**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ ………………………………………………………………...….. 4

1 СРЕДСТВА СОЗДАНИЯ ИГРОВОГО ПРИЛОЖЕНИЯ………………..... 5

1.1 Графическая библиотека OpenGL............................................................... 5

1.2 Использование OpenGl в языке программирования С#……………6

1.3 Интерфейс программирования приложений Windows Form.....................7

1.4 Использование OpenTK в приложениях Windows Form……………8

Список используемых источников ……………………………………11

**ВВЕДЕНИЕ**

В досуге современного человека компьютерные игры занимают достаточно важное место. Для многих игры – это вообще повседневное развлечение. 2020 год показал насколько игры важны для людей и перспективны для развития.

Для сравнения, объем рынка игр уже обогнал другие традиционные развлечения, такие как музыка и фильмы вместе взятые, $145.7 против $21.5 и $101 миллиардов соответственно.

Поэтому сегодня поговорим о более сложной и не всем знакомой части игр, это их разработка. Рассмотрим один из языков программирования для разработки игровых приложений, различные методы создания игр и технологии.

Компьютерная графика нашла широкое распространение и применение в повседневной жизни. Учёные используют компьютерную графику для анализа результатов моделирования. Инженеры и архитекторы используют трёхмерную графику для создания виртуальных моделей. Кинематографисты создают спецэффекты или полностью анимированные фильмы («Шрек», «История игрушек» и др.). В последние годы широкое распространение получили также компьютерные игры, максимально использующие трёхмерную графику для создания виртуальных миров.

В нашем проекте будет использоваться графическая библиотека OpenGl и средства языка программирования C# Windows Forms.

В ходе выполнения проекта мы окунемся в мир игровой разработки и самое главное получим достаточно большой багаж новых знаний, начиная от опыта работы над серьёзными проектами, заканчивая работой с различными паттернами программирования. Выполнив данный проект будут освоены и применены на практике различные технологии и средства разработки игровых приложений.

1. **СРЕДСТВА СОЗДАНИЯ ИГРОВЫХ ПРИЛОЖЕНИЙ**
   1. **Графическая библиотека OpenGL**

Для создания графического интерфейса и отображения объектов в нашем игровом приложении будем использовать компьютерную графику OpenGL.

*OpenGL* (*Open Graphics Library*) – спецификация, определяющая платформонезависимый (независимый от [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) [программный интерфейс](https://ru.wikipedia.org/wiki/API) для написания приложений, использующих двумерную и трёхмерную [компьютерную графику](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0). Разрабатывается в США и Европе, имеет тип лицензий *GNU*-/*EU*/.

Включает более 300 функций для рисования сложных трёхмерных сцен из простых примитивов. Используется при создании [компьютерных игр](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%B8%D0%B3%D1%80%D0%B0), [САПР](https://ru.wikipedia.org/wiki/%D0%A1%D0%90%D0%9F%D0%A0), [виртуальной реальности](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C), визуализации в научных исследованиях. Эффективные реализации *OpenGL* существуют для *Windows*, *Unix*-платформ, *PlayStation* 3 и *Mac* *OS*. На платформе [*Windows*](https://ru.wikipedia.org/wiki/Windows) конкурирует с [*Direct3D*](https://ru.wikipedia.org/wiki/Direct3D).

На базовом уровне, *OpenGL* – это просто спецификация, то есть документ, описывающий набор функций и их точное поведение. Производители оборудования на основе этой спецификации создают реализации – библиотеки функций, соответствующих набору функций спецификации.

*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 (чаще всего дерева), которое 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 работы различных 3D ускорителей от разработчика, предоставляя единый API;

– Скрыть различия в аппаратных возможностях устройств, требуя программной эмуляции недостающего функционала.

Отличительной особенностью *OpenGL* является поддержка расширений. Стандарт *OpenGL*, с появлением новых технологий, позволяет отдельным производителям добавлять в библиотеку функциональность через механизм расширений. Всякий раз, когда графическая компания выкатывает новую методику или новую большую оптимизацию для рендеринга, это часто встречается в расширении, реализованном в драйверах. Если оборудование, на котором работает приложение, поддерживает такое расширение, то разработчик может использовать функционал, предоставляемый этим расширением, для более продвинутой или эффективной графики. Таким образом, графический разработчик уже может использовать новые методы рендеринга, просто проверяя, поддерживается ли данное расширение видеокартой, при этом не дожидаясь, пока *OpenGL* добавит этот функционал в свою новую версию. Часто, когда расширение является популярным или очень полезным, оно в конечном итоге становится частью новой версии *OpenGL*.

Разработчик должен знать, доступны ли какие-либо из этих расширений, прежде чем их использовать (или использовать библиотеку расширений *OpenGL*). Это позволяет разработчику делать вещи лучше или эффективнее в зависимости от того, доступно ли расширение.

**1.2 Использование OpenGl в языке программирования** С#

Так как для реализации игрового приложения будет использоваться язык программирования C#, будем использовать библиотеку OpenTK. OpenTK – это одна из самых простых и популярных библиотек обеспечивающих доступ к графические инструментам OpenGL.

*Open* *Toolkit* является бесплатным проектом, который позволяет использовать *OpenGL*, *OpenGL* | *ES*, *OpenCL* и *OpenAL* *API*, из управляемых языков.

*OpenTK* начал жизнь в качестве экспериментальной вилки рамках *Тао*, прежде чем летом первоначального намерения 2006. Это был обеспечить чистый обертку, чем *Tao*. *OpenGL*, но он быстро вырос в фокусе: прямо сейчас, он обеспечивает доступ к различным *Khronos* и *Creative*, *API* для ручки и необходимую логику инициализации для каждого API. Таким образом, *Open* *Toolkit*, наиболее похожий на таких проектах, как *Тао*, *SlimDX*, *SDL* или *GLFW*.

В отличие от аналогичных библиотек, попытки *OpenTK* обеспечить единый интерфейс, который использует превосходную удалось выполнения. Вместо того, чтобы нетипизированных указателей, *OpenTK* обеспечивает дженерики. Вместо простых констант, *OpenTK* использует строго типизированные перечисления. Вместо списков простых функций, *OpenTK* отделяет функции за расширение категории. Общая математическая библиотека является интегрированной и непосредственно использовать в каждом *API*.

