**СОДЕРЖАНИЕ**

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

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

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

1.2 Интерфейс программирования приложений Windows Form...............6

1.3 Использование OpenGl в языке программирования С#.......................7

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

2 ОБЗОР АРХИТЕКТУРЫ И СТРУКТУРЫ ИГРОВОГО ПРИЛОЖЕНИЯ «СРАЖЕНИЕ ПОДВОДНЫХ ЛОДОК…………………………………...........10

2.1 Архитектура игрового приложения………………………………….10

2.2 Cтруктура игрового приложения ………………….……..……….….11

2.3 Обзор взамодействия классов игрового приложения……………….13

3 ТЕСТИРОВАНИЕ И ЭКСПЛУАТАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «СРАЖЕНИЕ ПОДВОДНЫХ ЛОДОК» 19

3.1 Алгоритм работы игрового приложения…………………………….19

3.2 Тестирование игрового приложения…………....……………………20

3.3 Эксплуатация игрового приложения……... …………………………22

ЗАКЛЮЧЕНИЕ ..………….………………………………………………..........27

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ…………..……………….…...28

ПРИЛОЖЕНИЕ А Листинг программы «Сражение подводных лодок».……29

ПРИЛОЖЕНИЕ Б Архитектура паттерна «Декоратор» 75

ПРИЛОЖЕНИЕ В Руководство программиста 76

ПРИЛОЖЕНИЕ Г Руководство системного программиста 77

ПРИЛОЖЕНИЕ Д Руководство пользователя…………………………………78

**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.

Компьютерная графика нашла широкое распространение и применение в повседневной жизни. Учёные используют компьютерную графику для анализа результатов моделирования. Инженеры и архитекторы используют компьютерную графику для создания виртуальных моделей. В последние годы широкое распространение получили также компьютерные игры, максимально использующие трёхмерную графику для создания виртуальных миров.

Поэтому далее рассмотрим более сложную и не всем знакомую часть игр, это их разработка.

При реализации данного проекта будет использоваться графическая библиотека OpenGl и средства языка программирования C# Windows Form для разработки игровых приложений, различные методы и технологии создания игр.

Исходя из вышеперечисленного сделан вывод, что работа над данным проектом является весьма актуальной в нынешнее время. В результате разработки игрового приложения будет получен серьёзный багаж 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 игровом приложении будет использоваться компьютерная графика 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 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 1.1–1.2 демонстрируется создание приложения в Windows Forms.

