**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc40809808)

[1 СРЕДСТВА СОЗДАНИЯ ИГРОВОГО ПРИЛОЖЕНИЯ 5](#_Toc40809809)

[1.1 Графическая библиотека OpenGL 5](#_Toc40809811)

[1.2 Использование OpenGl в языке программирования С# 6](#_Toc40809812)

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

[1.4 Использование OpenTK в приложениях Windows Form 8](#_Toc40809814)

[Список используемых источников](#_Toc40809823) ……………………………………………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 мы 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 мы будем использовать графическую библиотеку 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 мы освоим и применим 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.

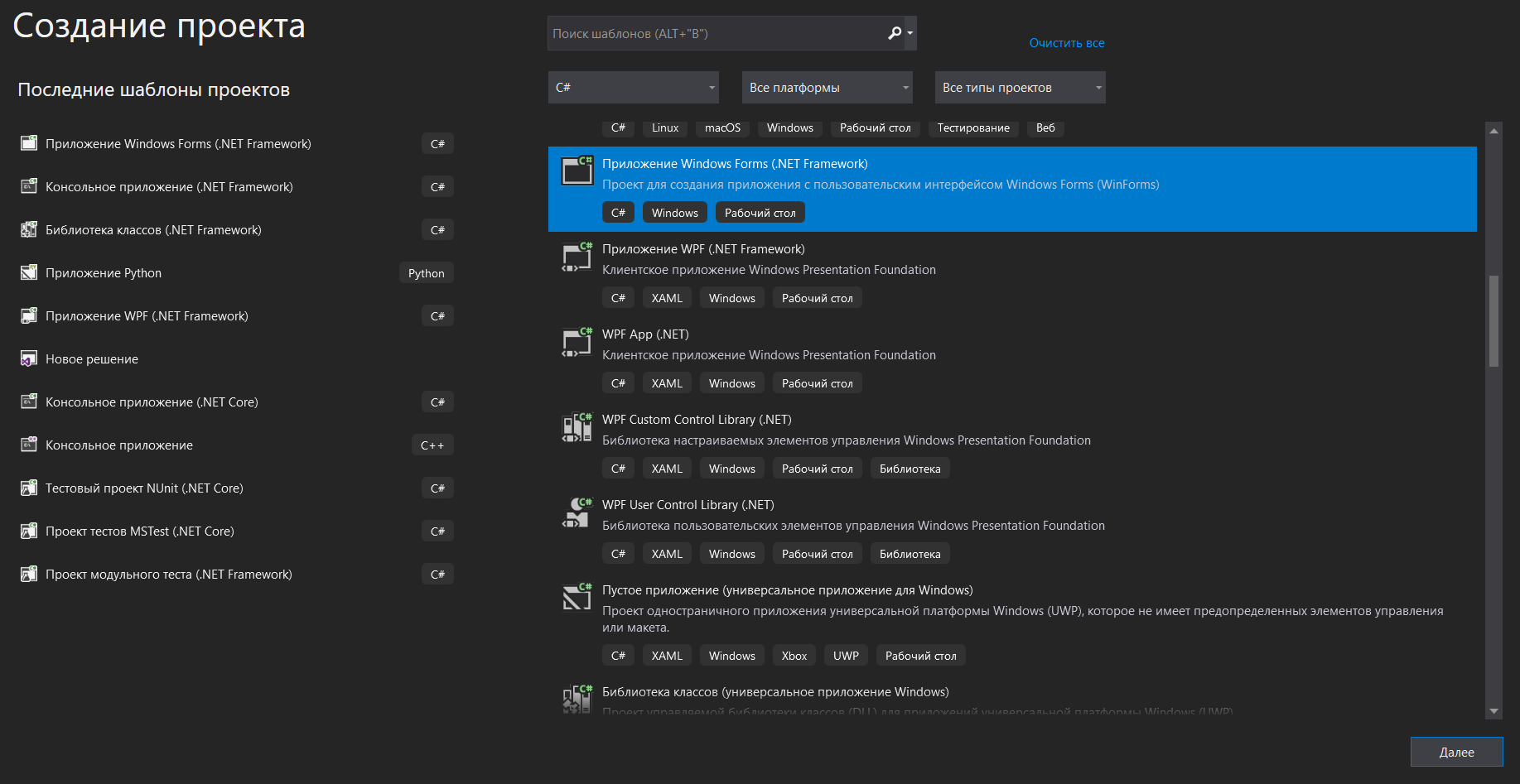


Рисунок 1.1 – Создание приложения Windows Forms

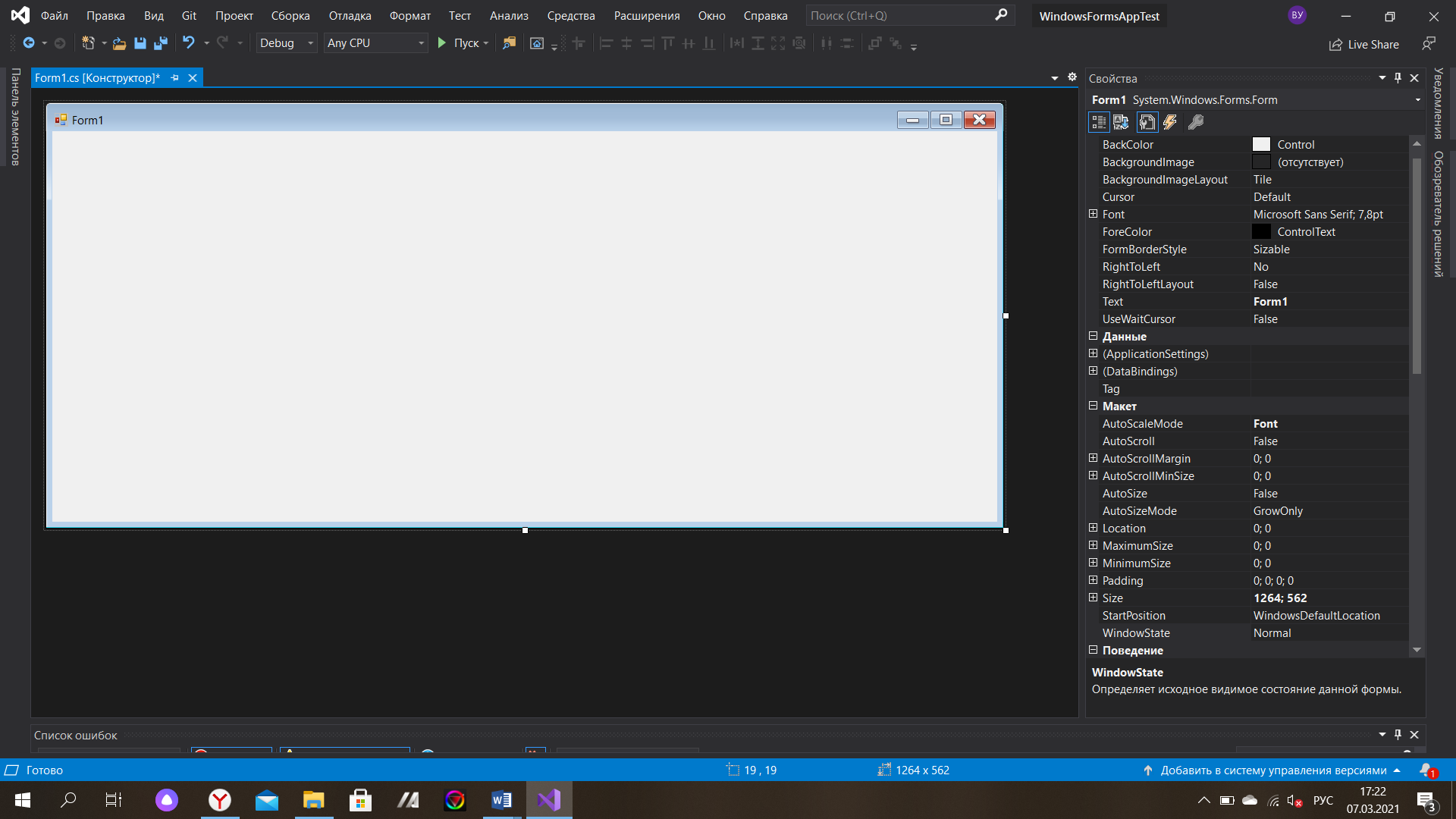


Рисунок 1.2 – Интерфейс работы с приложением

**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;

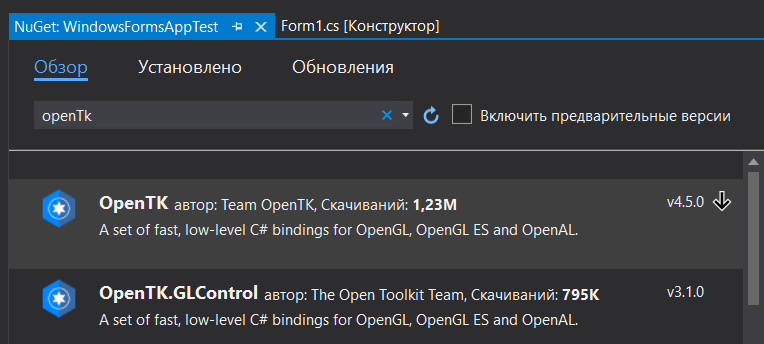


Рисунок 1.3 – Диспетчер пакетов NuGet

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

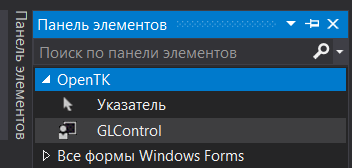


Рисунок 1.4 – Добавление компонент GLControl

1. Добавим GLControl на форму;

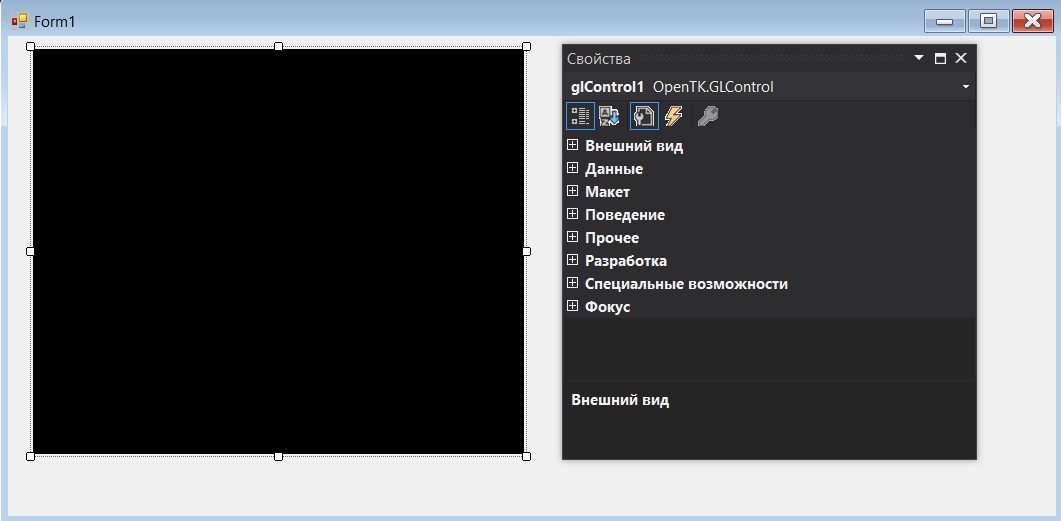


Рисунок 1.5 – Компонент GLControl

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. **Архитектура игрового приложения**

**2.1 Исходные данные игрового приложения**

Необходимо разработать приложение, реализующее многопользовательскую игру «Сражение подводных лодок». Каждый игрок управляет подводной лодкой, которая может стрелять различными видами торпед. Каждая лодка характеризуется скоростью плавания, бронёй и жизненной энергией. Перед началом игры каждый игрок на одинаковое количество очков может сконфигурировать свою подводную лодку: выбрать максимальное значение скорости, максимальное значение брони, жизненной энергии, количество торпед, одновременно находящихся на борту. Перед началом игры на каждую подводную лодку может быть загружено определённое число

