.

## **1.3 Сравнение графических средств *OpenGL* и *DirectX***

Для выбора графической библиотеки к поставленной задаче будет проведено сравнение библиотек *OpenGL* и *DirectX*. Каждая из этих библиотек имеет свои особенности, которые будут рассмотрены более подробно в данном разделе.

Библиотека *OpenGL (Open Graphics Library)* считается одной из лучших библиотек, как для профессионального применения, так и для игр. Однако, у неё существуют конкуренты.

Главных конкурентом считается *Direct3D* из пакета *DirectX*, разработанный компанией *Microsoft*.

Основной особенностью *OpenGL* можно считать простоту. Процесс обработки примитивов контролирует ядро *OpenGL*. Для передачи данных используется процедурная модель. В каждый момент времени состояние *OpenGL* определяется через набор переменных, задающих параметры обработки. Каждый новый переданный треугольник проходит обработку в соответствии с текущим состоянием. Такой механизм весьма эффективен, а код обычно короток и прост, поэтому *OpenGL* подходит для начала изучения трехмерной графики [4, c .21].

Также, преимуществом *OpenGL* является кроссплатформенность – переносимость программ с одной платформы на другую. Благодаря ей программы, написанные с помощью *OpenGL* можно успешно перенести на такие платформы как *Unix*, *Linux*, *SunOS*, *IRIX*, *Windows*, *MacOS*. В свою очередь *Direct3D* может работать только *на Intel* платформах под управлением операционной системы *Windows*.

Однако, в плане объектно-ориентированного подхода *OpenGL* уступает *Direct3D*. *OpenGL* работает по принципу конечного автомата, переходя из одного состояния в другое, совершая при этом какие-то преобразования.

Ещё одним преимуществом *Direct3D* является поддержка дешёвого оборудования, *OpenGL* же поддерживается не на всех графических картах, но для профессиональных ускорителей *OpenGL* является стандартом де-факто.

Структура *DirectX* значительно отличается от *OpenGL*. *DirectX* основан на модели *COM* (*Component Object Model*). Это означает, что в отличие от простого вызова функций эта модель предполагает выполнение некоторых дополнительных действий, связанных с компонентной архитектурой *DirectX*. Недостатком такой архитектуры является использование большого объема кода, необходимого для написания простой программы на *DirectX*. Для упрощения программирования разработчики *Microsoft* создали отдельную библиотеку *DirectX Common Files*, которая скрывает часто используемый код [5].

Архитектура *DirectX* сильно отличается от *OpenGL*, но в их развитии все более заметны тенденции к сближению. Такая ситуация возникает прежде всего потому, что обе библиотеки предназначены для эффективной работы с аппаратурой, и чем ближе их структура будет к "железу", тем меньше времени будет уходить на преобразование команд пользователя в команды аппаратуры.

По поддержке аппаратных функций *OpenGL* и *DirectX* эквиваленты. Новые функции в *OpenGL* доступны через механизм расширений, а в *DirectX* они появляются в новых версиях.

Важным достоинством *OpenGL* является открытый стандарт, разрабатываемый некоммерческой организацией *Khronos* *Group* при участии сообщества. Большинство крупных производителей *GPU* (*nVidia, AMD, Intel*), так или иначе, влияли на *OpenGL*.

Что касается *DirectX*, то здесь ситуация прямо противоположная. Только *Microsoft* может вносить какие-либо изменения в библиотеку. Таким образом, именно *Microsoft* в конечном итоге определяет все пути развития библиотеки, и если путь был выбран неверно, это может быть исправлено только в новой версии.

Между *OpenGL* и *DirectX* существуют различия в том, как каждый *API* обрабатывает объединение вершин для формирования примитивов, которые изложены ниже в таблице 1.1.

Таблица 1.1 – Сравнение характеристик *DirectX и OpenGL*

|  |  |  |
| --- | --- | --- |
| Характеристики | *OpenGL* | *DirectX* |
| Смешивание вершин | Отсутствует | Имеется |
| Множество операционных систем | Имеется | Отсутствует |
| Механизм расширения | Имеется | Имеется |
| Разработка | Многопользовательская разработка | *Microsoft* |
| Точная спецификация | Имеется | Отсутствует |
| Двухсторонне освещение | Имеется | Отсутствует |
| Объёмные текстуры | Имеется | Отсутствует |
| Аппаратно-независимые *Z*-буферы | Имеется | Отсутствует |
| Накопительные буферы | Имеется | Отсутствует |
| Полноэкранное сглаживание | Имеется | Имеется |
| Размытость | Имеется | Имеется |
| Глубина резкости | Имеется | Имеется |
| Стерео-рендеринг | Имеется | Отсутствует |
| Атрибуты размера | Имеется | Отсутствует |
| Выборка | Имеется | Отсутствует |
| Параметрические кривые и поверхности | Имеется | Отсутствует |
| Геометрия кэша | *Display Lists* (Списки отображения) | *Vertex Buffers* (Буфер вершин) |
| Эмуляция системы | Аппаратные средства отсутствуют | Определяет приложение |
| Интерфейс | *Procedure calls* (Процедурные вызовы) | *COM* |
| Обновления | Ежегодно | Ежегодно |
| Исходный код | *Sample* (Выборка) | *SDK Implementation*  (Осуществляется *SDK*) |

Исходя из данных таблицы 1.1, можно сделать вывод, что *OpenGL* подойдёт для тех разработчиков, для которых важна кроссплатформенность, обеспечивающая доступность всех новых функций и на *Windows* *XP*, где *Direct3D* 10/11 отсутствует. Можно отметить, что драйвера *Windows* не имеют поддержки *OpenGL*, поэтому их нужно ставить с сайта производителя, и такая вещь как *SDK* в *OpenGL* отсутствует. Кроме того, *OpenGL* не подходит для работы со звуком, вводом, сетью, поэтому нужно искать соответствующие инструменты, что может отразиться на времени разработки.

## **1.4 Обзор технологии *Windows* *Sockets***

*Windows Sockets* (*Winsock*) представляет собой интерфейс программирования (*API*), который определяет, как сетевое программное обеспечение *Windows* должно получать доступ к сетевым службам. Он определяет стандартный интерфейс между клиентским приложением *Windows* *TCP*/*IP* (например, *FTP*-клиентом или веб-браузером) и базовым стеком протоколов *TCP*/*IP*.

Библиотека *Winsock* поддерживает два вида сокетов:

– синхронные (блокируемые), которые задерживают управление на время выполнения операции;

– асинхронные (неблокируемые), которые возвращают его немедленно, продолжая выполнение в фоновом режиме, и, закончив работу, уведомляют об этом вызывающий код.

Сокеты позволяют работать со множеством протоколов и являются удобным средством межпроцессорного взаимодействия.

Независимо от вида, сокеты делятся на два типа:

– потоковые – работают с установкой соединения, обеспечивая надежную идентификацию обоих сторон и гарантируют целостность и успешность доставки данных, опираются на протокол *TCP*;

– дейтаграммные – работают без установки соединения и не обеспечивают ни идентификации отправителя, ни контроля успешности доставки данных, зато они заметно быстрее потоковых, опираются на протокол *UDP* [6, c. 32].

Выбор того или иного типа сокетов определяется транспортным протоколом, на котором работает сервер, – клиент не может по своему желанию установить с дейтаграммным сервером потоковое соединение.

Для организации сетевых коммуникаций в игре «Танки» использованы потоковые сокеты на основе протокола *TCP*, так как данный протокол гарантирует доставку сообщений и является более надёжным, чем *UDP*.

В данной главе были изучены аналогичные игры, исследованы материалы о графических средствах *DirectX* и *OpenGL*, рассмотрен *WinSock* *API*. Проанализировав данные о этих библиотеках, можно сделать вывод, что для реализации графики в игре «Танки» следует использовать *OpenGL*, так как данная библиотека обеспечивает кроссплатформенность, благодаря которой программа будет свободно переносится на большинстве платформ, а для организации сетевых коммуникаций должен использоваться интерфейс *Windows* *Sockets.*

# **2 АРХИТЕКТУРА И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «ТАНКИ»**

## **2.1 Определение основных этапов в разработке игрового приложения**

Компьютерная игра «Танки» относится к жанру аркада с элементами шу-тера и разрабатывается только в версии для персональных компьютеров.

Игра ориентирована на широкую аудиторию и не содержит ограничивающего контента. Дополнительную привлекательность игра имеет для владельцев не самой современной конфигурации ПК.

В игре принимают участие два игрока, каждый из которых управляет своим танком с помощью клавиатуры. В начале игры танки имеют одинаковые ха-трактористки: скорость движения, броню, вооружение (пушка и пулемёт), запас снарядов и топливо. В игре организовано сетевое взаимодействие, позволяющее запускать её с разных устройств.

Танки перемещаются по игровому полю с неоднородной территорией:

– участки с замедлением скорости;

– участки, на которых невозможно производить выстрелы;

– непреодолимые участки, но через которые можно производить выстрелы;

– участки, через которые нельзя проехать и нельзя стрелять.

Пули из пулемёта достигают цели моментально, у снарядов скорость ограниченна, так что танки могут уклоняться от выстрелов. Победителем считается игрок, уничтоживший танк соперника.

Для разработки игры «Танки» был выбран язык программирования C# и использована графика *OpenGL*.

Язык *C*# основан на строгой компонентной архитектуре и реализует передовые механизмы обеспечения безопасности кода, имеет удобный синтаксис и позволяет без больших трудозатрат разработать приложение.

Использование графики *OpenGL* в *С#* обеспечивает библиотека *OpenTK*, которая реализует графический элемент управления *GLControl*. Библиотека *OpenTK* отобразит графику на этом элементе управления.

В процессе проектирования архитектуры игры решено создать абстрактный класс игровой объект (*GameObject*) для создания объектов наследников *GameAreaObject* – игровые территории и *GamePhysicalObject* – физические игровые объекты. Игровые территории разделяются на три вида: непроходимые, замедляющие и территории, на которых нельзя стрелять. Физические игровые объекты могут быть подвижными (*MovableGameObject*) и неподвижными (*NotMovableGameObject*). К подвижным объектам относятся игроки и снаряды, к неподвижным – стены. Для предоставления движения будет определён интерфейс *IMovable*, который реализуется в классе *MovableGameObject*. От этого класса наследуются игрок и пуля.

Схематичная иерархия классов приложения представлена на рисунке 2.1.

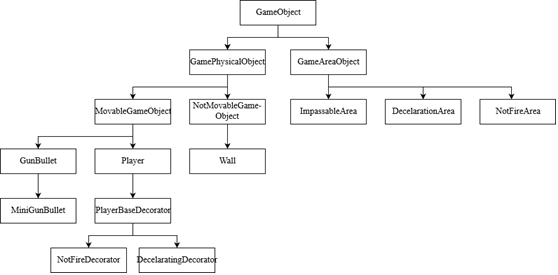


Рисунок 2.1 – Схематичная иерархия классов

Для генерации игровых объектов будет использован паттерн «фабричный метод». С помощью данного паттерна будут сгенерированы стены, игроки, сервисы. Паттерн «фабричный метод» позволяет обойтись без дублирования кода создания зависимых сущностей [7, c. 145].

Ситуации, в которых надо применять паттерн «фабричный метод» применять:

– когда заранее неизвестно, объекты каких типов необходимо создавать;

– когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать;

– когда создание новых объектов необходимо делегировать из базового класса классам наследникам.

На рисунке 2.2 приведена схема паттерна «фабричный метод».

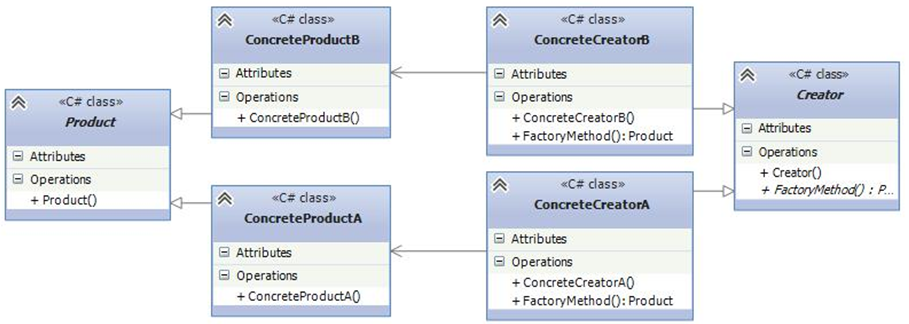


Рисунок 2.2 – Схема паттерна «фабричный метод»

Для реализации изменения характеристик игроков использован паттерн «декоратор». Декоратор позволяет динамически подключать к объекту дополнительную функциональность за счёт переопределения функционала декорируемого класса (рисунок 2.3).