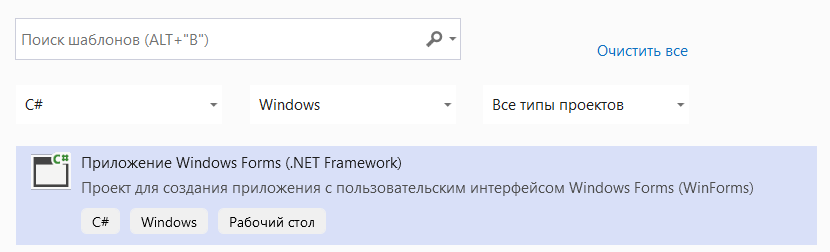


Рисунок 1.1 – Создание приложения Windows Forms

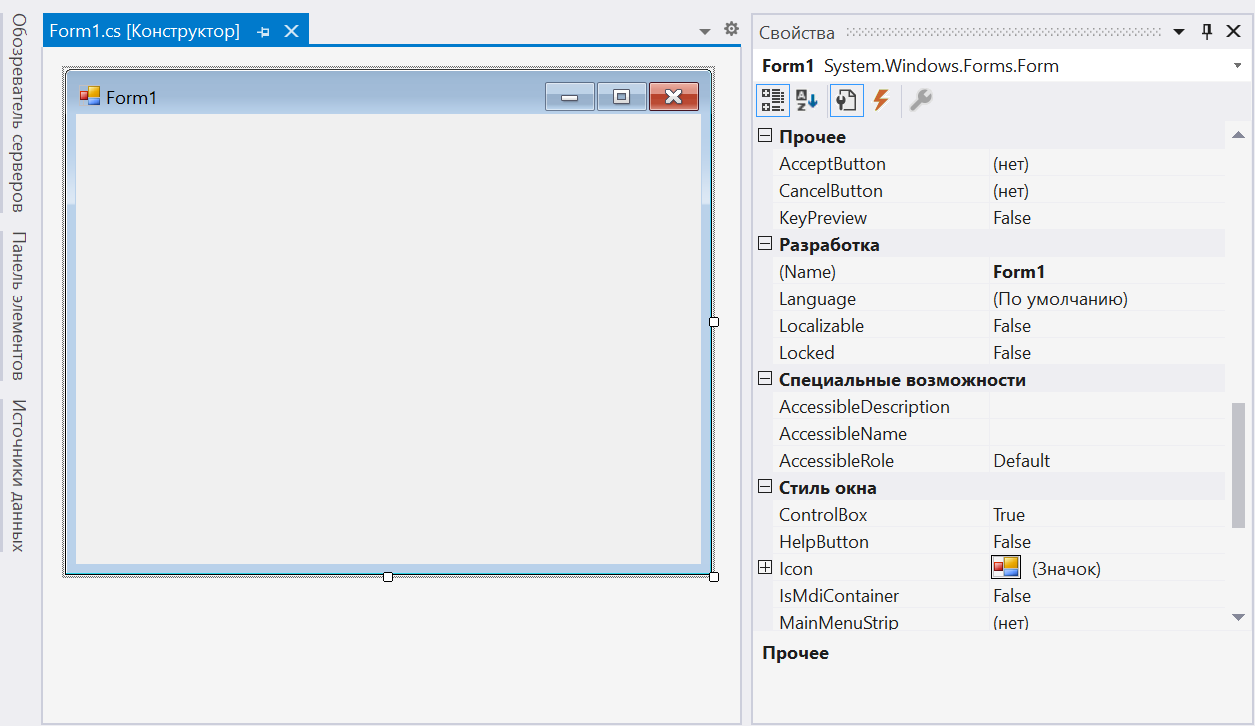


Рисунок 1.2 – Интерфейс работы с приложением

**1.3 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 графическим 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*.

**1.4 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 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.

Этапы подключение OpenTK в приложение Windows Forms:

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

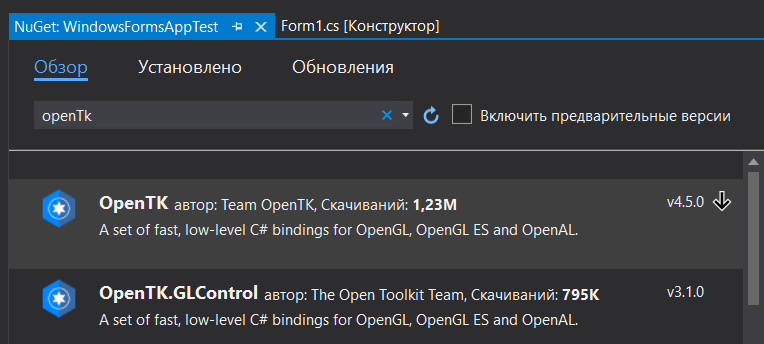


Рисунок 1.3 – Диспетчер пакетов NuGet

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

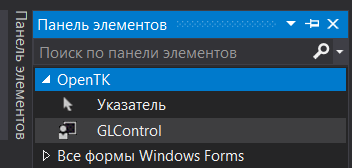


Рисунок 1.4 – Добавление компонента GLControl

1. Добавим GLControl на форму.