**1.3 Интерфейс программирования приложений Windows Form**

Отображать графику OpenGL будем с помощью с графического интерфейса Windows forms.

Windows Forms – интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависят от языка разработки.

То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на др.

На рисунках 1–2 демонстрируется создание приложения в Windows Forms.

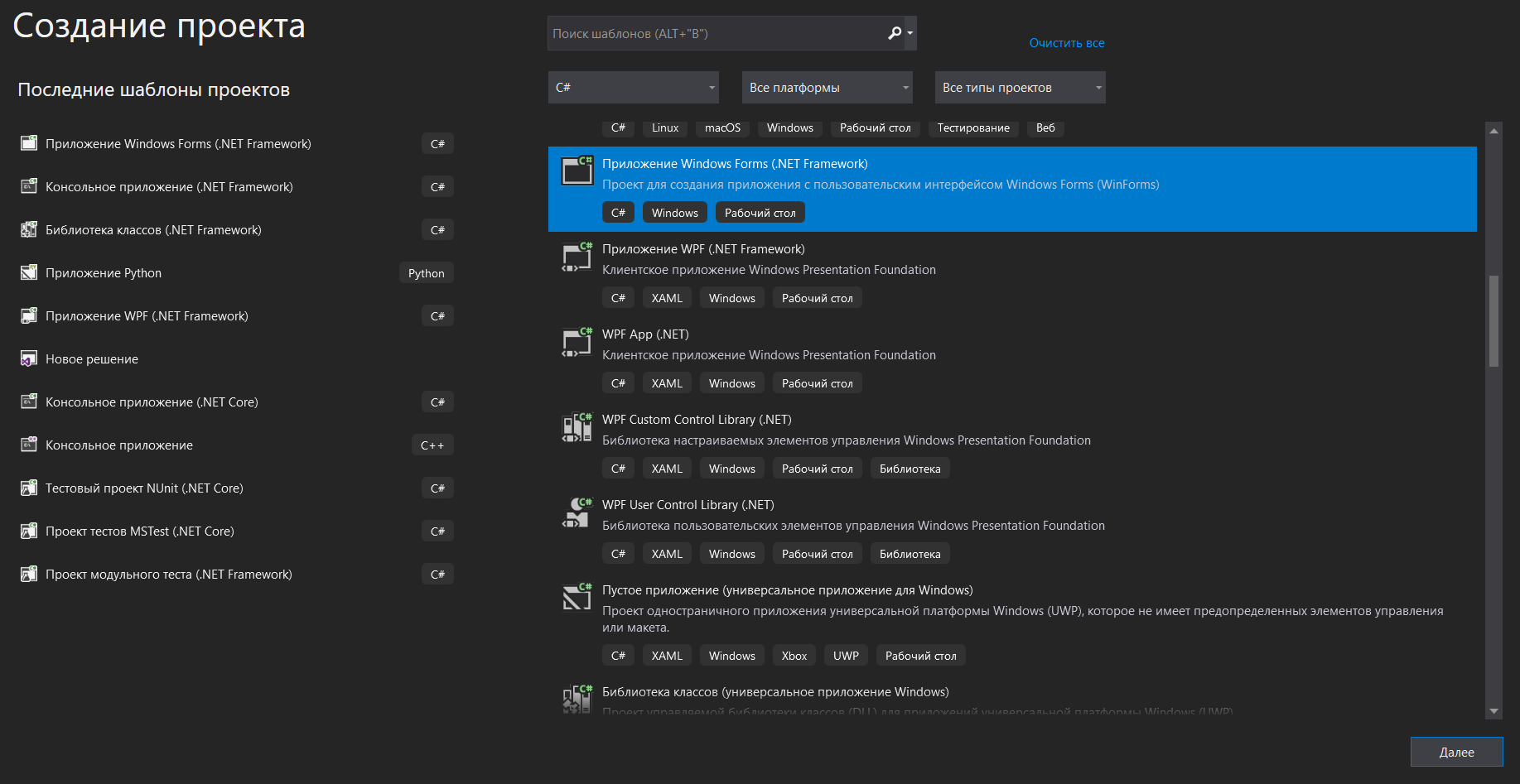


Рисунок 1.1 – Создание приложения Windows Forms

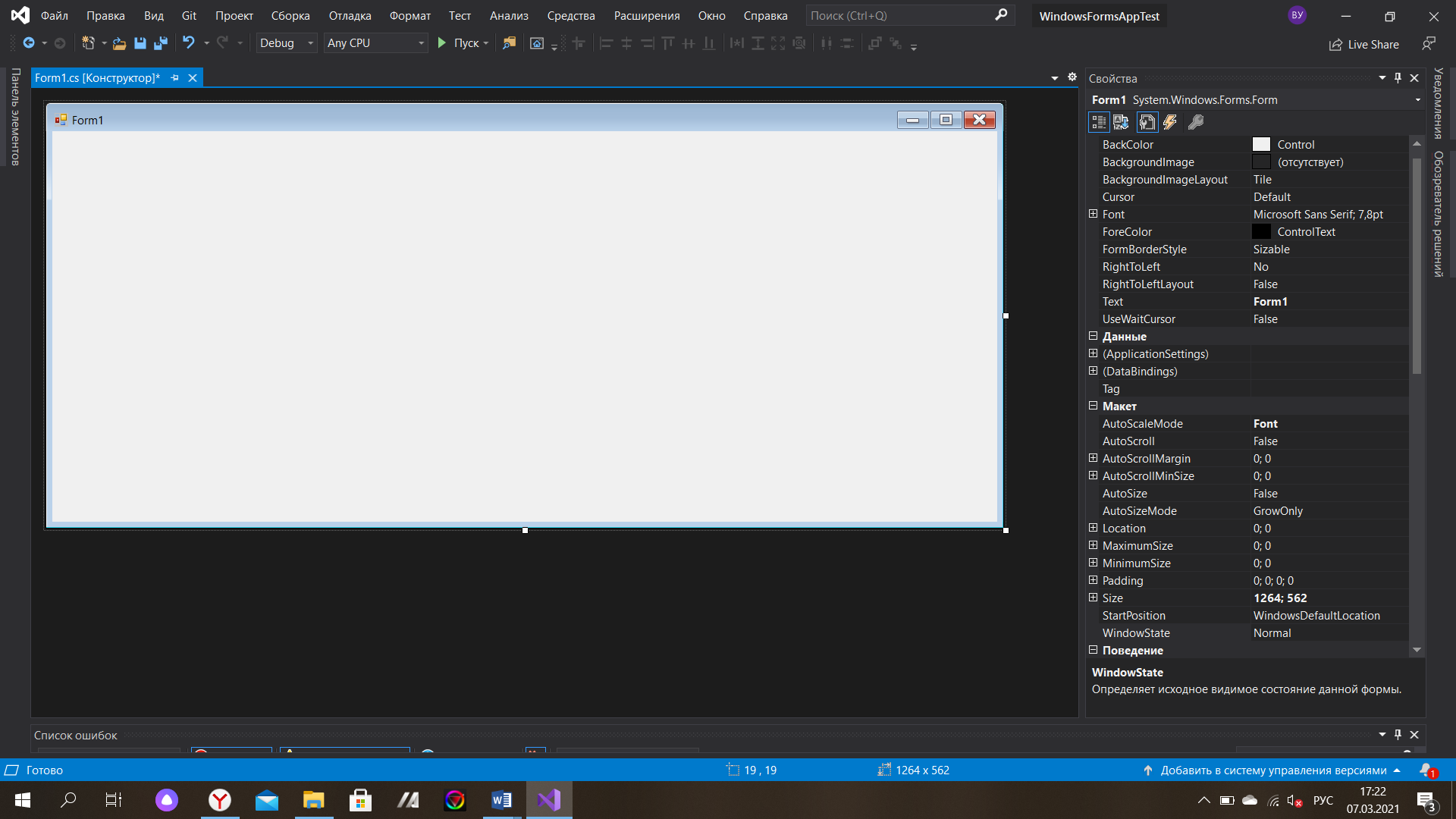


Рисунок 1.2 – Интерфейс работы с приложением

**1.4 Использование OpenTK в приложениях Windows Form**

Библиотка OpenTKудобна тем, что ставится очень быстро и легко через NuGet и позволяет, как создавать своё окно из консольного проложения, так и встраивать холст для рисования OpenGL графики в WPF и WinForms приложения, сочетая 2D/3D графику со стандартным GUI в одном приложении.

Для подключения OpenTK к нашему приложению воспользуемся NuGet – системой управления пакетами для платформ разработки Microsoft. Чтобы получить весь функционал нам потребуется подключить две библиотеки OpenTK и OpenTK.GLControl.