торпед различных типов. Торпеды отличаются скоростью перемещения, повреждающим эффектом, скорострельностью. Во время игры на экране случайным образом появляются призы, позволяющие пополнить боезапас, броню, жизненную энергию, увеличить/уменьшить на определённое число процентов скорость. Кроме того, периодически должен появляться эсминец и сбрасывать подводные мины, столкновение с которыми наносить повреждения лодкам. Мины уничтожаются торпедами, либо достигая дна. Проигрывает тот игрок, чья подводная лодка погибает. Для реализации использовать

средства языка программирования C# Windows form и шаблоны проектирования: «фабричный метод» – для реализации генерации призов: «декоратор» – для задания характеристик подводных лодок. Для отображения объектов на игровом поле использовать спрайтовую графику и средства OpenGL.

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

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 итогов игры.

Для реализации игры будем использовать собственный игровой движок.

Основные задачи игрового движка:

1. Работа со спрайтовой графикой;
2. Работа с 2D анимацией;
3. Физика перемещение и взаимодействие игровых объектов;
4. Реализация и интегрирование в игру паттерна «декоратор»;
5. Реализация и интегрирование в игру паттерна «фабричный метод».

**2.3 Структура игрового приложения**

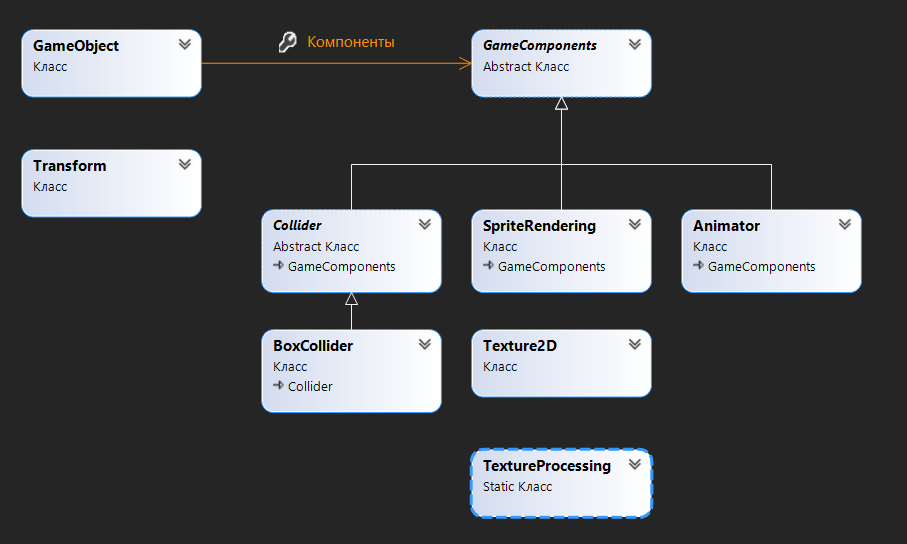
Рассмотрим схему классов нашего игрового движка.

Рисунок 2.1 – UML схема игрового движка

Подробно рассмотрим классы игрового движка.

Одним из главных классов игрового приложения является класс нашей 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:

1. OnRenderFrame – данный метод вызывается при рендеринге кадра и позволяет производить какие-либо графические изменения над нашими игровыми объектами.
2. OnUpdateFrame – данный метод вызывается при обновлении кадра и используется для реализации игровой логики и изменение игровых объектов.
3. OnLoad – данный метод вызывается после установления контекста OpenGL, но до входа в основной цикл игрового приложения.

Классы игрового движка:

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.

Возможные компоненты:

* Класс SpriteRenderer является производным от GameComponent и отвечает за отображение объекта на сцене;
* Класс Collider является производным от GameComponent и содержит в себе коллайдер игрового объекта
* Класс Animator является производным от GameComponent и отвечает за анимацию игровых объектов.