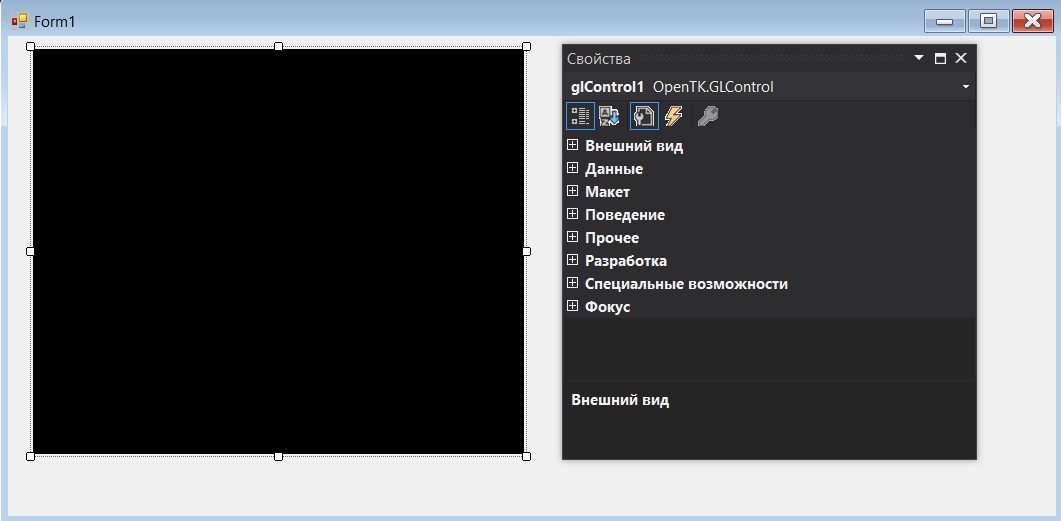


Рисунок 1.5 – Компонент GLControl

В данной главе рассказывалось 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 со всеми нужными технологиями.

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

**2 ОБЗОР АРХИТЕКТУРЫ И СТРУКТУРЫ ИГРОВОГО ПРИЛОЖЕНИЯ «СРАЖЕНИЕ ПОДВОДНЫХ ЛОДОК**

**2.1 Архитектура игрового приложения**

В процессе проектирования архитектуры игрового приложения был ис-пользован процесс декомпозиции. Задача была разбита на подзадачи, кторые проектировались и разрабатывались, а после были объединены в общее реше-ние. Каждый элемент декомпозиции является уникальным, что позволило сформировать и отличить составляющие объекта одну от другой. Элементы, полученные в результате декомпозиции, связаны с функциями отдельных подзадач единой задачей.

На рисунке 2.1 демонстрируется обобщенная функциональная схема приложения, описывающая последовательность работы функций приложения.

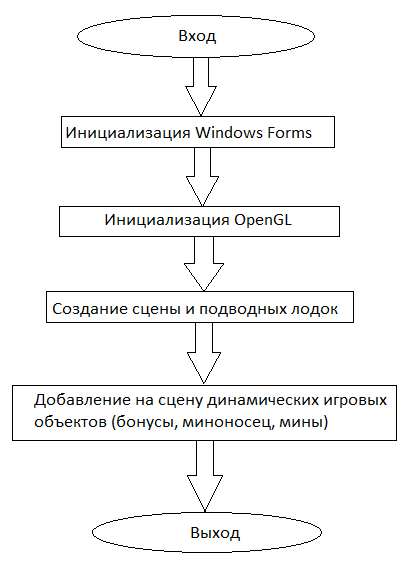


Рисунок 2.1 – Схема паттерна «Декоратор»

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.

**2.2 Структура 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;

– завершение игры, вывод победителя.

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. Приведённые выше паттерны так же будут реализованы в данном проекте.

Далее будет создан 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.

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. Перечисленные паттерны так же будут реализованы в библиотеке GameLogic.

Паттерн «Декоратор»– 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 2.2 демонстрируется схема реализации паттерна «Декоратор».