Библиотека OpenTK – это набор быстрых низкоуровневых привязок C# для OpenGL.

Библиотека OpenTK.GLControl выполняет инициализацию OpenGL с помощью графический компонент GLControl, который размещается на одной из форм приложения. Вывод изображения выпоняется в области компонента GLControl.

Подключение OpenTK в приложение Windows Forms:

1. Заходим в диспетчер пакетов NuGet и устанавливаем пакеты OpenTK и OpenTK.GLControl;

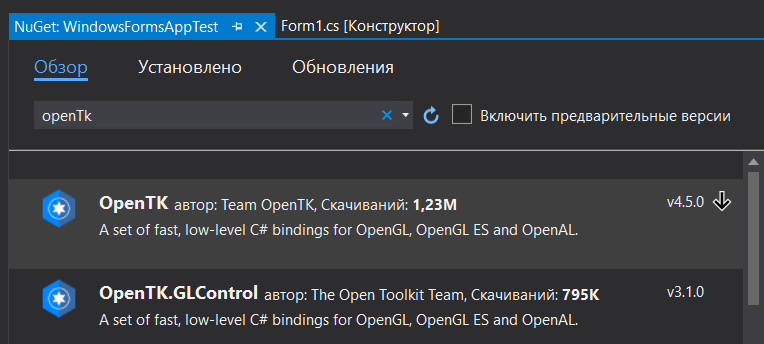


Рисунок 1.3 – Диспетчер пакетов NuGet

1. Для того чтобы вывести изображение необходимо добавить компонент GLControl на нашу форму. Заходим в панель элементов вкладка OpenTK.

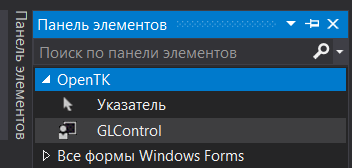


Рисунок 1.4 – Добавление компонент GLControl

1. Добавим GLControl на форму;

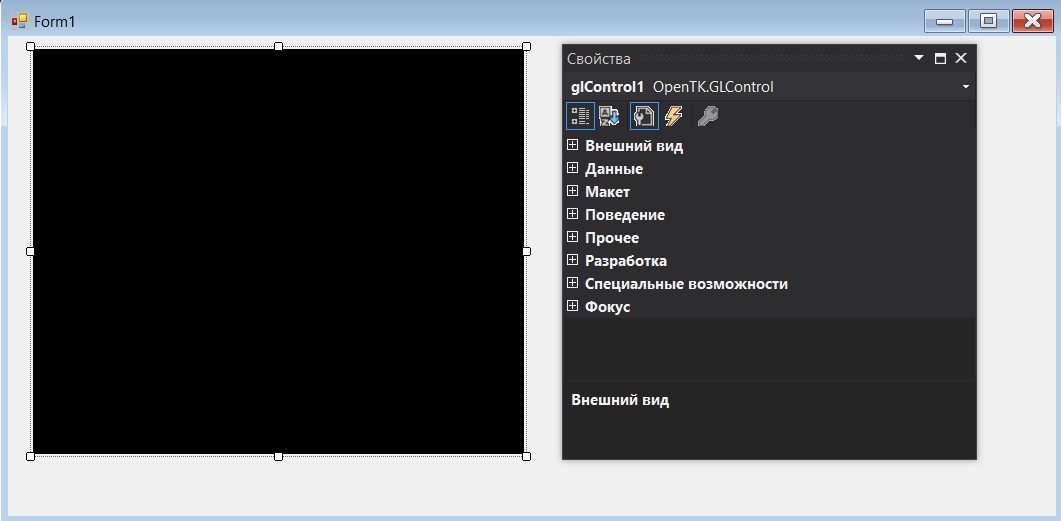


Рисунок 1.5 – Компонент GLControl

В этой главе было рассказано о графической библиотеке OpenGl и её использование в языке программирования С#, посредством библиотеки OpenTK. Были приведены базовые знания о библиотеках и их взаимодействии, так же приведён пример создания пустого проекта с нужными нам технологиями.