1. Класс Transform содержится в игровом объекте GameObject как обязательный параметр, так как все объекты должны иметь положение, вращение и масштаб.

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 её свойствами.

Теперь рассмотрим основные объекты нашей игры.

Игровые объекты:

1. Подводная лодка, которой будут управлять игроки.
2. Различного вида ракеты, которыми будет стрелять подводная лодка. В зависимости от вида наносит различные повреждения.
3. Различного вида призы, каждый приз будет увеличивать различные свойства подводной лодки.
4. Миноносец который будет периодически появляться и сбрасывать на игроков мины.

Для реализации классов танков и ракет используется наследование. В случае с подводной лодкой (Submarine) мы не используем наследование, так как подводные лодки не отличаются характеристиками, по желанию только спрайтами. Для ракет мы используем наследование, так как ракеты отличаются наносимыми повреждениями в зависимости от подтипа ракеты.

На рисунке 2.2 отображена иерархия игровых объектов.

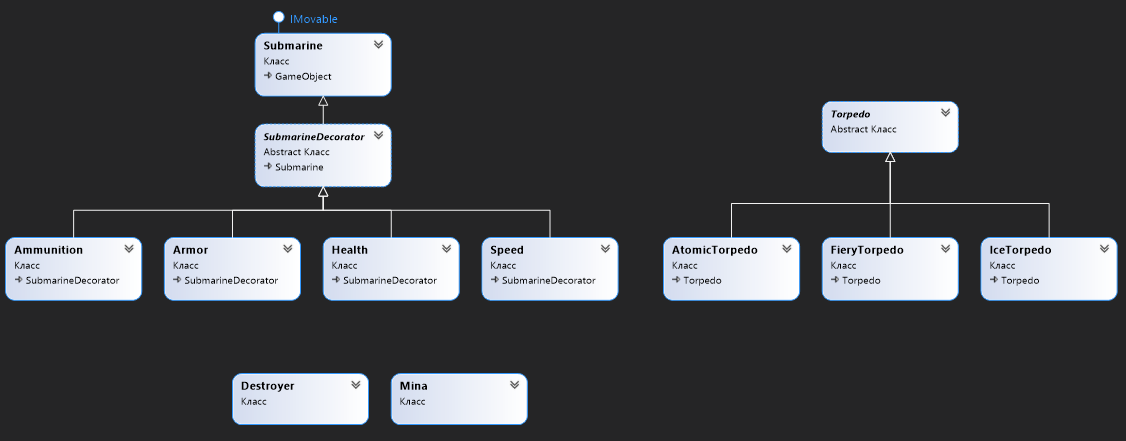


Рисунок 2.2 – Иерархия игровых объектов

Рассмотри паттерны программирования, которые будут использоваться в нашем игровом приложении.

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 представлена UML схема реализации паттерна «декоратор» для поставленной задачи.