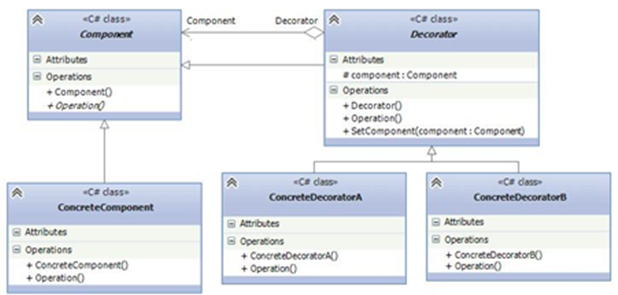


Рисунок 2.3 – Схема паттерна «декоратор»

Иерархия разработанных классов для игрового приложения приведена на рисунке 2.4.

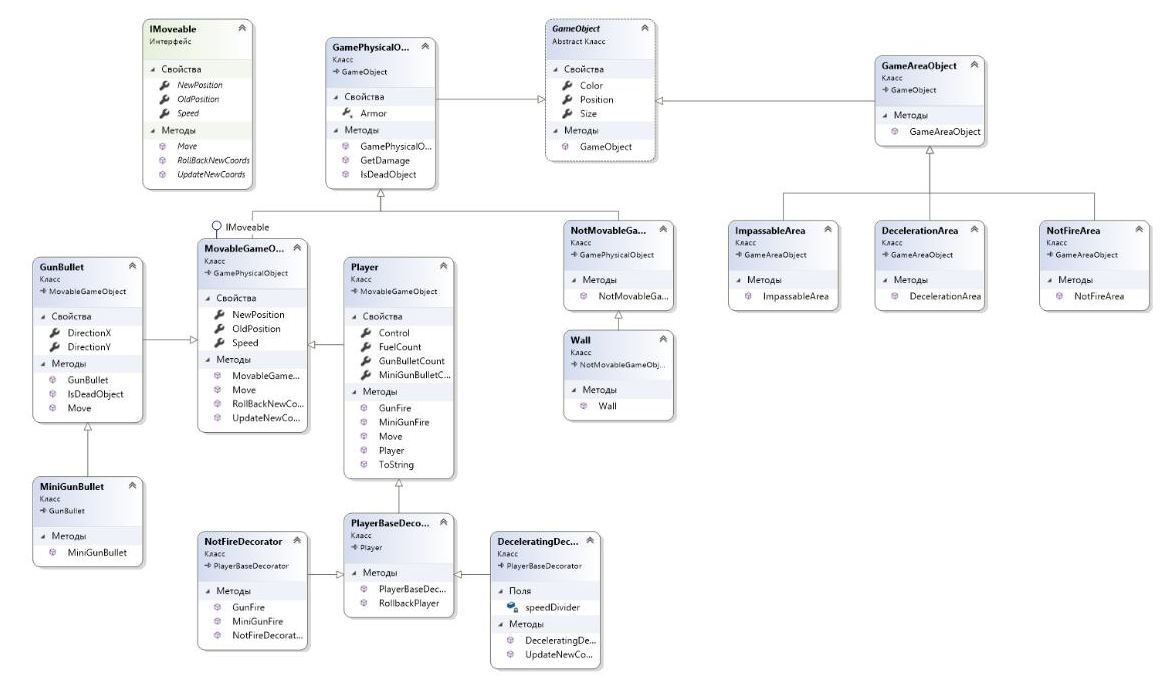


Рисунок 2.4 – Иерархия разработанных классов

При запуске данной игры пользователь попадает на игровое поле игры, которая представлена в окне *Windows Form*. Вид окна игрового приложения представлен на рисунке 2.5. На игровом поле находятся следующие игровые объекты:

* стена, которую можно разрушить выстрелом (рисунок 2.5, пункт 1);
* первый игрок, который может перемещаться с помощью клавиш и стрелять (рисунок 2.5, пункт 2);
* второй игрок, который может перемещаться с помощью клавиш и стрелять (рисунок 2.5, пункт 3);
* территория, на которой нельзя стрелять (рисунок 2.5, пункт 4);
* территория, через которую нельзя пройти (рисунок 2.5, пункт 5);
* территория, на которой замедляется скорость игрока (рисунок 2.5, пункт 6).

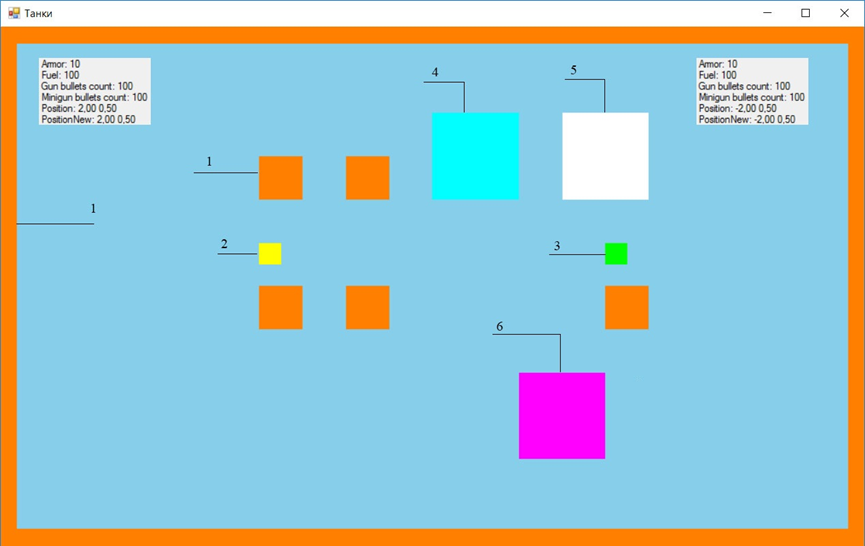


Рисунок 2.5 – Игровое поле

1 – стена, которую можно разрушить с помощью выстрела; 2 – первый игрок; 3 – второй игрок; 4 – территория, на которой нельзя стрелять; 5 – территория, через которую нельзя пройти; 6 – территория, на которой замедляется скорость игрока

В начале игры каждый игрок имеет одинаковые характеристики: броню, топливо, снаряды, пули. Характеристика первого игрока представлена на рисунке 2.6. Её отображение обеспечивает элемент управления *Label* (*Windows Forms*).

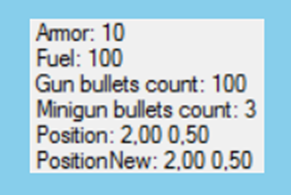


Рисунок 2.6 – Характеристика первого игрока

Когда игрок уничтожает соперника, он выигрывает. На рисунке 2.7 представлено сообщение о победе для первого игрока.

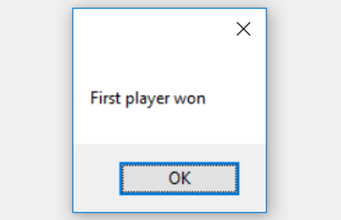


Рисунок 2.7 – Сообщение о победе первого игрока

Игроки могут уклоняться от выстрелов снарядов, так как их скорость ограничена. Управление выстрелами для первого игрока происходит клавишей *Space*, для второго игрока – клавишей *Numpad0*. Пули из пулемёта достигают соперника моментально.

Управление пулемётом для первого игрока происходит клавишей *С*, для второго игрока – клавишей *Numpad1*.

Когда заканчивается топливо, игрок больше не может перемещаться. Победителем считается игрок, уничтоживший танк соперника.

В таблице 2.1 приведено соответствие клавиш управления для двух игроков.

Таблица 2.1 – Функциональное описание и управление

|  |  |
| --- | --- |
| Клавиша | Результат |
| 1 | 2 |
| *W* | Движение вверх первого игрока |
| *S* | Движение вниз первого игрока |
| *A* | Движение влево первого игрока |
| *D* | Движение вправо первого игрока |
| *С* | Выстрел мгновенным снарядом первого игрока |
| *Space* | Выстрел снарядом первого игрока |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Клавиша | Результат |
| 1 | 2 |
| *NumpadUp* | Движение вверх второго игрока |
| *NumpadDown* | Движение вниз второго игрока |
| *NumpadLeft* | Движение влево второго игрока |
| *NumpadRight* | Движение вправо второго игрока |
| *Numpad0* | Выстрел снарядом второго игрока |
| *Numpad1* | Выстрел мгновенным снарядом второго игрока |

Из таблицы 2.1 следует, что каждый игрок может управлять только назначенными клавишами.

## **2.2 Программная реализация игрового приложения**

Для взаимодействия игровых объектов были реализованы и созданы классы, листинги которых приведены в приложении А.

Основные классы приложения:

1. *GameObject* – абстрактный класс для генерации игровых объектов, который содержит свойства позиции, цвета и размера игрового объекта, конструктор с параметрами цвета, позиции и размера;
2. *GamePhysicalObject* – класс для создания физических игровых объектов, который содержит свойство брони, конструктор с параметрами цвета, позиции и размера, методы *GetDamage()* для получения урона и *IsDeadObject()* для определения уничтожен ли объект;
3. *GameAreaObject* – класс для создания территорий, содержащий конструктор с параметрами позиции и цвета;
4. *MovableGameObject –* класс для генерации подвижных объектов, который содержит свойства старой позиции, новой позиции, скорости игрового объекта, конструктор с параметрами цвета, позиции, брони, скорости и размера, методы *Move()* для перемещения, *RollBackCoords()* для возвращения координат, *UpdateNewCoors()* для обновления координат;
5. *NotMovableGameObject* – класс для генерации неподвижных объектов, который содержит конструктор с параметрами цвета, позиции, размера и брони;
6. *ImpassableArea –* класс для создания территорий, через которые нельзя пройти, содержит конструктор с параметром позиции;
7. *DecelarationArea –* класс для создания территорий, на которых происходит замедление игрока, содержит конструктор с параметром позиции;
8. *NotFireArea –* класс для создания территорий, на которых нельзя производить выстрелы, содержит конструктор с параметром позиции;
9. *Player –* класс для создания игрока, который содержит свойства контроллера, подсчета топлива, количества снарядов и пуль, конструктор с параметрами цвета, позиции, контроллера, методы *MiniGunFire()* для снарядов, *GunFire()* для пуль, *Move()* для перемещения, *ToString()* для получения строкового представления данных;
10. *Wall –* класс для создания стен, содержащий конструктор с параметрами позиции, размера, брони;
11. *GunBullet –* класс для создания снарядов, который содержит свойства направления по *x* и *y*, методы *IsDeadObject()* для определения уничтожен ли объект, *Move()* для перемещения;
12. *MiniGunBullet –* класс для создания пуль из пулемёта, которые достигают цели моментально;*PlayerBaseDecorator –* класс, служащий для декорации класса игрока и заменены его функционала и содержащий метод *RollbackPlayer()* для возвращения игрока;
13. *NotFireDecorator –* класс, служащий для декорации класса игрока и заменены его функционалаи содержащий переопределённые методы *MiniGunFire()* для снарядов, *GunFire()* для пуль*;*
14. *DecelaratingDecorator –* класс, служащий для декорации класса игрока и заменены его функционала, содержащий поле коэффициента скорости и переопределённый метод *UpdateNewCoords()* для обновления координат*;*
15. *CollisionService* – класс, реализующий сервис коллизийи содержащий поле игрового хранилища и перегруженные методы *IsCollisionPossible()* для определения возможности коллизии, *ProcessCollisin()* для создания коллизии, *GetNearestObject()* для получения ближайшего объекта*;*
16. *DrawingService* – класс, содержащий метод *DrawObject()* для отрисовки игровых объектов*;*
17. *GameActionService* – класс, реализующий сервис игровых действийи содержащий поля игрового хранилища, сервисов действий игрока и коллизий и методы *ProcessPlayerAction()* для создания действий игрока, *ProcessBulletCollision()* для создания коллизии пуль, *ProcessBulletMoving()* для создания движения пуль, *IsFirstPlayerWon()* для проверки выигрыша первого игрока*, IsSecondPlayerWon()* для проверки выигрыша второго игрока*, DisposeDeadObject()* для удаления мертвых объектов*;*
18. *PlayerActionService –* класс, реализующий сервис действий игрокаи содержащий методы *ProcessPlayerMoving()* для создания движения игрока, *IsPlayerMoved()* для проверки движения игрока*, IsPlayerFired()* для проверки выстрелов игрока*, IsPlayerFiredByMiniGun()*для проверки снарядов игрока*;*
19. *ControlSettings –* класс, содержащий свойства настройки соответствия клавиш и конструктор с параметрами *Up*, *Down*, *Left*, *Right*, *Fire*, *FireMiniGun*;
20. *Helper* – класс, содержащий метод *GetObjectDirections()*, который возвращает направление объекта;
21. *DLLImporter –* класс, испортирующий методы сервера из динамической библиотеки;
22. *GameConfig* – конфигурационный класс для настройки глобальных значений.