Для наращивания функционала игрового приложения, нужно использовать множество различных технологий и функций, о них будет рассказано в следующих главах.

1. **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «СРАЖЕНИЕ ПОДВОДНЫХ ЛОДОК»**
   1. **Архитектура игрового приложения**

Необходимый функционал игрового приложения:

– Удобное меню игры;

– Возможность редактирование характеристик лодки в начале игры;

– Многопользовательское управление;

– Возможность ведения боя между двумя игроками;

– Получение бонусов во время игры;

– Вывод итога игры.

Для разработки игровой логики приложения необходимо создать собственным игровой движок. Игровой движок – это программная реализация базового функционала игрового приложения. Для создания игрового движка будет создана библиотека классов, которая будет содержать классы определяющие функционал движка. Со стороны графической реализации игры игровой движок должен уметь работать со спрайтовой графикой и анимацией, то есть уметь отображать и изменять графические объекты. Со стороны функционала игрового приложения игровой движок должен уметь считывать и изменять все необходимые действия игровых объектов, уметь изменять положение, размер, считывать столкновения игровых объектов и т. д.

Для того чтобы реализовать всю логику игры и использовать игровой движок необходимо создать проект, который будет включать в себя классы различных игровых объектов и логику.

Так же при написании игры будут использоваться паттерны программирования декоратор и фабричный метод. Декоратор будет применяться для изменений характеристик подводной лодки, а фабричный метод будут случайным образом генерировать призы на сцене. Данные паттерны так же будут реализованы в этом проекте.

Далее необходимо создать проект приложения Windows Forms. Главное предназначение проекта Windows Forms заключается в том, чтобы обеспечить удобное меню игрового приложения. Так же в этом проекте будет реализован

класс главной сцены игры, в нём будет задействован игровой движок и библиотека классов с различной логикой. Так как логика и графика игрового приложения будет разделена на разные слои, то в этом же проекте будет осуществлено управление игрой c клавиатуры для двух игроков, путем ввода с клавиатуры.

* 1. **Структура игрового приложения «Сражение подводных лодок»**

При написании игры будут использоваться паттерны программирования декоратор и фабричный метод. Декоратор будет применяться для изменений характеристик подводной лодки, а фабричный метод будут случайным образом генерировать призы на сцене. Данные паттерны так же будут реализованы в этом проекте.

Паттерн Декоратор (Decorator) – представляет структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность.

Данный паттерн следует использовать, когда объекту нужно добавлять новый функционал. Отличительной особенностью данного паттерна от обычного наследования является возможность убрать у объекта добавленный функционал, прямо во время работы программы. Т.е. декоратор позволяет снимать декорации, но снятие декораций должно происходить в порядке, обратном наложению.

На рисунке 2.3 демонстрируется схема реализации паттерна «Декоратор».

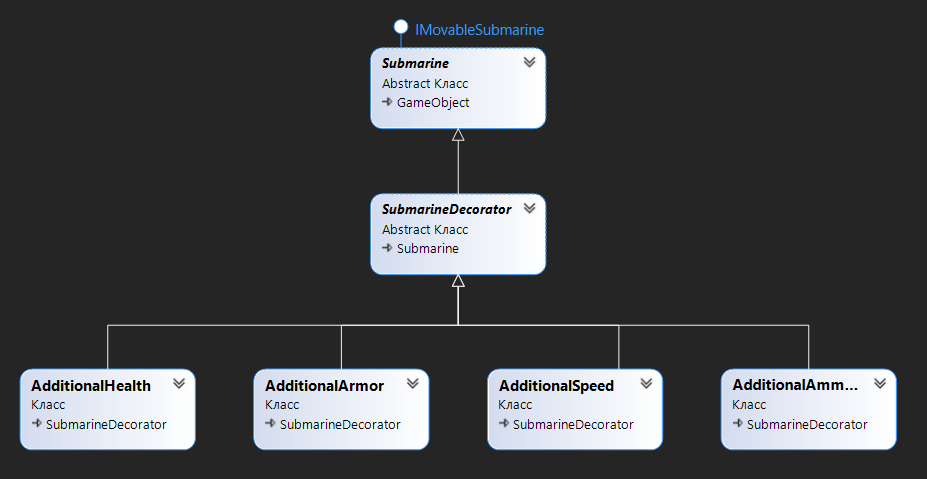


Рисунок 2.3 – схема паттерна «Декоратор»

Описание классов участвующих в реализации паттерна:

1. Submarine – данный класс представляет подводную лодку. Часто классы которые определяют интерфейс для наследуемых объектов, являются абстрактными. Применение декораций возможно, так как Submarine содержит виртуальные методы декорируемых свойств. В данном случае это не так, так как подводные лодки не отличаются характеристиками, по желанию только спрайтами.
2. SubmarineDecorator – сам декоратор, реализуется посредством абстрактного класса и имеет тот же базовый класс, что и декорируемые объекты. Наследники данного класса представляют дополнительные функциональности, которыми должен быть расширен объект Submarine.
3. AdditionalAmmunition – дополнительный функционал для класса Submarine в виде пополнения боеприпасов.
4. AdditionalArmor – дополнительный функционал для класса Submarine в виде увеличения брони.
5. AdditionalHealth – дополнительный функционал для класса Submarine в виде пополнения здоровья.
6. AdditionalSpeed – дополнительный функционал для класса Submarine в виде увеличения скорости.

Фабричный метод (Factory method) – это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

В каких случаях необходимо применять данный паттерн:

– 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 заранее неизвестно, объекты каких типов необходимо создавать;

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

На рисунке 2.4 демонстрируется схема реализации паттерна «Фабричный метод».