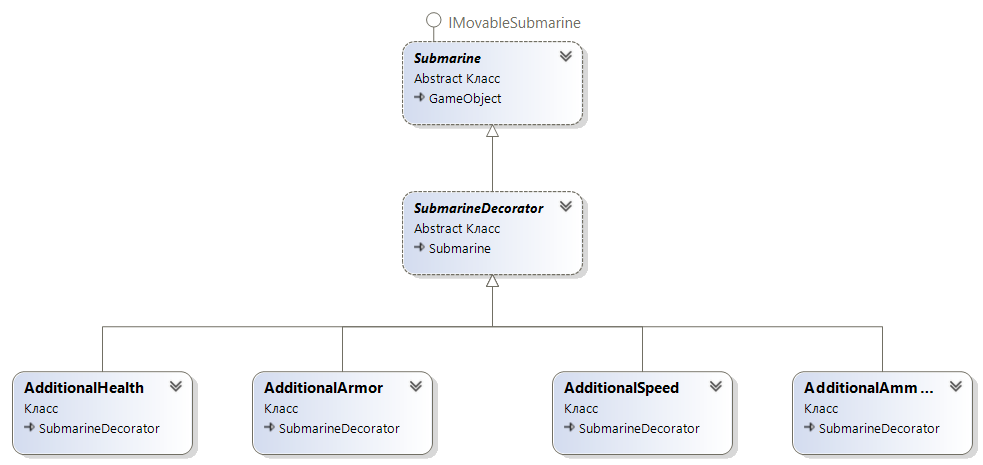


Рисунок 2.2 – Схема паттерна «Декоратор»

Паттерна «Фабричный метод» – 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 2.3 демонстрируется схема реализации паттерна «Фабричный метод».

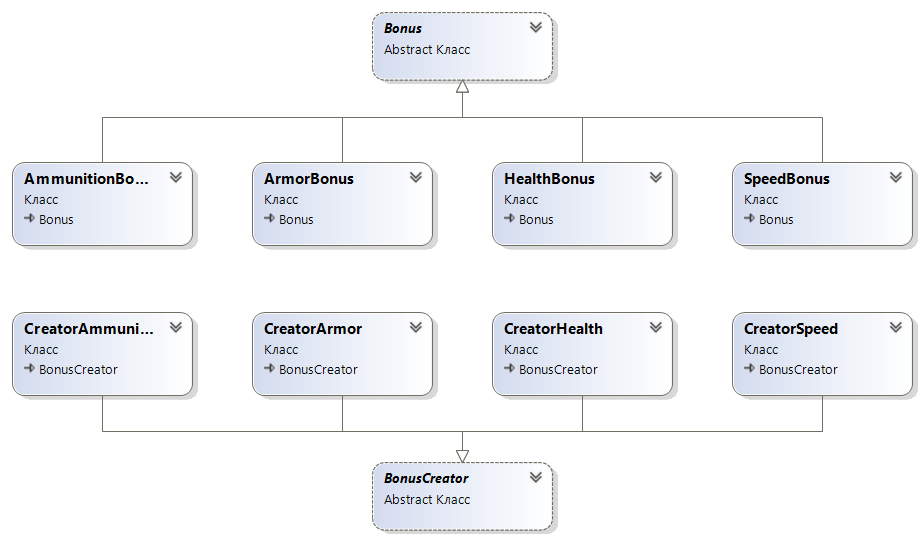


Рисунок 2.3 – Схема паттерна «Фабричный метод»

**2.3 Обзор взамодействия классов игрового приложения**

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, который включает в себя классы наследуемые от типа 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 TextureProcessing создаёт текстуру на основы добавленного изображения и выводит класс Texture2D со всеми данными о текстуре;
5. SpriteRendering – класс, который имеет статический метод отображения спрайтов.

На рисунке 2.4 демонстрируется схема игрового движка.

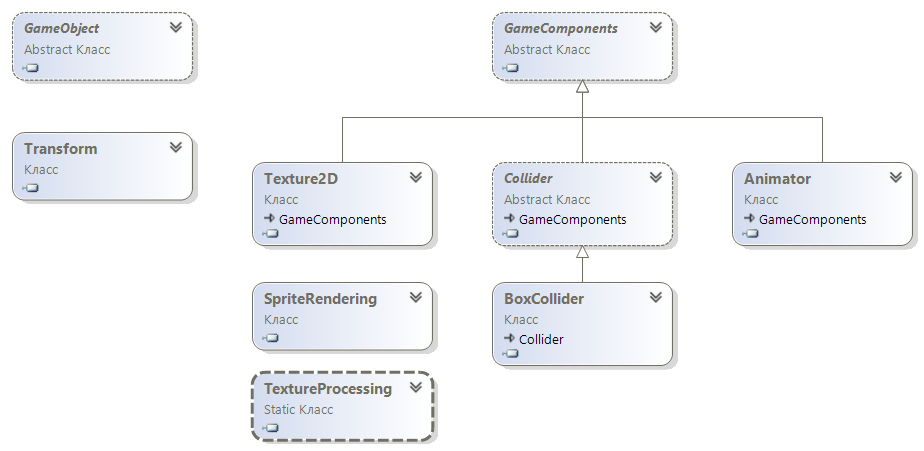


Рисунок 2.4 – Схема игрового движка

Теперь рассмотрим библиотеку GameLogic и опишем взаимодействие классов библиотеки.