Интерфейс *IMoveable* содержит свойства скорости, старой и новой позиции, методы *UpdateNewCoords()* для обновления координат*, Move() –* для передвижения, *RollBackNewCoords()* для возвращения новых координат. Данный интерфейс наследует класс *MovableGameObject* иреализуетметоды *Move()*, *RollBackNewCoords()*, *UpdateNewCoords().*

Для генерации игровых объектов использован паттерн «фабричный метод». С помощью данного паттерна будут сгенерированы стены, игроки, сервисы. Шаблон проектирования «фабричный метод» позволяет не дублировать код создания зависимых сущностей.

Паттерн «фабричный метод» используется для реализации классов *ComponentFactory, GameObjectFactory, ServiceFactory.*

Класс *ComponentFactory* генерирует игровое поле и управление клавиатурой.

Класс *GameObjectFactory* генерирует игровые объекты: игроки, стены, снаряды, пули, территории.

Класс *ServiceFactory* генерирует сервисы коллизий, рисования, действий игрока, действий игры.

Для изменения характеристик игроков использован паттерн «декоратор». Декоратор позволяет динамически подключать к объекту дополнительную функциональность. Они предоставляет наследованию более гибкую альтернативу, поскольку позволяют динамически в процессе выполнения определять новые возможности у объектов [8, c. 111].

Схема декоратора представлена на рисунке 2.5.

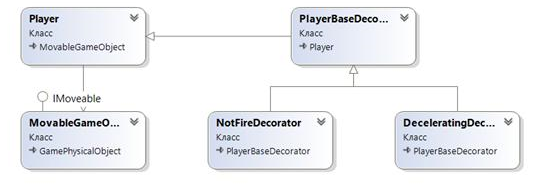


Рисунок 2.5 – Схема декоратора для изменения характеристик игрока

Для организации сетевого взаимодействия между игроками был разработан проект сервера на *C*++ с использованием *Windows* *Socket.*

Проект сервера представляет собой библиотеку динамической компоновки (*Dynamic*-*Link* *Library*)*.* Данная библиотека в дальнейшем используется в проекте самой игры, который представляет собой клиентской приложение для разработанного сервера.

Серверсодержитфункцию *createAndListenSocket* (*\* char serverAddress*, *u\_short port*), которая принимает адрес сервера и порт. Основная задача этой функции – создание и прослушивание сокета. Также на сервере реализована функция *sendMessageToSocket* (*int* *key*, *u*\_*short* *port*), принимающая клавишу и порт. Эта функция осуществляет отправку сообщений сокету.

Схема работы сервера, работающего по протоколу *TCP* с использованием сокетов, приведена на рисунке 2.6.

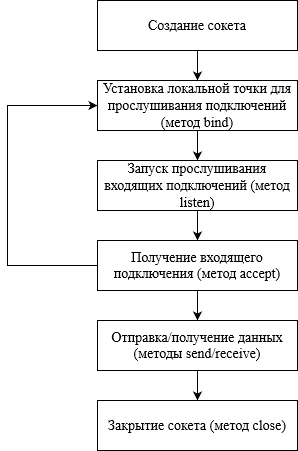


Рисунок 2.6 – Схема работы *TCP* сервера с использованием сокетов

В проекте игры реализован вспомогательный статический класс *DLLImporter*, в котором импортируются функции *createAndListenSocket* (*string serverAddress, ushort port*)и *sendMessageToSocket (int key, ushort port).* Для этого к ним был добавлен атрибут *DllImport*, который указывает, что это метод передаётся библиотекой динамических компонентов (*DLL*) как статическая точка входа.

Атрибут *DllImport* в качестве параметров принимает имя библиотеки и *CallingConvention*,равный *Cdec*,который назначает соглашение о вызовах, которое требуется для вызова методов, реализованных ы неуправляемом коде. Вызывающий объект этого параметра очищает стек, что позволяет работать с методами с переменным числом параметров.

Класс *DLLImporter* используется в методе *RunCppServer() –* длязапускасервераив *SendMessageToCpp* (*Keys key*), который принимает в качестве входного параметра клавишу *key* и используется для передачи положения другому игроку*.*

В данной главе были определены основные этапы проектирования приложения, рассмотрены основные игровые объекты, составлена иерархия классов игры «Танки», описаны основные классы и методы программы, изучены паттерны проектирования «фабричный метод» и «декоратор», рассмотрена организация сетевого взаимодействия игры «Танки».

# **3 ВЕРИФИКАЦИЯ И АНАЛИЗ РЕЗУЛЬТАТОВ**

## **3.1 Эксплуатация сетевого игрового приложения «Танки»**

Разработанная *2D* игра *«*Танки*»* рассчитана на двух игроков. Каждый из которых управляет своим танком с помощью клавиатуры. В начале игры танки имеют одинаковые характеристики: скорость движения, броню, вооружение (пушка и пулемёт), запас снарядов и топливо. Танки перемещаются по игровому полю с неоднородной территорией: участки с замедлением скорости; участки, на которых невозможно производить выстрелы; непреодолимые участки, но через которые можно производить выстрелы; участки, через которые нельзя проехать и нельзя стрелять. Пули из пулемёта достигают цели моментально, у снарядов скорость ограниченна, так что танки могут уклоняться от выстрелов. Победителем считается игрок, уничтоживший танк соперника.

Так как данная игра является сетевой, при подключении только одного игрока приложение не будет корректно работать.

Если сначала подключился первый игрок, выполнит действие, и следом подключится второй игрок, у последнего будет не актуальная информация о состоянии первого игрока

При запуске данной игры пользователь попадает на главную сцену игры, которая представлена в окне *Windows Form* (рисунок 3.1). На игровом поле изображены два игрока и несколько видов препятствий: стены, которые можно разрушить с помощью выстрелов, и территории, которые меняют характеристики игроков при их прохождении. Территория, на которой нельзя производить выстрелы обозначена голубым цветом, непроходимая территория – белым, территория, замедляющая игрока – розовым.

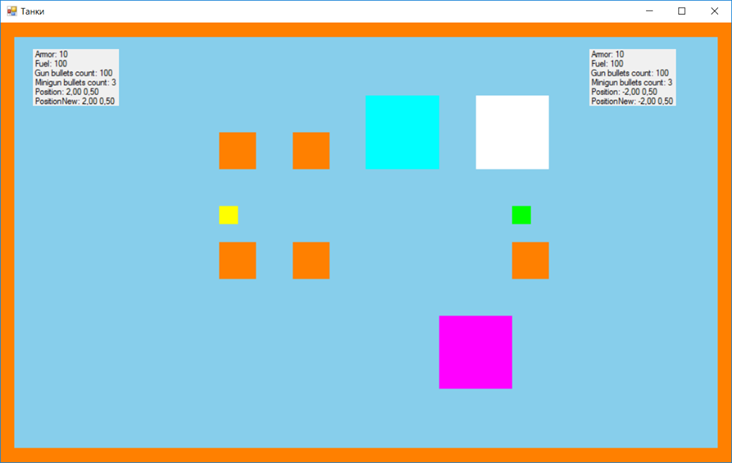


Рисунок 3.1 – Игровое поле приложения «Танки»

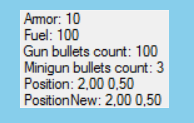
В начале игры каждый игрок имеет одинаковые характеристики. Начальное значение брони – 10, топлива – 100, пуль – 100, мгновенных снарядов – три (рисунок 3.2).

Рисунок 3.2 – Характеристика первого игрока

Для произведения выстрела первый игрок в начале нажимает клавишу *D*, чтобы повернуть влево, затем клавишу *Space* для выстрела пулей (рисунок 3.3). Второй игрок может уклониться от этой пули. Количество топлива и пуль первого игрока уменьшилось на одну единицу. Когда топливо заканчивается, танк больше не может двигаться.

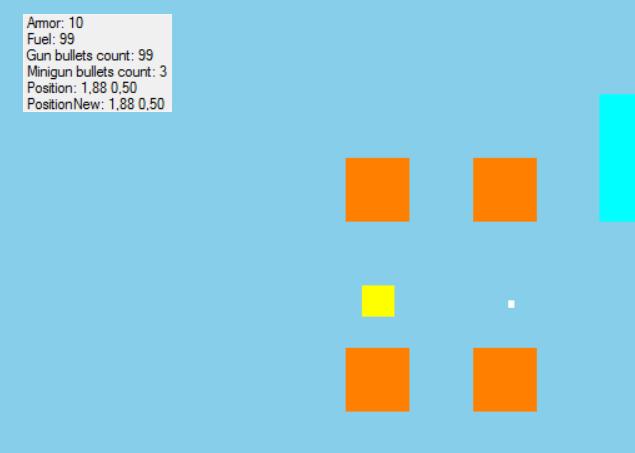


Рисунок 3.3 – Выстрел первого игрока

Что касается второго игрока, его броня уменьшилась на одну единицу, так как он не уклонился (рисунок 3.4).

Статистика о топливе, броне, количестве пуль и положении, а также их ихменения отображаются в верхней части экрана. Справа вверху – для первого игрока, слева вверху – для второго игрока.

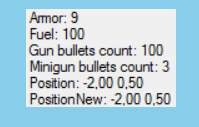


Рисунок 3.4 – Характеристика второго игрока

Когда первый игрок стреляет с помощью клавиши *С*, второй не может уклониться, поэтому получает урон (рисунок 3.5). Начальное количество мгновенных снарядов у каждого игрока равно трём.

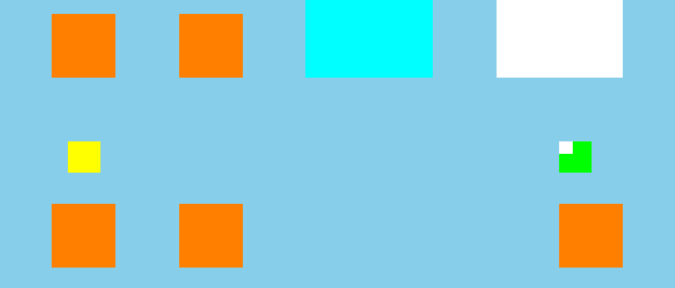


Рисунок 3.5 – Выстрел первого игрока мгновенным снарядом

Игроки могут уничтожать стены. Значение здоровья стены равно пяти, поэтому стена исчезает после попадания пяти пуль (рисунок 3.6).

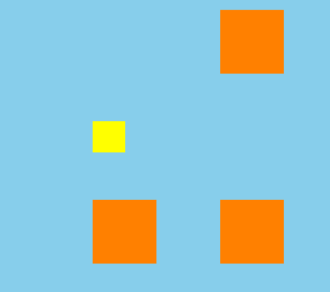


Рисунок 3.6 – Уничтожение стены

При выстреле пуля касается стены, так как происходит коллизия этих объектов. (рисунок 3.7).

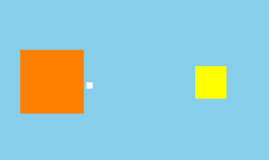


Рисунок 3.7 – Коллизия пули и стены

В игре так же реализована коллизия игроков при наезде танков. Танки игроков сталкиваются и не могут двигаться дальше. (рисунок 3.8).

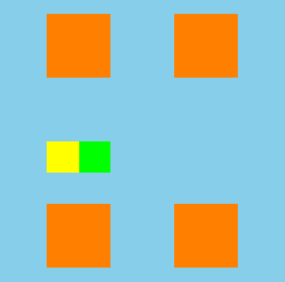


Рисунок 3.8 – Коллизия игроков

Аналогичные действия может производить второй игрок. Для выстрелов нужно использовать клавишу *Numpad0*, а для мгновенных выстрелов клавишу *Numpad1*.

Если у одного из игроков заканчивается топливо (*fuel*), то фактически он уже считается проигравшим.

В игре побеждает танк, который первый уничтожает танк соперника.