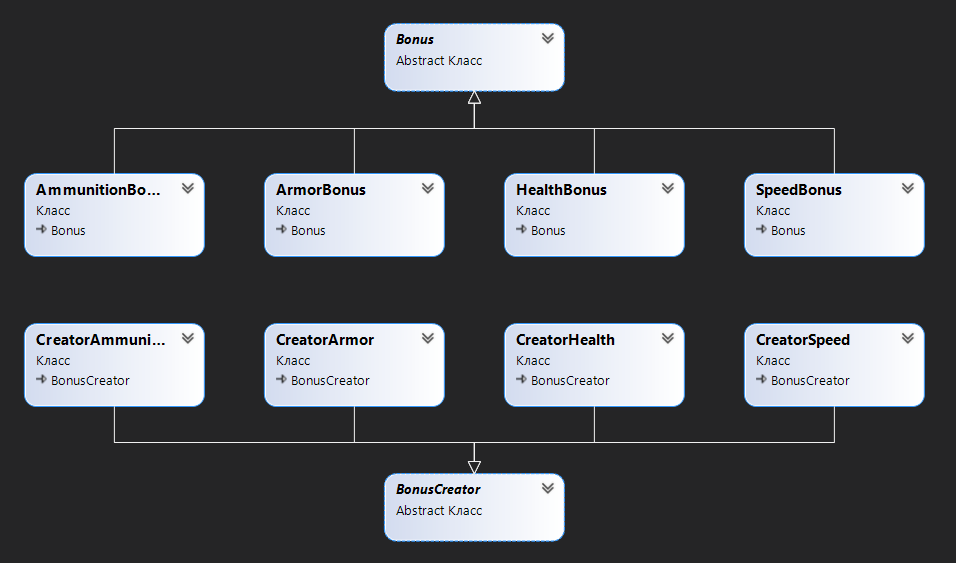


Рисунок 2.4 – схема паттерна «Фабричный метод»

Описание классов участвующих в реализации паттерна:

1. Bonus – абстрактный класс определяющий интерфейс класса, объекты которого необходимо генерировать;
2. AmminitionBonus – конкретный класс представляющий реализацию класса Bonus. Класс реализует пополнение боеприпасов;
3. SpeedBonus – конкретный класс представляющий реализацию класса Bonus. Класс реализует увеличение скорости;
4. HealthBonus – конкретный класс представляющий реализацию класса Bonus. Класс реализует пополнение здоровья;
5. ArmorBonus – конкретный класс представляющий реализацию класса Bonus. Класс реализует пополнение брони;
6. BonusCreator – абстрактный класс определяющий абстрактный фабричный метод CreateBonus, который возвращает объект Bonus. Наследники данного класса определяют свою реализацию метода CreatorBonus. Метод CreateBonus каждого отдельного класса-создателя возвращает определенный конкретный тип бонуса.
7. CreatorSpeed – возвращает класс с бонусом скорости.
8. CreatorHealth – возвращает класс с бонусом здоровья.
9. CreatorArmor – возвращает класс с бонусом брони.
10. CreatorAmmunition – возвращает класс с бонусом боеприпасов.

**2.3 Схема взаимодействия классов приложения «Сражение подводных лодок»**

Для более удобной реализации игрового приложения было создано два проекта, являющимися библиотеками классов. Это библиотека GameEngine и GameLogic. Библиотека GameEngine содержит все основные классы реализующие функционал игрового движка, а библиотека GameLogic содержит реализацию паттернов, различные игровые объекты и остальные классы игровой логики.

Подробно рассмотрим библиотеку GameEngine и опишем взаимодействие классов библиотеки.

Классы библиотеки GameEngine:

1. GameObject является абстрактным базовым типом для всех игровых объектов. GameObject содержит в себе список компонентов игрового объекта (GameComponent);
2. GameComponent является базовым типом для всех компонентов игры, он позволяет добавлять игровым объектам различные свойства и функционал;
3. Класс SpriteRenderer является производным от GameComponent и отвечает за отображение объекта на сцене;
4. Класс Collider является производным от GameComponent и содержит в себе коллайдер игрового объекта;
5. Класс Animator является производным от GameComponent и отвечает за анимацию игровых объектов;
6. Класс Transform содержится в игровом объекте GameObject как обязательный параметр, так как все объекты должны иметь положение, вращение и масштаб. Класс Transform содержит такие поля как:

– Position – положение игрового объекта в координатах x, y;

– Rotation – вращение игрового объекта вокруг осей x, y;

– Scale – масштаб игрового объекта.

1. Класс Collider является абстрактным базовым типом для коллайдеров;
2. Класс BoxCollider является наследником класса Collider и описывает коллайдеры, которые имеют прямоугольную форму;
3. Класс Texture2D хранит в себе номер в буфере и размеры добавленной текстуры;
4. Статический класс TextuteProcessing создаёт текстуру на основы добавленного изображения и выводит класс Texture2D со всеми данными о текстуре.
5. SpriteRendering – класс, который имеет статический метод отображения спрайтов.