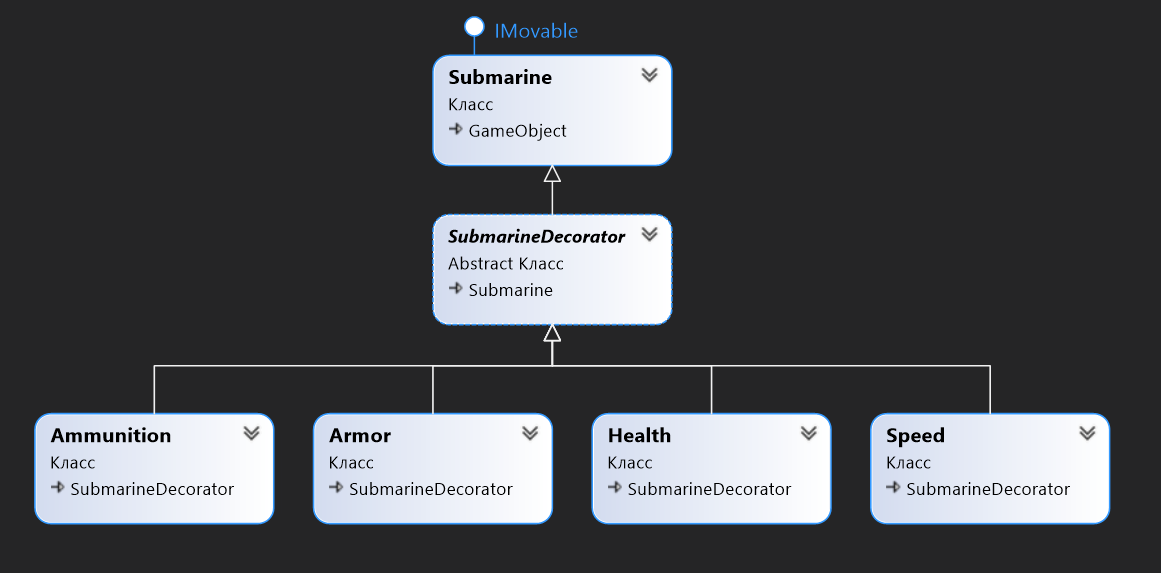


Рисунок 2.3 – UML схема паттерна «Декоратор»

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. Ammunition: дополнительный функционал для класса Submarine в виде пополнения боеприпасов.
4. Armor: дополнительный функционал для класса Submarine в виде увеличения брони.
5. Health: дополнительный функционал для класса Submarine в виде пополнения здоровья.
6. Speed: 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 представлена UML схема реализации паттерна «Фабричный метод» для поставленной задачи.

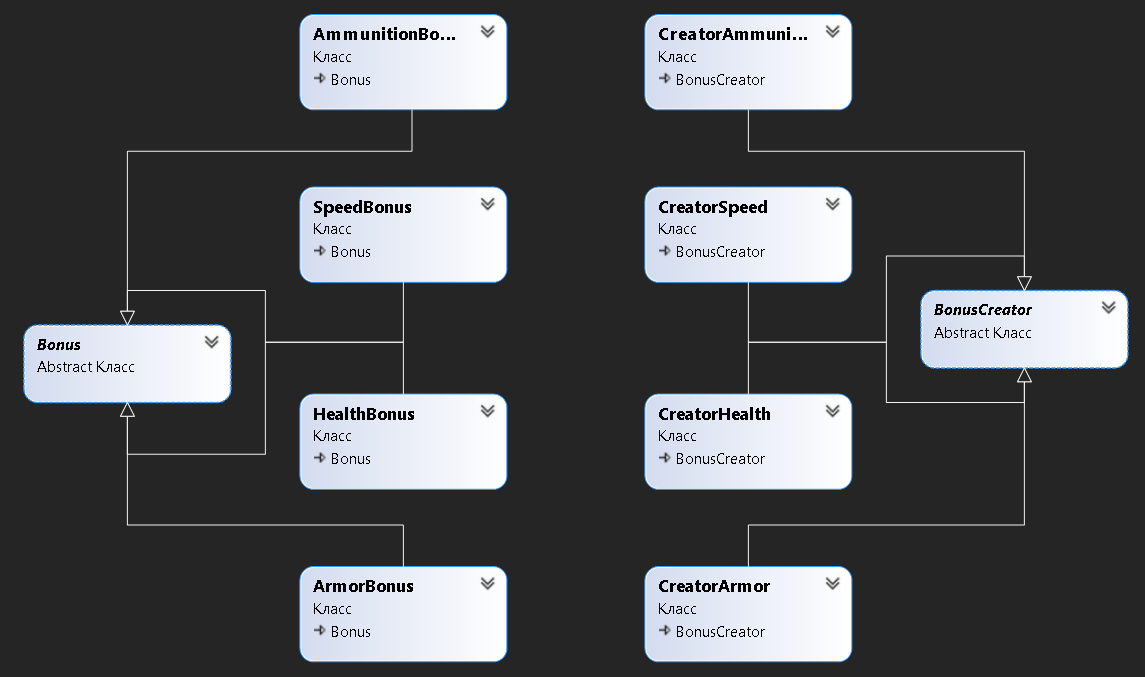


Рисунок 2.4 – UML схема паттерна «Фабричный метод»

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.

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