У проигравшего игрока значение брони (*armor*) становится равно нулю, и игра генерирует сообщение о победе игрока (рисунок 3.9).

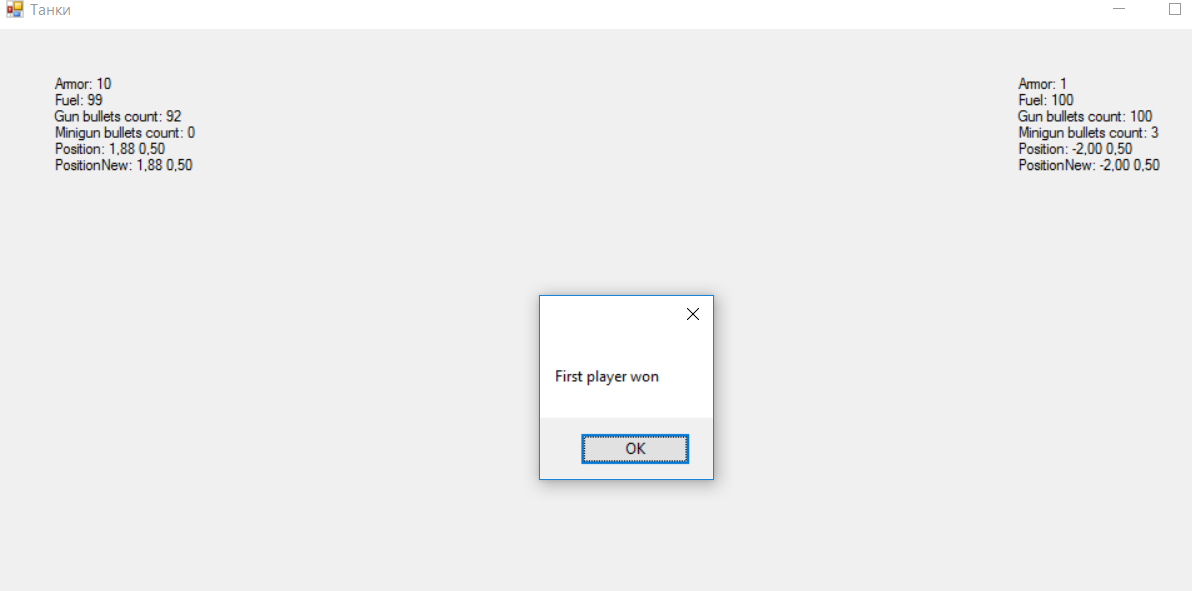


Рисунок 3.9 – Завершение игры

После заверения игры, сообщение о победе одного из игроков генерируется у каждого игрока одновременно. Для повторной игры каждому из игроков требуется к ней переподключиться.

## **3.2 Верификация работы приложения «Танки»**

Для взаимодействия игровых объектов были реализованы и созданы классы, листинги которых приведены в приложении А.

Для тестирования правильности работы приложения, были выделены слабые места приложения, к ним подобраны всевозможные кейсы и протестированы вручную.

Список возможных кейсов:

1. движение танка;
   1. движение должно быть возможно по всей свободной территории;
   2. движение должно быть возможно по всем зонам, кроме зоны с невозможностью движения;
   3. движение не должно быть возможно в координаты, на которых находится стена;
   4. движение не должно быть возможно в координаты, на которых находится танк противника;
   5. после движения танка, уровень топлива должен уменьшиться.
2. движение пули из пушки;
   1. пуля не должна проходить насквозь танка;
   2. пуля не должна проходить насквозь стены;
   3. пуля не должна уничтожать выпустивший её танк при создании;
   4. пуля не должна проскакивать танк;
   5. пуля не должна проскакивать стену.
3. движение пули из пулемёта;
   1. пуля должна появиться только на объекте, с которым столкнулась
4. попадание пули в объект;
   1. у пули должен спуститься уровень брони до 0;
   2. пуля должна быть уничтожена;
   3. у объекта, в который попала пуля, должен снизиться уровень брони;
   4. если у объекта, в который попала пуля, уровень брони равен нулю, он должен быть уничтожен;
   5. объект не должен быть уничтожен, если уровень его брони остался больше нуля;
   6. у объекта, в который попала пуля, броня не должна опуститься ниже нуля;
   7. после потери всей брони танк должен быть уничтожен, а его противник назначен победителем.
5. потеря всего топлива;
   1. танк должен утратить возможность к движению;
   2. танк должен сохранить способность к стрельбе;
   3. танк должен иметь возможность поворачивать башню.

е) потеря всех снарядов;

1. танк должен утратить возможность к стрельбе из того оружия, к которому закончились патроны, но сохранить возможность стрелять из другого.

ж) движения по зонам

1. если танк находится на замедляющей территории, он должен быть способен двигаться, но с меньшей скоростью
2. если танк покидает замедляющую территорию, его скорость должна восстановиться
3. если танк движется по территории с ограничением стрельбы, он не должен стрелять ни из одного из орудий
4. если танк покидает территорию с ограничением стрельбы, его способность к стрельбе должна быть восстановлена.

Следуя выделенным кейсам, приложение было многократно протестировано, на разных этапах разработки были найдены и устранены дефекты. Новые дефекты не были найдены. Игра работает без сбоев. На выходе получено работающее и отлаженное приложение.

## **3.3 Модульное тестирование**

Для тестирования работы приложения были созданы модульные тесты. Были протестированы следующие методы *DeceleratingDecorator()*, *NotFireDecorator()*, *CreateControlSettings()*, *CreateGameRepository()*, *CreateWall()*, *CreatePlayer()*, *CreateDeceleratingDecorator()*, *CreateInpassableArea()*, *CreateNotFireArea()*, *CreateDecelerationArea()*, *CreateGunBullet()*, *GetObjectDirections()*, *IsDeadObject()*, *RollBackNewCoords()*, *Move()*.

На рисунке 3.10 приведены результаты выполнения модульных тестов.

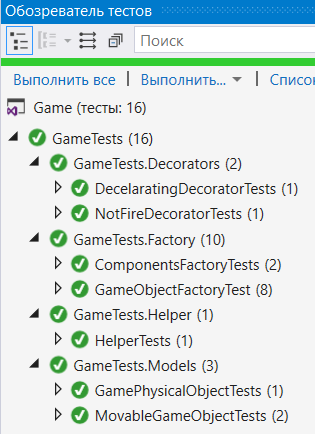


Рисунок 3.10 – Результаты выполнения модульных тестов

Результаты данных модульных текстов свидетельствуют и правильности работы конкретных методов. В ходе разработки приложения все дефекты были исправлены.

В данной главе тестировались методы и игровой процесс приложения. На основе показанных выше сведений, можно сделать вывод, что приложение работает корректно.

**ЗАКЛЮЧЕНИЕ**

При выполнении задания первоначально был проведен аналитический обзор средств решения задачи в области разработки игровых приложений. Были изучены характеристики *OpenGL* и C#. Также было выявлено, что успешная игра жанра «Танки» должна быть реализована с помощью библиотеки *OpenGL* и *Windows* *Form*.

В качестве средства технической реализации приложения была выбрана среда разработки *Visual* *Studio* 2019.

При решении поставленной задачи было разработано игровое приложение, с использованием средства языка программирования *C*# *Windows* *Form* и графики *OpenGl*, позволяющее двум пользователем при помощи клавиатуры управлять двумя игроками для перемещения и осуществления выстрелов. Сетевые коммуникации между игроками организованы с помощью *winsock*.

В соответствии с заданием была разработана 2*D*-игра жанра «Танки». В качестве технического средства реализации приложения был выбран язык программирования *C*#. Игра имеет одно окно *Windows* *Form*, на котором друг напротив друга отображаются два игрока и несколько разновидностей препятствий. Также на нём идёт подсчёт характеристик и статусов игроков. Игроки в начале игры имеют одинаковые характеристики: броня, снаряды и боеприпасы. При столкновении игроков друг с другом, а также с препятствиями, использованы основные принципы коллизии. В игре реализована анимация и графика с использованием *OpenGl*. Также реализованы шаблоны проектирования: «фабричный метод» – для реализации генерации игровых объектов и «декоратор» – для изменения характеристик игроков.

Данная игра рассчитана на широкий круг пользователей, как на взрослых, так и на детей. Её достоинствами являются простота в управлении и низких системных требованиях. Игра способствует развитию реакции у детей, учит рассуждать и логически мыслить, развивает образное и пространственное мышление. Игра позволяет учиться быстрее реагировать на неожиданные повороты событий, что поможет в учёбе и в жизни. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

При выполнении курсового проекта были изучены различные свойства реализации, создания и проектирования компьютерных игр с использованием языка программирования C#. Также изучены основные свойства визуальных компонентов выбранной среды реализации для графического вывода компьютерной игры. Можно сделать выводы о том, что данный курсовой проект был выполнен в полном соответствии поставленным задачам.

**Список использованных источников**

1. Программирование с использованием OpenGL [Электронный ресурс] – Режим доступа: http://www.opengl.org.ru/ – Дата доступа: 25.09.2020.
2. Верма, Р. Д. Введение в OpenGL / Р. Д. Верма. – 2-е изд. – М. : Горячая линия – Телеком, 2017 – 304 с.
3. DirectX graphics and gaming [Электронный ресурс] – Режим доступа: https://docs.microsoft.com/en-us/windows/win32/directx/ – Дата доступа: 30.09.2020.
4. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов / Д. Вольф. – М. : ДМК Пресс, 2015. – 368 с.
5. Сравнение OpenGL и Direct3D [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/79257/ – Дата доступа: 30.09.2020.
6. Шмидт, Д. Программирование сетевых приложений на С++. Том 2. Систематическое повторное использование: ACE и каркасы / Д. Шмидт, С, Хьюстон. – М. : Бином, 2010. – 400с
7. Тепляков С. А. Паттерны проектирования на платформе .NET / С. А. Тепляков. – СПб. : Питер, 2015. – 320 с.
8. Head First. Паттерны проектирования / Э. Фримен [и др.]. – СПб. : Питер, 2019. – 656 с.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

Класс *ComponentsFactory*:

using Game.Models.ModelComponents;

using System.Windows.Forms;

namespace Game.Factory

{

/// <summary>

/// Класс генерирует игровое поле и управление клавиатурой

/// </summary>

public static class ComponentsFactory

{

// игровое поле

private static readonly GameRepository gameRepository;

static ComponentsFactory()

{

gameRepository = new GameRepository();

}

// настройки клавиатуры

public static ControlSettings CreateControlSettings(Keys up, Keys down, Keys left, Keys right, Keys fire, Keys fireMiniGun)

{

return new ControlSettings(up, down, left, right, fire, fireMiniGun);

}

// создание игрового поля

public static GameRepository CreateGameRepository()

{

return gameRepository ?? new GameRepository();

}

}

}

Класс *GameObjectFactory*:

using Game.Models;

using Game.Models.Areas;

using Game.Models.Decorators;

using Game.Models.ModelComponents;

using System.Numerics;

namespace Game.Factory

{

/// <summary>

/// Класс генерирует игровые объекты: игроки, стены, снаряды, пули, территории.