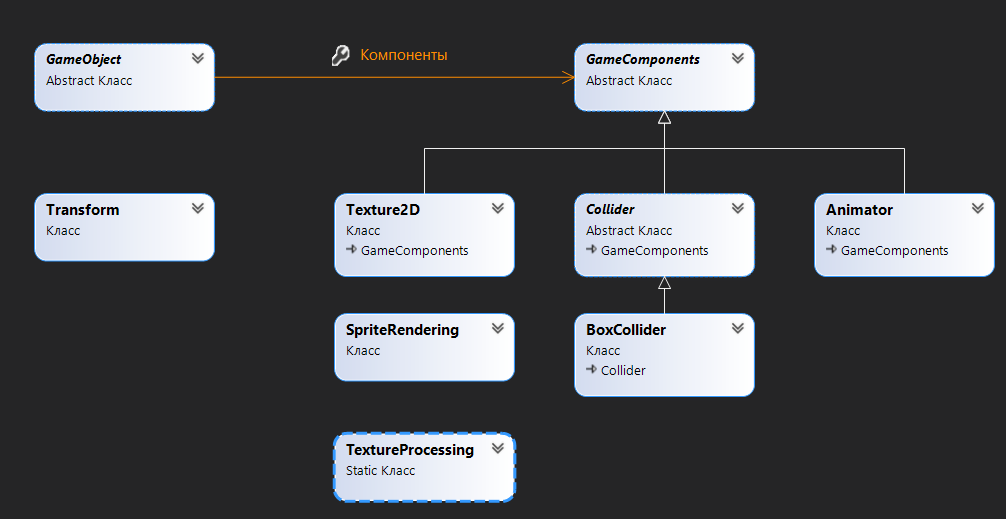


Рисунок 2.1 – схема игрового движка

Теперь рассмотрим библиотеку GameLogic и опишем взаимодействие классов библиотеки. Для таких игровых объектов как подводная лодка и ракета будет использоваться наследование. В случае с подводной лодкой наследование будет использоваться для того, чтобы различать тип лодок при попадании в них ракет, хотя подводные лодки не отличаются характеристиками, по желанию только спрайтами. Для ракет наследование будет использоваться, так как ракеты отличаются характеристиками в зависимости от типа ракеты.

Рассмотрим игровые объекты библиотеки GameLogic.

Игровые объекты:

– Подводная лодка, которой будут управлять игроки;

– Различного вида ракеты, которыми будет стрелять подводная лодка. В зависимости от вида наносит различные повреждения;

– Различного вида призы, каждый приз будет увеличивать различные свойства подводной лодки;

– Миноносец который будет периодически появляться и сбрасывать на игроков мины;

– Так же границы сцены являются игровыми объектами, так как имеют коллайдер.

Описание классов игровых объектов:

1. Submarine – базовый абстрактный класс подводной лодки, имеет различные свойства отвечающие за характеристики лодки, так же методы передвижения и получения урона лодке;
2. BlueSubmarine – является наследником Submarine, лодка синего игрока;
3. RedSubmarine – является наследником Submarine, лодка красного игрока;
4. Rocket – базовый абстрактный класс ракеты, имеет различные свойства отвечающие за характеристики ракеты, так же метод движения ракеты;
5. FieryRocket – является наследником Rocket, огненная ракета. Данная ракеты имеет самую быструю скорость и скорость перезарядки;
6. IceRocket – является наследником Rocket, ледяная ракета. Данная ракеты имеет средние показатели по всем характеристикам;
7. AtomicRocket – является наследником Rocket, атомная ракета. Данная ракеты имеет наивысшие показатели по нанесению урона, но имеет самую медленную скорость и скорость перезарядки;
8. Destroyer – класс отвечающий за миноносец. Не имеет коллайдера, так как миноносец должен только передвигаться и сбрасывать мины;
9. Mina – класс отвечающий за мину. Имеет различные характеристики и метод передвижения по вертикали;
10. BonusGenerator – класс хранящий в себе информацию о бонусе, который случайным образом генерируется на сцене;
11. Background – класс отвечающий за задний фон сцены, не имеет коллайдера;
12. Border – класс отвечающий за границы сцены, границы имеют только коллайдер не отображается.

На рисунке 2.2 демонстрируется схему с игровыми объектами.

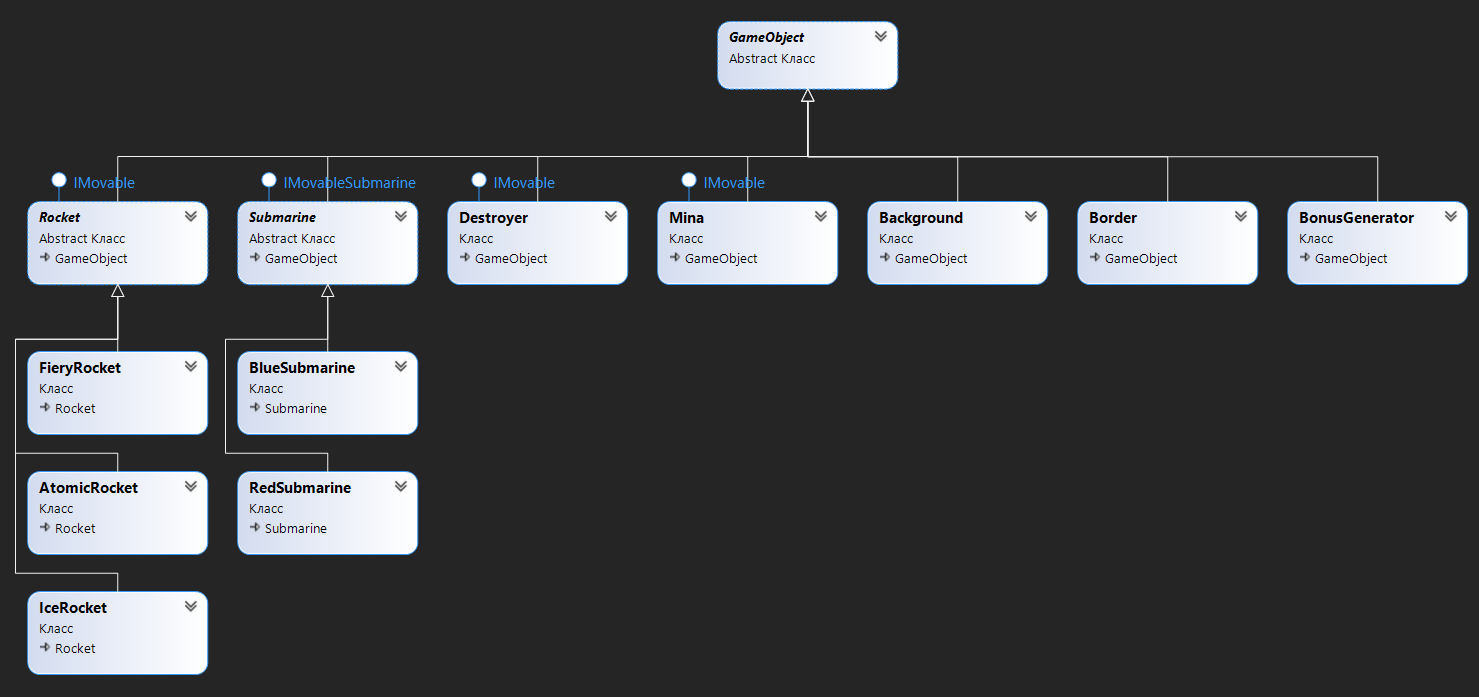


Рисунок 2.2 – Классы игровых объектов

Помимо игровых объектов библиотека GameLogic содержит различные классы игровой логики. Далее будут рассмотрены классы игровой логики.