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. Перечисленные паттерны так же будут реализованы в библиотеке GameLogic.

Описание классов участвующих в реализации паттерна «Декоратор»:

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 дополнительный функционал, которым 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.

Описание классов участвующих в реализации паттерна «Фабричный метод»:

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.

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.

На рисунке 2.5 демонстрируется схему с игровыми объектами.

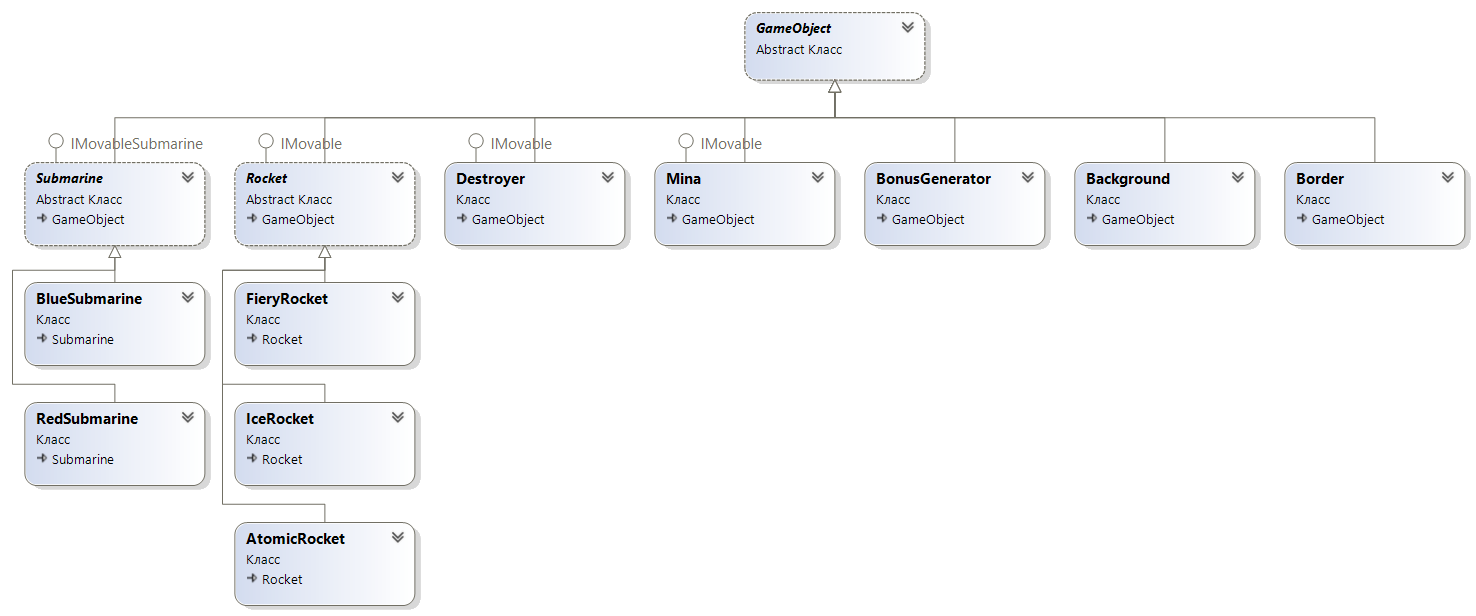


Рисунок 2.5 – Игровые объекты

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.

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

На рисунке 2.6 демонстрируется схема с классами библиотеки GameLogic.