/// </summary>

public static class GameObjectFactory

{

//созадние игрока

public static Player CreatePlayer(Vector4 color, Vector2 position, ControlSettings controlSettings)

{

return new Player(color, position, controlSettings);

}

// создние стены

public static Wall CreateWall(Vector2 position, float size, int armor = Constants.WallHealth)

{

return new Wall(position, size, armor);

}

// созадние пуль

public static GunBullet CreateGunBullet(Player player)

{

var (directionX, directionY) = Helper.Helper.GetObjectDirections(player);

return new GunBullet(

new Vector2(

player.Position.X + player.Size \* directionX - (directionX == 0 ? player.Size / 2 : 0),

player.Position.Y + player.Size \* directionY - (directionY == 0 ? player.Size / 2 : 0)),

directionX,

directionY);

}

// созадние мгновенных снарядов

public static MiniGunBullet CreateMiniGunBullet(Player player)

{

var collisionService = ServiceFactory.CreateCollisionService();

var (directionX, directionY) = Helper.Helper.GetObjectDirections(player);

var nearestObject = collisionService.GetNearestObjectInPath(player);

if (nearestObject == null)

{

return null;

}

return new MiniGunBullet(

new Vector2(

nearestObject.Position.X,

nearestObject.Position.Y),

directionX,

directionY);

}

// создание территорий

public static DecelerationArea CreateDecelerationArea(Vector2 position)

{

return new DecelerationArea(position);

}

public static ImpassableArea CreateInpassableArea(Vector2 position)

{

return new ImpassableArea(position);

}

public static NotFireArea CreateNotFireArea(Vector2 position)

{

return new NotFireArea(position);

}

// декоратор, замедляющий игрока

public static DeceleratingDecorator CreateDeceleratingDecorator(Player player)

{

return new DeceleratingDecorator(player);

}

// декоратор, запрещающий стрелять игроку

public static NotFireDecorator CreateNotFireArea(Player player)

{

return new NotFireDecorator(player);

}

}

}

Класс *ServiceFactory*:

using Game.Services;

namespace Game.Factory

{

/// <summary>

/// Класс генерирует сервисы коллизий, рисования, действий игрока, действий игры

/// </summary>

public static class ServiceFactory

{

// сервис коллизий

public static CollisionService CreateCollisionService()

{

return new CollisionService(ComponentsFactory.CreateGameRepository());

}

// сервис отрисовки

public static DrawingService CreateDrawingService()

{

return new DrawingService();

}

// сервис действий игрока

public static PlayerActionService CreatePlayerActionService()

{

return new PlayerActionService();

}

// сервис действий игры

public static GameActionService CreateGameActionService()

{

return new GameActionService(

CreatePlayerActionService(),

ComponentsFactory.CreateGameRepository(),

CreateCollisionService());

}

}

}

Класс *DLLImporter*:

using System;

using System.Runtime.InteropServices;

namespace Game.Helper

{

public static class DLLImporter

{

[DllImport("WSDll.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern IntPtr createAndListenSocket([MarshalAs(UnmanagedType.LPStr)] string serverAddress, ushort port);

[DllImport("WSDll.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int sendMessageToSocket(int key, ushort port);

}

}

Класс *Helper*:

using Game.Models.Base;

using System;

namespace Game.Helper

{

/// <summary>

/// Класс для определения направления объекта

/// </summary>

public static class Helper

{

public static (int, int) GetObjectDirections(MovableGameObject movableObject)

{

// Если двигались вправо, направление 1

// Если двигались влево, направление -1

var directionX = movableObject.Position.X - movableObject.OldPosition.X;

if (directionX != 0)

{

directionX /= Math.Abs(directionX);

}

// Если двигались вверх, направление 1

// Если двигались вниз, направление -1

var directionY = movableObject.Position.Y - movableObject.OldPosition.Y;

if (directionY != 0)

{

directionY /= Math.Abs(directionY);

}

// Если изначально стояли на месте, стреляем вправо

if (directionX == 0 && directionY == 0)

{

directionX = 1;

}

return ((int)directionX, (int)directionY);

}

}

}

Интерфейс *IMoveable*:

using System.Numerics;

namespace Game.Interfaces

{

/// <summary>

/// Интерфейс, содержащий методы поведения движущихся объектов

/// </summary>

public interface IMoveable

{

float Speed { get; set; }

Vector2 OldPosition { get; set; }

Vector2 NewPosition { get; set; }

void UpdateNewCoords(float x, float y);

void Move();

void RollBackNewCoords();

}

}

Класс *DecelerationArea*:

using Game.Models.Base;

using System.Numerics;

namespace Game.Models.Areas

{

/// <summary>

/// Класс территории, на которой происходит замедление

/// </summary>

public class DecelerationArea : GameAreaObject

{

public DecelerationArea(Vector2 position)

: base(position, new Vector4(1, 0, 1, 1))

{

}

}

}

Класс *ImpassableArea*:

using Game.Models.Base;

using System.Numerics;

namespace Game.Models.Areas

{

/// <summary>

/// Класс для создания территорий, через которые нельзя пройти

/// </summary>

public class ImpassableArea : GameAreaObject

{

public ImpassableArea(Vector2 position)

: base(position, new Vector4(1, 1, 1, 1))

{

}

}

}

Класс *NotFireArea*:

using Game.Models.Base;

using System.Numerics;

namespace Game.Models.Areas

{

/// <summary>

/// Класс территории, на которой невозможны выстрелы

/// </summary>

public class NotFireArea : GameAreaObject

{

public NotFireArea(Vector2 position)

: base(position, new Vector4(0, 1, 1, 1))

{

}

}

}

Класс *GameAreaObject*:

using System.Numerics;

namespace Game.Models.Base

{

/// <summary>

/// класс для создания территорий, содержащий кнострутор с параметрами позиции и цвета

/// </summary>

public class GameAreaObject : GameObject

{

public GameAreaObject(Vector2 position, Vector4 color)

: base(position, color, Constants.AreaSize)

{

}

}

}

Класс *GameObject*:

using System.Numerics;

namespace Game.Models.Base

{

/// <summary>

/// Абстрактный класс для генерации игровых объектов, который содержит свойства позиции,

/// цвета и размера игрового объекта, конструктор с параметрами цвета, позиции и размера

/// </summary>

public abstract class GameObject

{

public float Size { get; set; }

public Vector2 Position { get; set; }

public Vector4 Color { get; set; }

public GameObject(Vector2 position, Vector4 color, float size)

{

Position = position;

Color = color;

Size = size;

}

}

}

Класс *GamePhysicalObject*:

using System.Numerics;

namespace Game.Models.Base

{

/// <summary>

/// Класс для создания физических игровых объектов, который содержит свойство брони, конструктор с параметрами цвета, позиции и размера

/// </summary>

public class GamePhysicalObject : GameObject

{

protected float Armor { get; set; }

public GamePhysicalObject(Vector2 position, Vector4 color, float size, int armor)

: base(position, color, size)

{

Armor = armor;

}

// для получения урона

public void GetDamage(float damage = 1)

{

Armor -= damage;

}

// для определения уничтожен ли объект

public virtual bool IsDeadObject()

{

return Armor <= 0;

}

}

}

Класс *MovableGameObject*:

using Game.Interfaces;

using System.Numerics;

namespace Game.Models.Base

{

/// <summary>

/// Класс для генерации подвижных объектов, который содержит свойства старой позиции, новой позиции, скорости игрового объекта

/// </summary>

public class MovableGameObject : GamePhysicalObject, IMoveable

{

public Vector2 OldPosition { get; set; }

public Vector2 NewPosition { get; set; }

public float Speed { get; set; }

public MovableGameObject(Vector2 position, Vector4 color, float size, int health, float speed)

: base(position, color, size, health)

{

OldPosition = position;

NewPosition = position;

Speed = speed;

}

// движение

public virtual void Move()

{

OldPosition = Position;

Position = NewPosition;

}

// возвращение новых координат

public void RollBackNewCoords()

{

NewPosition = Position;

}

// обновление новых координат

public virtual void UpdateNewCoords(float x, float y)

{

NewPosition = new Vector2(NewPosition.X + x, NewPosition.Y + y);

}

}

}

Класс *NotMovableGameObject*:

using System.Numerics;

namespace Game.Models.Base

{

/// <summary>

/// Класс для генерации неподвижных объектов

/// </summary>

public class NotMovableGameObject : GamePhysicalObject

{

public NotMovableGameObject(Vector2 position, Vector4 color, float size, int armor)

: base(position, color, size, armor)

{

}

}

}

Класс *DeceleratingDecorator*:

namespace Game.Models.Decorators

{

public class DeceleratingDecorator : PlayerBaseDecorator

{

private readonly float speedDivider;

public DeceleratingDecorator(Player player)

: base (player)

{

speedDivider = Constants.SpeedDivider;

}

public override void UpdateNewCoords(float x, float y)

{

base.UpdateNewCoords(x / speedDivider, y / speedDivider);

}

}

}

Класс *NotFireDecorator*:

namespace Game.Models.Decorators

{

public class NotFireDecorator : PlayerBaseDecorator

{

public NotFireDecorator(Player player)

: base(player)

{

}

public override GunBullet GunFire()

{

return null;

}

public override MiniGunBullet MiniGunFire()

{

return null;

}

}

}

Класс *PlayerBaseDecorator*:

namespace Game.Models.Decorators

{

public class PlayerBaseDecorator : Player

{

public PlayerBaseDecorator(Player player)

: base(player.Color, player.Position, player.Control)

{

NewPosition = player.NewPosition;

OldPosition = player.OldPosition;

GunBulletCount = player.GunBulletCount;

MiniGunBulletCount = player.MiniGunBulletCount;

FuelCount = player.FuelCount;

}

public Player RollbackPlayer()

{

return new Player(Color, Position, Control)

{

NewPosition = NewPosition,

OldPosition = OldPosition,

FuelCount = FuelCount,

MiniGunBulletCount = MiniGunBulletCount,

GunBulletCount = GunBulletCount

};

}

}

}

Класс *ControlSettings*:

using System.Windows.Forms;

namespace Game.Models.ModelComponents

{

public class ControlSettings

{

public Keys Up { get; }

public Keys Down { get; }

public Keys Left { get; }

public Keys Right { get; }

public Keys Fire { get; }

public Keys FireMiniGun { get; }

public ControlSettings(Keys up, Keys down, Keys left, Keys right, Keys fire, Keys fireMiniGun)

{

Up = up;

Down = down;

Left = left;

Right = right;

Fire = fire;

FireMiniGun = fireMiniGun;

}

}

}

Класс *GunBullet*:

using Game.Models.Base;

using System.Numerics;

using System;

namespace Game.Models

{

public class GunBullet : MovableGameObject

{

public int DirectionX { get; }

public int DirectionY { get; }

public override bool IsDeadObject()

{

return base.IsDeadObject() ||

Math.Abs(Position.X) >= Constants.MaxMapPosition ||

Math.Abs(Position.Y) >= Constants.MaxMapPosition;

}

public GunBullet(

Vector2 position, int directionX, int directionY,

float size = Constants.GunBulletSize, float speed = Constants.GunBulletSpeed)

: base(position, new Vector4(10, 10, 10, 0), size, Constants.BulletHealth, speed)

{

DirectionX = directionX;

DirectionY = directionY;

}

public override void Move()

{

base.Move();

UpdateNewCoords(Speed \* DirectionX, Speed \* DirectionY);

}

}

}

Класс *MiniGunBullet*:

using System.Numerics;

namespace Game.Models

{

/// <summary>

/// Класс для мгновенного снаряда, конструктор содержит позицию и координаты

/// </summary>

public class MiniGunBullet : GunBullet

{

public MiniGunBullet(Vector2 position, int directionX, int directionY)

: base(position, directionX, directionY, Constants.MiniGunBulletSize)

{

}

}

}

Класс *Player*:

using Game.Factory;

using Game.Models.Base;

using Game.Models.ModelComponents;

using System.Numerics;

namespace Game.Models

{

/// <summary>

/// Класс игрока, конструктор которого содержит цвет, позицию, настройки клавиш

/// </summary>

public class Player: MovableGameObject

{

public ControlSettings Control { get; }

public int FuelCount { get; set; }

public int GunBulletCount { get; set; }

public int MiniGunBulletCount { get; set; }

public Player(Vector4 color, Vector2 position, ControlSettings controlSettings)

: base(position, color, Constants.PlayerSize, Constants.PlayerHealth, Constants.PlayerSpeed)

{

Control = controlSettings;

GunBulletCount = Constants.PlayerGunBulletCount;

MiniGunBulletCount = Constants.PlayerMiniGunBulletCount;

FuelCount = Constants.PlayerFuelCount;

}

// выстрел мнгновенным снарядом

public virtual MiniGunBullet MiniGunFire()

{

if (MiniGunBulletCount > 0)

{

MiniGunBulletCount--;

return GameObjectFactory.CreateMiniGunBullet(this);

}

return null;

}

// выстрел снарядом

public virtual GunBullet GunFire()

{

if (GunBulletCount > 0)

{

GunBulletCount--;

return GameObjectFactory.CreateGunBullet(this);

}

return null;

}

// движение игрока

public override void Move()

{

if (FuelCount > 0)

{

FuelCount--;

base.Move();

}

else

{

OldPosition = new Vector2(

OldPosition.X - (NewPosition.X - Position.X),

OldPosition.Y - (NewPosition.Y - Position.Y));

NewPosition = Position;

}

}

// строковое представление характеристик игрока

public override string ToString()

{

return

$"Armor: {Armor}\n" +

$"Fuel: {FuelCount}\n" +

$"Gun bullets count: {GunBulletCount}\n" +

$"Minigun bullets count: {MiniGunBulletCount}\n" +

$"Position: {Position.X:f2} {Position.Y:f2}\n" +

$"PositionNew: {NewPosition.X:f2} {NewPosition.Y:f2}";