Классы игровой логики:

– Timerr – класс описывающий таймер, содержит метод начала таймера, который принимает выполняющуюся функция и время, и содержит метод остановки таймера;

– Direction – перечисление которое содержит направления движения;

– IMovableSubmarine – интерфейс содержащий метод передвижения лодки по вертикали и горизонтали;

– IMovable – интерфейс содержащий метод движения для игровых объектов, которые могут передвигаться только по вертикали;

– RocketAmmunition – класс, который описывает амуницию подводной лодки;

– RocketType – перечисление содержащие различные типы ракет.

Так же для реализации сцены и меню игрового приложения будет создан проект Windows Forms, под названием BattleOfTheSubmarinesGames. Одним из главных классов игрового приложения является класс игровой сцены, который наследует класс GameWindow из графической библиотеки OpenTK. В классе нашей сцене мы создаем окно OpenTK и переопределяем самые необходимые методы. Данный класс не указан в игровом движке, так как он является сценой для игровых объектов, но при этом использует средства библиотеки OpenTK.

Основные методы класса сцены:

– OnRenderFrame – данный метод вызывается при рендеринге кадра и позволяет производить какие-либо графические изменения над нашими игровыми объектами;

– OnUpdateFrame – данный метод вызывается при обновлении кадра и используется для реализации игровой логики и изменение игровых объектов;

– OnLoad – данный метод вызывается после установления контекста OpenGL, но до входа в основной цикл игрового приложения.

На рисунке 2.3 демонстрируется схема с классами библиотеки GameLogic.

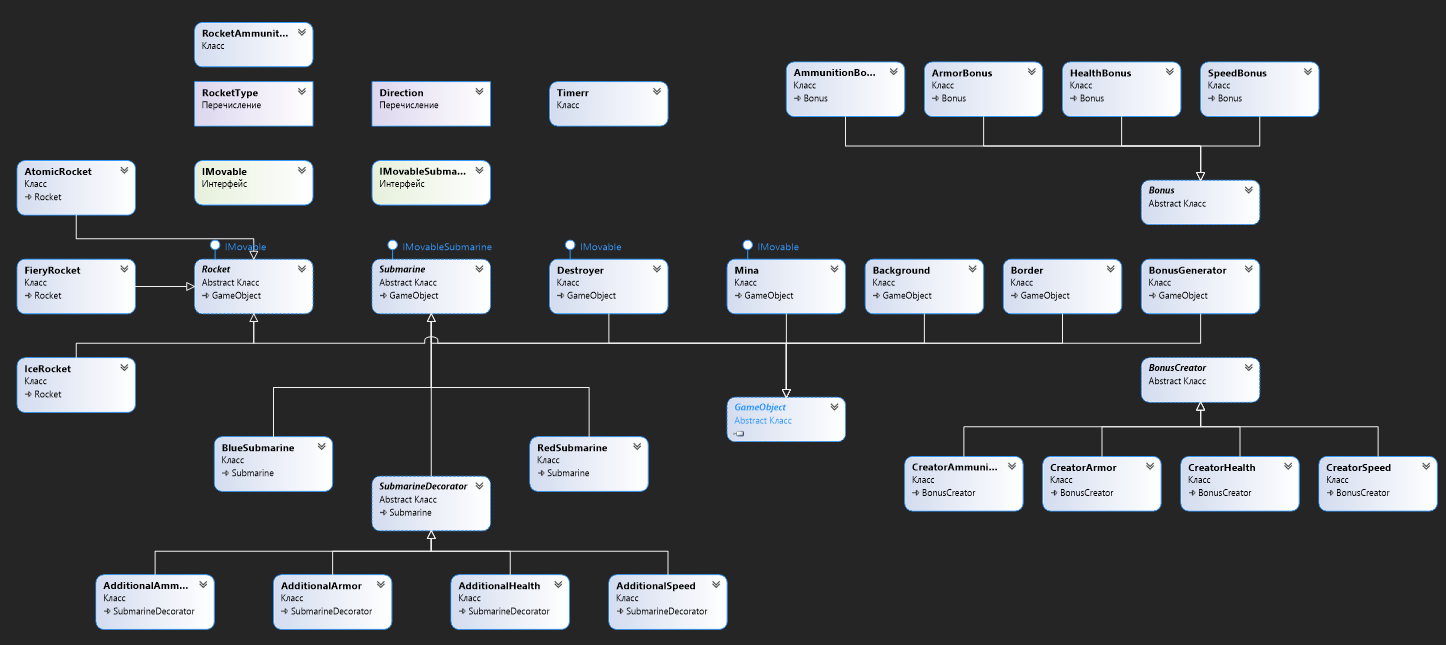


Рисунок 2.3 – схема с классами библиотеки GameLogic

1. **ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «СРАЖЕНИЕ ПОДВОДНЫХ ЛОДОК»**
   1. **Реализация игрового приложения «Сражение подводных лодок»**

Опишем алгоритм реализующий игровое приложение «Сражение подводных лодок».

После запуска игры на выполнение запускается проект BattleOfTheSubmarinesGames, игроки попадают в меню игры, которое содержит заставку и кнопку «Играть». После нажатия кнопки «Играть» на форме появляется конструктор снаряжения лодок. Снаряжение лодок реализовано через класс CustomizationSubmarines, данный класс создаёт элементы управления на форме для ввода характеристик лодок и считывает введённую информацию. После того как игроки снарядили лодки, в класс MainWindow передаётся информация о размере окна и характеристик лодок. В инициализаторе класса MainWindow загружаются все текстуры игры и сохраняются в словаре, для удобного доступа к ним. Далее создаются игровые объекты и добавляются в лист, содержащий объекты участвующие в игре. Запускаются таймера с генерацией бонусов и появлением миноносца. После того как все необходимые данные были подгружены начинают работать самые главыне методы класса MainWindow, отвечающие за рендеринг кадров и обновление кадров – это методы OnRenderFrame и OnUpdateFrame. Метод OnRenderFrame покадрово отображает игровые объекты, а в методе OnUpdateFrame взаимодействуют все методы классы, которые обеспечивают реализацию всего функционала игры.

Управление подводными лодками осуществляется посредством двух методов – это ControllingRedBluePlayer, который отвечает за управление синей лодкой и ControllingRedPlayer, который отвечает за управление красной лодки соответственно.

Главная механика подводной лодки – это стрельба. За снаряжения подводных лодок отвечает класс RocketAmmunition, он содержит информацию о ракетах лодок. Подводная лодка имеет определённое количество ракет разного типа, которые игроки выбрали в начале игры. Игрок может свободно переключаться между ракетами разного типа. На различные игровые объекты ракета реагирует по-разному. При попадании ракетой в объект типа Submarine, лодка получает урон равный характеристикам урона ракеты, но с условием что лодка не является отправителем ракеты. Данный механизм был реализован в классе Rocket, посредством поля отвечающего за тип отправителя лодки. При попадании ракетой в границы сцены ракета уничтожается, а при попадании ракеты с ракетой не происходит никаких действий.

Следующая важная механика в игре – это генерация бонусов. В начале игры запускается таймер, который выполняет метод CreateBonus. Метод CreateBonus генерирует случайный приз и располагает его в случайных координатах сцены. При столкновении лодки с бонусом лодка декорируется по свойствам, аналогичным типу класса бонуса. Если приз не подобрали, то с началом новой итерации таймера бонус будет удалён и создан новый.

Так же в игре существует механика появления миноносца и сбрасывание им мин. Данный механизм реализован посредством двух методов. Метод CreateDestroyer имеет свой таймер и выполняется циклически через определённое время, данный метод создаёт объекты типа Destroyer, которые отвечают за реализацию миноносца. Миноносец движется в случайные стороны, при обнаружении им лодки, выполняется метод CreateMina, который создаёт объекты типа Minа и скидывает их на лодку. При попадании миной в лодку, лодка получает урон равный характеристикам урона мины. Мину можно уничтожить ракетой.

Игра завершается в том случае, если у какой-либо лодки заканчивается здоровье. При завершении игры останавливаются все таймера, которые отвечают за создание объектов, со сцены удаляется проигравшая лодка и выводится надпись с названием победившей лодки.

**3.2** **Результаты тестирования игрового приложения «Сражение подводных лодок»**

Для тестирования игрового приложения был создан проект c модульными тестами *AppVerification*. Проект *AppVerification* тестирует классы библиотек *GameEngine* и *GameLogic*. Для тестирования используются статические методы класса *Assert* и классы модульного тестирования, которые находятся в пространстве имен *Microsoft.VisualStudio.Test-Tools.UnitTesting*. Данный проект позволяет проверить на корректность разработанные классы, их методы и свойства.

Модульные тесты библиотеки *AppVerification*:

– *AdditionalHealth\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением жизней лодки;

– *AdditionalArmor\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением брони лодки;

– *AdditionalSpeed\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением скорости лодки;

– *AdditionalAmmunition\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением амуниции лодки;

– *HealthBonus\_Test* – модульный тест класса *FactoryMethod\_Tests*, тестирующий создание бонуса с увеличением здоровья лодки;

– *ArmorBonus\_Test* – модульный тест класса *FactoryMethod\_Tests*, тестирующий создание бонуса с увеличением брони лодки;

– *SpeedBonus\_Test* – модульный тест класса *FactoryMethod\_Tests*, тестирующий создание бонуса с увеличением скорости лодки;

– *AmmunitionBonus\_Test* – модульный тест класса *FactoryMethod\_Tests*, тестирующий создание бонуса с увеличением амуниции лодки;

– *CollisionTrue\_Test\_SamePosition* – модульный тест класса *Collider\_Tests*, тестирующий столкновений объектов с одинаковыми позициями;

– *CollisionTrue\_Test\_RandomPosition* – модульный тест класса *Collider\_Tests*, тестирующий столкновений объектов со случайными позициями;

– *TakingDamage\_Test\_RandomValue* – модульный тест класса *Submarine\_Tests*, тестирующий метод *TakingDamage* класса *Submarine*, который отвечает за получение урона лодкой;

– *Shoot\_Test* – модульный тест класса *Submarine\_Tests*, тестирующий метод *Shoot* класса *Submarine*, который отвечает за выстрел лодки.

На рисунке 3.1 демонстрируется результат выполнения модельных тестов проекта *AppVerification.*

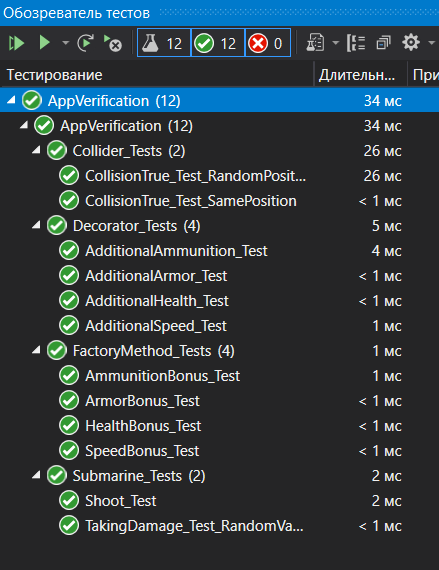


Рисунок 3.1 – результат выполнения тестов проекта AppVerification

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Программирование компьютерной графики. Современный *OpenGL* / Боресков А.В. – ДМК-Пресс: 2019. – 373с.
2. Паттерны проектирования в *C*# и .*NET* – 2017. – Режим доступа: https://metanit.com/sharp/patterns/. – Дата доступа: 04.03.2021.
3. *CLR* *via* *C*#. Программирование на платформе *Microsoft* .*NET* *Framework* 4.5 на языке *C*#/. СПб.:Питер, 2020. – 896c.
4. *OpenGL SuperBible* / *Graham Sellers, Richard S. Wright Jr., Nicholas* *Haemel* – *Addison*-*Wesley*: 2015 –105c.