}

}

}

Класс *Wall*:

using Game.Models.Base;

using System.Numerics;

namespace Game.Models

{

/// <summary>

/// Класс стена, конструктор которого содержит позицию, размер, броню

/// </summary>

public class Wall : NotMovableGameObject

{

public Wall(Vector2 position, float size, int armor)

: base(position, new Vector4(1, 0.5f, 0, 1), size, armor)

{

}

}

}

Класс *CollisionService*:

using Game.Models;

using Game.Models.Areas;

using Game.Models.Base;

using Game.Models.Decorators;

using System;

using System.Linq;

using System.Numerics;

namespace Game.Services

{

/// <summary>

/// Класс сервиса коллизий

/// </summary>

public class CollisionService

{

private readonly GameRepository gameRepository;

public CollisionService(GameRepository gameRepository)

{

this.gameRepository = gameRepository;

}

#region Collision cheching

public bool IsCollisionPossible(MovableGameObject movableObject1, MovableGameObject movableObject2)

{

return IsCollisionPossible(

movableObject1.NewPosition, movableObject1.Size,

movableObject2.NewPosition, movableObject2.Size);

}

public bool IsCollisionPossible(MovableGameObject movableObject, GameObject gameObject)

{

return IsCollisionPossible(

movableObject.NewPosition, movableObject.Size,

gameObject.Position, gameObject.Size);

}

public bool IsCollisionPossible(MovableGameObject movableObject, NotMovableGameObject notMovableObject)

{

return IsCollisionPossible(movableObject, (GameObject)notMovableObject);

}

public bool IsCollisionPossible(GunBullet bullet, MovableGameObject movableGameObject)

{

return IsCollisionPossible(bullet, movableGameObject.NewPosition) ||

IsCollisionPossible(bullet, new Vector2(

movableGameObject.NewPosition.X - movableGameObject.Size,

movableGameObject.NewPosition.Y - movableGameObject.Size)) ||

IsCollisionPossible((MovableGameObject)bullet, movableGameObject);

}

public bool IsCollisionPossible(GunBullet bullet, GamePhysicalObject gamePhysicalObject)

{

if (gamePhysicalObject is MovableGameObject)

{

return IsCollisionPossible(bullet, (MovableGameObject)gamePhysicalObject);

}

return IsCollisionPossible(bullet, (NotMovableGameObject)gamePhysicalObject);

}

public bool IsCollisionPossible(GunBullet bullet, NotMovableGameObject notMovableGameObject)

{

return IsCollisionPossible(bullet, notMovableGameObject.Position) ||

IsCollisionPossible(bullet, new Vector2(

notMovableGameObject.Position.X - notMovableGameObject.Size,

notMovableGameObject.Position.Y - notMovableGameObject.Size)) ||

IsCollisionPossible((MovableGameObject)bullet, notMovableGameObject);

}

private bool IsCollisionPossible(GunBullet bullet, Vector2 point)

{

var xPositions = new float[]

{

bullet.Position.X,

bullet.Position.X - bullet.Size,

bullet.NewPosition.X,

bullet.NewPosition.X - bullet.Size

};

var xMax = xPositions.Max();

var xMin = xPositions.Min();

var yPositions = new float[]

{

bullet.Position.Y,

bullet.Position.Y - bullet.Size,

bullet.NewPosition.Y,

bullet.NewPosition.Y - bullet.Size

};

var yMax = yPositions.Max();

var yMin = yPositions.Min();

return point.X <= xMax && point.X >= xMin &&

point.Y <= yMax && point.Y >= yMin;

}

private bool IsCollisionPossible(

Vector2 firstObjectPoint, float firstObjectSize,

Vector2 secondObjectPoint, float secondObjectSize = Constants.MiniGunBulletSize)

{

return firstObjectPoint.X > secondObjectPoint.X - secondObjectSize &&

firstObjectPoint.X - firstObjectSize < secondObjectPoint.X &&

firstObjectPoint.Y > secondObjectPoint.Y - secondObjectSize &&

firstObjectPoint.Y - firstObjectSize < secondObjectPoint.Y;

}

#endregion

public void ProcessCollision(GunBullet bullet, GamePhysicalObject gamePhisicalObject)

{

bullet.GetDamage();

gamePhisicalObject.GetDamage();

}

public Player ProcessCollision(Player player, DecelerationArea \_)

{

return new DeceleratingDecorator(player);

}

public Player ProcessCollision(Player player, NotFireArea \_)

{

return new NotFireDecorator(player);

}

public void ProcessCollision(Player player)

{

player.RollBackNewCoords();

}

public GamePhysicalObject GetNearestObjectInPath(MovableGameObject movableObject)

{

var (directionX, directionY) = Helper.Helper.GetObjectDirections(movableObject);

return GetNearestObjectInPath(movableObject, directionX, directionY);

}

public GamePhysicalObject GetNearestObjectInPath(GunBullet bullet)

{

return GetNearestObjectInPath(bullet, bullet.DirectionX, bullet.DirectionY);

}

private GamePhysicalObject GetNearestObjectInPath(MovableGameObject movableObject, int directionX, int directionY)

{

var gameObjectsInLine = gameRepository.GetAllGamePhysicalObjects(movableObject);

var positionX = movableObject.NewPosition.X;

var positionY = movableObject.NewPosition.Y;

if (directionX == 0)

{

gameObjectsInLine = gameObjectsInLine

.Where(q => q.Position.X >= positionX && positionX >= q.Position.X - q.Size)

.Where(q =>

{

return (directionY > 0

? positionY < q.Position.Y

: positionY > q.Position.Y) ||

IsCollisionPossible(movableObject, q);

})

.OrderBy(q => Math.Abs(positionY - q.Position.Y))

.ToList();

}

else

{

gameObjectsInLine = gameObjectsInLine

.Where(q => q.Position.Y >= positionY && positionY >= q.Position.Y - q.Size).ToList();

gameObjectsInLine = gameObjectsInLine

.Where(q =>

{

return (directionX > 0

? positionX < q.Position.X

: positionX > q.Position.X) ||

IsCollisionPossible(movableObject, q);

})

.OrderBy(q => Math.Abs(positionX - q.Position.X))

.ToList();

}

return gameObjectsInLine.FirstOrDefault();

}

}

}

Класс *DrawingService*:

using OpenTK.Graphics.OpenGL;

using OpenTK;

using Game.Models;

using Game.Models.Base;

namespace Game.Services

{

/// <summary>

/// Класс сервис отрисовки игровых объектов

/// </summary>

public class DrawingService

{

/// <summary>

/// Отрисовка игрового объекта

/// </summary>

/// <param name="gameObject"></param>

public void DrawObject(GameObject gameObject)

{

GL.MatrixMode(MatrixMode.Modelview);

var modelview = Matrix4.LookAt(Vector3.Zero, Vector3.UnitZ, Vector3.UnitY);

GL.LoadMatrix(ref modelview);

const int i = 1;

GL.Begin(PrimitiveType.Quads);

GL.Color4(gameObject.Color.X, gameObject.Color.Y, gameObject.Color.Z, gameObject.Color.W);

GL.Vertex3(gameObject.Position.X, gameObject.Position.Y, 3.0f);

GL.Vertex3(gameObject.Position.X, -gameObject.Size / i + gameObject.Position.Y, 3.0f);

GL.Vertex3(-gameObject.Size / i + gameObject.Position.X, -gameObject.Size / i + gameObject.Position.Y, 3.0f);

GL.Vertex3(-gameObject.Size / i + gameObject.Position.X, gameObject.Position.Y, 3.0f);

GL.End();

}

}

}

Класс *GameActionService*:

using Game.Models;

using Game.Models.Areas;

using Game.Models.Decorators;

using System.Linq;

using System.Windows.Forms;

namespace Game.Services

{

/// <summary>

/// Класс сервиса действий игры

/// </summary>

public class GameActionService

{

private readonly PlayerActionService playerActionService;

private readonly GameRepository gameRepository;

private readonly CollisionService collisionService;

public GameActionService(PlayerActionService playerActionService, GameRepository gameField, CollisionService collisionService)

{

this.playerActionService = playerActionService;

this.gameRepository = gameField;

this.collisionService = collisionService;

}

public void ProcessPlayerAction(Keys key)

{

gameRepository.Player1 = ProcessPlayerAction(gameRepository.Player1, key);

gameRepository.Player2 = ProcessPlayerAction(gameRepository.Player2, key);

}

private Player ProcessPlayerAction(Player player, Keys key)

{

var anotherPlayer = gameRepository.Player1 == player

? gameRepository.Player2

: gameRepository.Player1;

var bullets = gameRepository.Bullets;

var walls = gameRepository.Walls;

var areas = gameRepository.GameAreas;

// Выстрел игрока

if (playerActionService.IsPlayerFired(player, key))

{

var newBullet = player.GunFire();

if (newBullet != null)

{

bullets.Add(newBullet);

}

}

// Выстрел из пулемёта игрока

else if (playerActionService.IsPlayerFiredByMiniGun(player, key))

{

var newBullet = player.MiniGunFire();

if (newBullet != null)

{

bullets.Add(newBullet);

}

}

// Движение игрока

// Проверка на коллизии с другими игроками

// Проверка на коллизии со стенами

// Проверка на коллизию с непроходимой территорией

else if (playerActionService.IsPlayerMoved(player, key))

{

playerActionService.ProcessPlayerMoving(player, key);

// Если нет коллизии - движение

if (!collisionService.IsCollisionPossible(player, anotherPlayer) &&

!walls.Any(q => collisionService.IsCollisionPossible(player, q)) &&

!areas.Any(q => q is ImpassableArea && collisionService.IsCollisionPossible(player, q)))

{

// Коллизия с территориями-декораторами

var collisionArea = areas.FirstOrDefault(q => collisionService.IsCollisionPossible(player, q));

if (collisionArea != null)

{

switch (collisionArea)

{

case DecelerationArea decelerationArea:

player = collisionService.ProcessCollision(player, decelerationArea);

break;

case NotFireArea notFireArea:

player = collisionService.ProcessCollision(player, notFireArea);

break;

}

}

else if (player is PlayerBaseDecorator)

{

player = (player as PlayerBaseDecorator).RollbackPlayer();

}

player.Move();

}

// Если коллизия - обрабатываем

else

{

collisionService.ProcessCollision(player);

}

}

return player;

}

public void ProcessBulletsCollisions()

{

// Проверка всех пуль

foreach (var bullet in gameRepository.Bullets)

{

var nearestObject = collisionService.GetNearestObjectInPath(bullet);

if (!bullet.IsDeadObject() && nearestObject != null && collisionService.IsCollisionPossible(bullet, nearestObject))

{

collisionService.ProcessCollision(bullet, nearestObject);

}

}

}

public void ProcessBulletsMoving()

{

foreach (var bullet in gameRepository.Bullets)

{

if (bullet.IsDeadObject())

{

continue;

}

bullet.Move();

}

}

public bool IsFirstPlayerWon()

{

return gameRepository.Player2.IsDeadObject();

}

public bool IsSecondPlayerWon()

{

return gameRepository.Player1.IsDeadObject();

}

public void DisposeDeadObjects()

{

var deadBullets = gameRepository.Bullets.Where(q => q.IsDeadObject()).ToList();

var deadWalls = gameRepository.Walls.Where(q => q.IsDeadObject()).ToList();

foreach (var deadBullet in deadBullets)

{

gameRepository.Bullets.Remove(deadBullet);

}

foreach (var deadWall in deadWalls)

{

gameRepository.Walls.Remove(deadWall);

}

}

}

}

Класс *PlayerActionService*:

using Game.Models;

using System.Windows.Forms;

namespace Game.Services

{

/// <summary>

/// Класс действий игрока

/// </summary>

public class PlayerActionService

{

/// <summary>

/// Движение игрока

/// </summary>

/// <param name="player">игрок</param>

/// <param name="key">клавиша</param>

public void ProcessPlayerMoving(Player player, Keys key)

{

if (key == player.Control.Up)

{

player.UpdateNewCoords(0, player.Speed);

}

else if (key == player.Control.Down)

{

player.UpdateNewCoords(0, -player.Speed);

}

else if (key == player.Control.Left)

{

player.UpdateNewCoords(+player.Speed, 0);

}

else if (key == player.Control.Right)

{

player.UpdateNewCoords(-player.Speed, 0);

}

}

public bool IsPlayerMoved(Player player, Keys key)

{

return key == player.Control.Up ||

key == player.Control.Down ||

key == player.Control.Left ||

key == player.Control.Right;

}

public bool IsPlayerFired(Player player, Keys key)

{

return player.Control.Fire == key;

}

public bool IsPlayerFiredByMiniGun(Player player, Keys key)

{

return player.Control.FireMiniGun == key;

}

}

}

Класс *Constants*:

namespace Game

{

/// <summary>

/// Класс, хранящий константы

/// </summary>

public static class Constants

{

public const float PlayerSize = 0.25f;

public const float PlayerSpeed = 0.125f;

public const int PlayerHealth = 10;

public const int PlayerGunBulletCount = 100;

public const int PlayerMiniGunBulletCount = 3;

public const int PlayerFuelCount = 100;

public const int BorderWallHealth = int.MaxValue;

public const float BorderWallSize = 0.2f;

public const float BorderWallLeftTopX = 5.0f;

public const float BorderWallLeftTopY = 3.0f;

public const float BorderWallRigftBottomX = -4.8f;

public const float BorderWallRigftBottomY = -2.8f;

public const int WallHealth = 5;

public const float WallSize = 0.5f;

public const float GunBulletSize = 0.05f;

public const float GunBulletSpeed = 0.3f;

public const float MiniGunBulletSize = 0.1f;

public const int BulletHealth = 1;

public const int AreaSize = 1;

public const float SpeedDivider = 2;

public const float MaxMapPosition = 100;

}

}

Класс *GameRepository***:**

using Game.Models;

using Game.Models.Base;

using System.Collections.Generic;

namespace Game

{

public class GameRepository

{

public Player Player1 { get; set; }

public Player Player2 { get; set; }

public List<Wall> Walls { get; set; } = new List<Wall>();

public List<GunBullet> Bullets { get; set; } = new List<GunBullet>();

public List<GameAreaObject> GameAreas { get; set; } = new List<GameAreaObject>();

public List<GamePhysicalObject> GetAllGamePhysicalObjects(GamePhysicalObject excludeObject = null)

{

var result = new List<GamePhysicalObject>() { Player1, Player2 };

result.AddRange(Walls);

if (excludeObject != null)

{

result.Remove(excludeObject);

}

return result;

}

public List<GameObject> GetAllGameObjects(GamePhysicalObject excludeObject = null)

{

var result = new List<GameObject>(GameAreas) { Player1, Player2 };

result.AddRange(Bullets);

result.AddRange(Walls);

if (excludeObject != null)

{

result.Remove(excludeObject);

}

return result;

}

}

}

Класс *GameConfig*:

using Game.Factory;

using Game.Models;

using System.Collections.Generic;

using System.Numerics;

using System.Windows.Forms;

namespace Game

{

/// <summary>

/// Конфигурационный класс для настройки глобальных значений

/// </summary>

public class GameConfig

{

public Player Player { get; set; }

public string ServerAdress { get; set; }

public int Port { get; set; }

// Словарь с настройками

public static Dictionary<int, GameConfig> Configs = new Dictionary<int, GameConfig>

{

[1] = new GameConfig

{

Player = GameObjectFactory.CreatePlayer(

new Vector4(1.0f, 1.0f, 0.0f, 1.0f),

new Vector2(2, 0.5f),

ComponentsFactory.CreateControlSettings(Keys.W, Keys.S, Keys.A, Keys.D, Keys.Space, Keys.C)),

ServerAdress = "192.168.100.8",

Port = 8080

},

[2] = new GameConfig

{

Player = GameObjectFactory.CreatePlayer(

new Vector4(0.0f, 1.0f, 0.0f, 1.0f),

new Vector2(-2, 0.5f),

ComponentsFactory.CreateControlSettings(Keys.NumPad8, Keys.NumPad2, Keys.NumPad4, Keys.NumPad6, Keys.NumPad0, Keys.NumPad1)),

ServerAdress = "192.168.100.7",

Port = 8000

}

};

}

}

Класс *Form1*:

using System;

using System.Windows.Forms;

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

using Game.Factory;

using Vector2 = System.Numerics.Vector2;

using Vector4 = System.Numerics.Vector4;

using Game.Services;

using System.Threading.Tasks;

using System.Threading;

using Game.Models.Areas;

using System.Net.Sockets;

using System.Text;

using System.Net;

using Newtonsoft.Json;

using System.Linq;

using Game.Helper;

using System.Runtime.InteropServices;

namespace Game

{

public partial class Form1 : Form

{

private readonly GameConfig gameConfig;

private readonly string enemyServerAddres;

private readonly int enemyServerPort;

private readonly GameRepository gameRepository;

private readonly GameActionService gameActionService;

private readonly DrawingService drawingService;

private bool isGameEnd = false;

public Form1(int applicationId)

{

this.gameConfig = GameConfig.Configs[applicationId];

this.enemyServerAddres = applicationId == 1

? GameConfig.Configs[2].ServerAdress

: GameConfig.Configs[1].ServerAdress;

this.enemyServerPort = applicationId == 1

? GameConfig.Configs[2].Port

: GameConfig.Configs[1].Port;

// Создание сервисов и репозиториев

gameRepository = ComponentsFactory.CreateGameRepository();

gameActionService = ServiceFactory.CreateGameActionService();

CreateAreas();

CreatePlayers();

CreateWalls();

drawingService = ServiceFactory.CreateDrawingService();

InitializeComponent();

Task.Run(() =>

{

//RunServer();

RunCPPServer();

});

Task.Run(() =>

{

while (!isGameEnd)

{

glControl1.Invalidate();

Thread.Sleep(100);

}

});

}

public void RunCPPServer()

{

while (true)

{

IntPtr ptrRes = DLLImporter.createAndListenSocket("", (ushort)this.gameConfig.Port);

var responseData = Marshal.PtrToStringAnsi(ptrRes);

var deserializedKey = JsonConvert.DeserializeObject<Keys>(responseData);

gameActionService.ProcessPlayerAction(deserializedKey);

}

}

public void SendMessageToCPP(Keys key)

{

var responseData = DLLImporter.sendMessageToSocket((int)key, (ushort)this.enemyServerPort);

}

/// <summary>

/// Прослушивает все подключения по порту

/// </summary>

public void RunServer()

{

TcpListener server = null;

try

{

IPAddress localAddr = IPAddress.Parse(this.gameConfig.ServerAdress);

server = new TcpListener(localAddr, this.enemyServerPort);

// запуск слушателя

server.Start();

while (true)

{

byte[] data = new byte[1024];

// получаем входящее подключение

TcpClient client = server.AcceptTcpClient();

// получаем сетевой поток для чтения и записи

NetworkStream stream = client.GetStream();

// получение данных

int bytes = stream.Read(data, 0, data.Length);

var responseData = Encoding.UTF8.GetString(data, 0, bytes).ToString();

var deserializedKey = JsonConvert.DeserializeObject<Keys>(responseData);

gameActionService.ProcessPlayerAction(deserializedKey);

// закрываем поток

stream.Close();

// закрываем подключение

client.Close();

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

finally

{

if (server != null)

server.Stop();

}

}

/// <summary>

/// Обработка запросов клиента

/// </summary>

public void SendKeyToOpponent(Keys key)

{

TcpClient client = new TcpClient();

client.Connect(this.enemyServerAddres, this.enemyServerPort);

NetworkStream stream = client.GetStream();

var keyToString = JsonConvert.SerializeObject(key);

var data = Encoding.UTF8.GetBytes(keyToString);

stream.Write(data, 0, data.Length);

// Закрываем потоки

stream.Close();

client.Close();

}

private void CreateAreas()

{

gameRepository.GameAreas.Add(new NotFireArea(new Vector2(0, 2)));

gameRepository.GameAreas.Add(new DecelerationArea(new Vector2(-1, -1)));

gameRepository.GameAreas.Add(new ImpassableArea(new Vector2(-1.5f, 2)));

}

private void CreatePlayers()

{

gameRepository.Player1 = GameConfig.Configs[1].Player;

gameRepository.Player2 = GameConfig.Configs[2].Player;

}

private void CreateWalls()

{

for (var x = Constants.BorderWallLeftTopX; x >= Constants.BorderWallRigftBottomX; x -= Constants.BorderWallSize)

{

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(x, Constants.BorderWallLeftTopY),

Constants.BorderWallSize,

Constants.BorderWallHealth));

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(x, Constants.BorderWallRigftBottomY),

Constants.BorderWallSize,

Constants.BorderWallHealth));

}

for (var y = Constants.BorderWallLeftTopY; y >= Constants.BorderWallRigftBottomY; y -= Constants.BorderWallSize)

{

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(Constants.BorderWallLeftTopX, y),

Constants.BorderWallSize,

Constants.BorderWallHealth));

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(Constants.BorderWallRigftBottomX, y),

Constants.BorderWallSize,

Constants.BorderWallHealth));

}

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(1, 1.5f),

Constants.WallSize,

Constants.WallHealth));

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(2, 1.5f),

Constants.WallSize,

Constants.WallHealth));

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(1, 0f),

Constants.WallSize,

Constants.WallHealth));

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(-2, 0f),

Constants.WallSize,

Constants.WallHealth));

gameRepository.Walls.Add(GameObjectFactory.CreateWall(

new Vector2(2, 0f),

Constants.WallSize,

Constants.WallHealth));

}

private void GlControl1\_Load(object sender, EventArgs e)

{

GL.ClearColor(Color4.SkyBlue);

}

private void GlControl1\_Resize(object sender, EventArgs e)

{

GL.Viewport(0, 0, glControl1.Size.Width, glControl1.Size.Height);

GL.MatrixMode(MatrixMode.Projection);

var projection = Matrix4.CreatePerspectiveFieldOfView(

(float)Math.PI / 2,

glControl1.Size.Width / (float)glControl1.Size.Height,

1.0f,

64.0f);

GL.LoadMatrix(ref projection);

}

private void GlControl1\_Paint(object sender, PaintEventArgs e)

{

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

player1Info.Text = gameRepository.Player1.ToString();

player2Info.Text = gameRepository.Player2.ToString();

DrawObjects();

// Проверка столкновений пуль с чем-нибудь и обработка

gameActionService.ProcessBulletsCollisions();

// Уничтожение мёртвых объектов

gameActionService.DisposeDeadObjects();

// Обработка полёта пули

gameActionService.ProcessBulletsMoving();

// Проверка условий победы

if (gameActionService.IsFirstPlayerWon())

{

glControl1.Dispose();

MessageBox.Show("First player won");

isGameEnd = true;

return;

}

if (gameActionService.IsSecondPlayerWon())

{

glControl1.Dispose();

MessageBox.Show("Second player won");

isGameEnd = true;

return;

}

glControl1.SwapBuffers();

GL.Clear(ClearBufferMask.ColorBufferBit);

GL.LoadIdentity();

}

private void DrawObjects()

{

foreach (var gameObject in gameRepository.GetAllGameObjects())

{

drawingService.DrawObject(gameObject);

}

}

private void GlControl1\_KeyDown(object sender, KeyEventArgs e)

{

// Проверка, чтобы не было введено запрещённых действий, например, движение другого игрока

var currentPlayerControl = this.gameConfig.Player.Control;

var currentPlayerActions = new Keys[]

{

currentPlayerControl.Down,

currentPlayerControl.Up,

currentPlayerControl.Left,

currentPlayerControl.Right,

currentPlayerControl.Fire,

currentPlayerControl.FireMiniGun,

};

var isKeyValidForThisApplication = currentPlayerActions.Contains(e.KeyCode);

if (!isKeyValidForThisApplication)

{

return;

}

// Обработка ввода

gameActionService.ProcessPlayerAction(e.KeyCode);

//SendKeyToOpponent(e.KeyCode);

SendMessageToCPP(e.KeyCode);

glControl1.Invalidate();

}

}

}

Файл реализация библиотеки *dllmain*.*cpp*:

// dllmain.cpp : Определяет точку входа для приложения DLL.

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include "pch.h"

#include "WSDll.h"

#undef UNICODE

#define WIN32\_LEAN\_AND\_MEAN

#include <winsock2.h>

#include <ws2tcpip.h>

#include <stdlib.h>

#include <string>

#include <iostream>

#include <sstream>

#include <fstream>

#pragma comment (lib, "Ws2\_32.lib")

#define DEFAULT\_BUFLEN 512

char SERVERADDR[15];

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

switch (ul\_reason\_for\_call)

{

case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

\_\_declspec(dllexport) char\* createAndListenSocket(char \* \_, u\_short port) {

// Чтенние ip из файла

std::string line;

std::ifstream in("D:\\ipaddress.txt");

if (in.is\_open())

{

while (getline(in, line))

{

std::cout << line << std::endl;

}

}

in.close();

// Initialize Winsock

WSADATA wsaData;

int iResult = 0;

SOCKET ListenSocket = INVALID\_SOCKET;

sockaddr\_in service;

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != NO\_ERROR) {

throw L"WSAStartup() failed with error: " + iResult;

}

// Create a SOCKET for listening for incoming connection requests.

ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (ListenSocket == INVALID\_SOCKET) {

WSACleanup();

throw L"socket function failed with error: " + WSAGetLastError();

}

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

// IP address, and port for the socket that is being bound.

service.sin\_family = AF\_INET;

//service.sin\_addr.s\_addr = inet\_addr("192.168.100.7");

service.sin\_port = htons(port);

// преобразование IP адреса из символьного в сетевой формат

HOSTENT\* hst;

if (inet\_addr(SERVERADDR) != INADDR\_NONE)

service.sin\_addr.s\_addr = inet\_addr(SERVERADDR);

else

{

// попытка получить IP адрес по доменному имени сервера

if (hst = gethostbyname(SERVERADDR))

// hst>h\_addr\_list содержит не массив адресов, а массив указателей на адреса

((unsigned long\*)&service.sin\_addr)[0] =

((unsigned long\*\*)hst->h\_addr\_list)[0][0];

else

{

//printf("Invalid address %s\n", SERVERADDR);

closesocket(ListenSocket);

WSACleanup();

throw L"Invalid address %s\n", SERVERADDR + WSAGetLastError();;

}

}

//=================

iResult = bind(ListenSocket, (SOCKADDR\*)&service, sizeof(service));

if (iResult == SOCKET\_ERROR) {

iResult = closesocket(ListenSocket);

WSACleanup();

char res[10] = "";

\_itoa\_s(WSAGetLastError(), res, 10);

return res;

}

// Listen for incoming connection requests

// on the created socket

if (listen(ListenSocket, SOMAXCONN) == SOCKET\_ERROR)

wprintf(L"listen function failed with error: %d\n", WSAGetLastError());

wprintf(L"Listening on socket...\n");

// Get data from client

SOCKET ClientSocket = INVALID\_SOCKET;

// Accept a client socket

ClientSocket = accept(ListenSocket, NULL, NULL);

if (ClientSocket == INVALID\_SOCKET) {

closesocket(ListenSocket);

WSACleanup();

throw "accept failed with error: " + WSAGetLastError();

}

char recvbuf[DEFAULT\_BUFLEN];

int recvbuflen = DEFAULT\_BUFLEN;

// Receive until the peer shuts down the connection

do {

iResult = recv(ClientSocket, recvbuf, recvbuflen, 0);

} while (iResult > 0);

// Close socket

iResult = closesocket(ListenSocket);

if (iResult == SOCKET\_ERROR) {

WSACleanup();

throw L"closesocket function failed with error " + WSAGetLastError();

}

WSACleanup();

return recvbuf;

}

\_\_declspec(dllexport) int sendMessageToSocket(int key, u\_short port) {

char serverAddress[] = "192.168.100.8";

WSADATA wsaData;

SOCKET ConnectSocket = INVALID\_SOCKET;

struct addrinfo\* result = NULL,

\* ptr = NULL,

hints;

int iResult;

char sendbuf[4] = "";

\_itoa\_s(key, sendbuf, 10);

char portbuf[5] = "";

\_itoa\_s(port, portbuf, 10);

// Initialize Winsock

iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);

if (iResult != 0) {

throw "WSAStartup failed with error";

}

ZeroMemory(&hints, sizeof(hints));

hints.ai\_family = AF\_UNSPEC;

hints.ai\_socktype = SOCK\_STREAM;

hints.ai\_protocol = IPPROTO\_TCP;

// Resolve the server address and port

iResult = getaddrinfo(serverAddress, portbuf, &hints, &result);

if (iResult != 0) {

WSACleanup();

throw "getaddrinfo failed with error";

}

// Attempt to connect to an address until one succeeds

for (ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

// Create a SOCKET for connecting to server

ConnectSocket = socket(ptr->ai\_family, ptr->ai\_socktype,

ptr->ai\_protocol);

if (ConnectSocket == INVALID\_SOCKET) {

WSACleanup();

throw "socket failed with error";

}

// Connect to server.

iResult = connect(ConnectSocket, ptr->ai\_addr, (int)ptr->ai\_addrlen);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

ConnectSocket = INVALID\_SOCKET;

continue;

}

break;

}

freeaddrinfo(result);

if (ConnectSocket == INVALID\_SOCKET) {

WSACleanup();

throw "Unable to connect to server!\n";

}

// Send an initial buffer

iResult = send(ConnectSocket, sendbuf, 4, 0);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

WSACleanup();

throw "send failed with error";

}

printf("Bytes Sent: %ld\n", iResult);

// shutdown the connection since no more data will be sent

iResult = shutdown(ConnectSocket, SD\_SEND);

if (iResult == SOCKET\_ERROR) {

closesocket(ConnectSocket);

WSACleanup();

throw "shutdown failed with error";

}

// cleanup

closesocket(ConnectSocket);

WSACleanup();

return key;

}

Файл заголовка *wsdll*.*h*:

#pragma once

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#include <windows.h>

#include <WinSock2.h>

#define WINSOCKLIBRARY\_API \_\_declspec(dllexport)

extern "C" WINSOCKLIBRARY\_API char\* createAndListenSocket(char\* serverAddress, u\_short port);

extern "C" WINSOCKLIBRARY\_API int sendMessageToSocket(int key, u\_short port);

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Схема иерархии классов**

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Руководство пользователя**

1. Введение.

Руководство пользователя обеспечивает получение пользователем базовых навыков по эксплуатации игрового приложения.

Разработанное программное приложение предназначено для игры двух игроков на одном экране.

Игровое приложение обладает следующим функционалом:

* игровое поле в виде полигона;
* передвижение игроков;
* разные виды препятствий;

– разные виды патронов.

Для использования программного приложения пользователь должен быть ознакомлен с:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

2. Назначение и условия применения.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере. Данная программа помогает провести время с друзьями, улучшить реакцию, учит рассуждать и логически мыслить, развивает образное и пространственное мышление. Игра позволяет учиться быстрее реагировать на неожиданные повороты событий, а это поможет в учёбе и в жизни. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы игрового приложения необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие следующих устройств ввода: стандартная клавиатура и сенсорная панель;

– наличие устройства вывода – экран.

3. Подготовка к работе.

Для установки приложения необходимо загрузить на компьютер готовую программу с игрой. Также требуется, чтобы на компьютере была установлена библиотека *OpenGL* последней версии. После этого в папке с программой зайти в *Game* и запустить файл *Game.exe.*

Если все инструкции соблюдены и приложение не выдаёт никаких сообщений об ошибках, значит программа работает исправно.

4. Аварийные ситуации.

Чтобы избежать ошибок при использовании программы, необходимо соблюдать порядок действий и условия пользования, описанные в пункте 3 данного руководства пользователя.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова.

5. Правила игры «Танки».

При запуске игрового приложения будет показано игровое поле, на котором находится два игрока и стены. Начальный экран игры представлен на рисунке В.1:

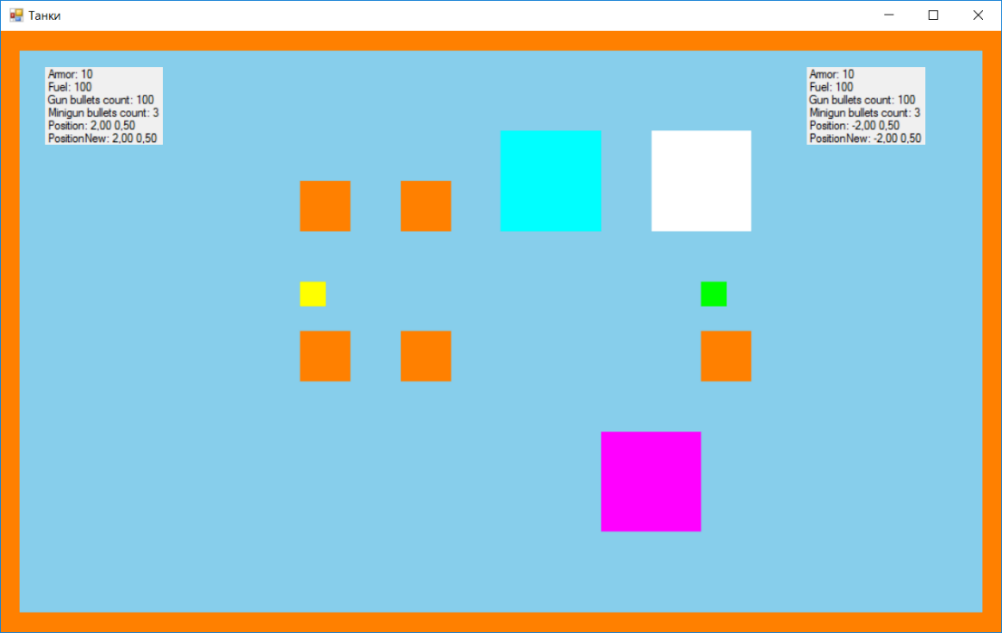


Рисунок В.1 – Игровое поле

Игра *«*Танки*»* рассчитана на двух игроков, которые в начале игры находятся напротив друг друга. Каждый из них управляет своим танком с помощью клавиатуры. Первый игрок использует клавиши *W*, *S*, *A*, *D* дляперемещения, *Space* для выстрела снарядом, *С* для выстрела пушкой*,* второй игрок *– Numpad8, Numpad2, Numpad4, Numpad6* дляперемещения, *Numpad0* для выстрела снарядом, *Numpad1* для выстрела пушкой.Танки имеют одинаковые характеристики: скорость движения, броню, вооружение снарядов и топливо.

Танки перемещаются по игровому полю с неоднородной территорией: участки с замедлением скорости; участки, на которых невозможно производить выстрелы; непреодолимые участки, но через которые можно производить выстрелы; участки, через которые нельзя проехать и нельзя стрелять. Пули из пулемёта достигают цели моментально, у снарядов скорость ограниченна, так что танки могут уклоняться от выстрелов. Победителем считается игрок, уничтоживший танк соперника.

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство программиста**

1. Назначение и условия применения.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере. Данная программа помогает провести время с друзьями, улучшить реакцию, учит рассуждать и логически мыслить, развивает образное и пространственное мышление. Игра позволяет учиться быстрее реагировать на неожиданные повороты событий, а это поможет в учёбе и в жизни. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы игрового приложения необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие следующих устройств ввода: стандартная клавиатура и сенсорная панель;
* наличие устройства вывода – экран

2. Характеристики программы.

Приложение работает в автоматическом режиме. При запуске открывается главное окно программы, в котором показано игровое поле, то есть лабиринт. На этом игровом поле два игрока, которыми можно управлять с помощью клавиатуры.

3. Обращение к программе

Приложение запускается путём открытия файла *Game.exe* находящимся в папке *Game.*

4. Входные и выходные данные

В данной программе в качестве исходных данных используются язык программирования *C#*, *OpenGL*, *Windows* *Form* и шаблоны проектирования, такие как: «фабричный метод» и «декоратор».

«Фабричный метод» – реализует генерацию игровых объектов, «декоратор» – изменяет характеристики игрока.

# **ПРИЛОЖЕНИЕ Д**

(обязательное)

**Руководство системного программиста**

1. Назначение и условия применения

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере. Данная программа помогает провести время с друзьями, улучшить реакцию, учит рассуждать и логически мыслить, развивает образное и пространственное мышление. Игра позволяет учиться быстрее реагировать на неожиданные повороты событий, а это поможет в учёбе и в жизни. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы игрового приложения необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие следующих устройств ввода: стандартная клавиатура и сенсорная панель;
* наличие устройства вывода – экран.

2. Структура программы

Игровое приложение логически можно разбить на несколько составляющих: графический интерфейс пользователя, средства работы с графикой и непосредственно логику игровых объектов и игрового процесса.

Графический интерфейс включает отображение игрового поля.

Графическая составляющая приложения отвечает за процесс рендеринга сцены и отображения состояния игровой сессии.

Логика игровых объектов включает в себя реализацию игровых механик и логику поведения игровых объектов.

3. Проверка программы

Отсутствие сообщения об ошибках свидетельствует о корректной работе приложения.