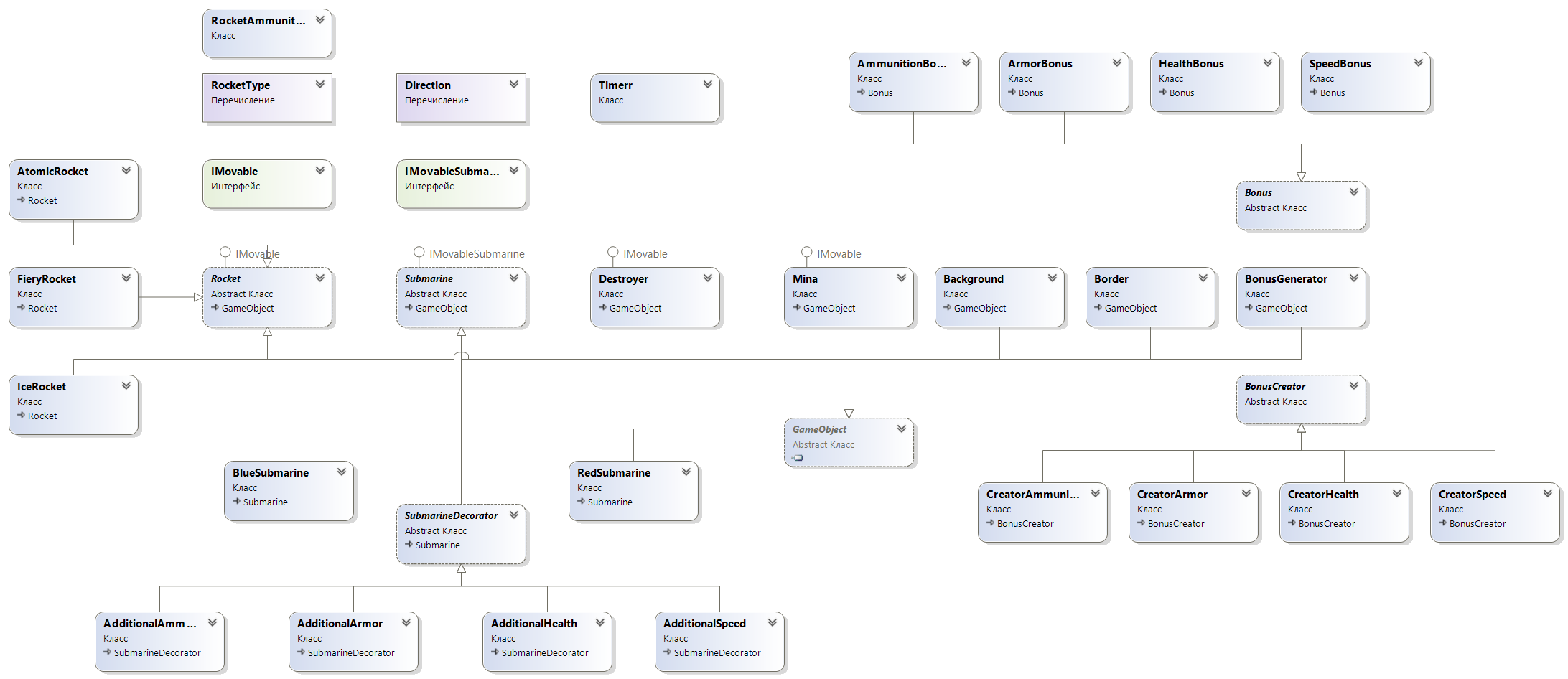


Рисунок 2.6 – схема с классами библиотеки GameLogic

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 главной сцены будет создано окно OpenTK и переопределены 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.

**3** **ТЕСТИРОВАНИЕ И ЭКСПЛУАТАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «СРАЖЕНИЕ ПОДВОДНЫХ ЛОДОК»**

**3.1 Алгоритм работы игрового приложения**

Рассмотрим 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 «Начать игру». После нажатия кнопки «Начать игру» 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 Тестирование игрового приложения**

Для тестирования уже созданного функционала 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*.

Библиотека *AppVerification* содержит 12 модульных тестов, которые тестируют самые сложные и важные по функционалу классы. Модульные тесты тестируют все классы учавствующие в реализации паттернов «Декоратор» и «Фабричный метод», так же классы подводных лодок и классы отвечающие за отслеживание столкновений игровых объектов.

Приведённые выше модульные тесты позволяют проверить на корректность разработанные классы, их методы и свойства.

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

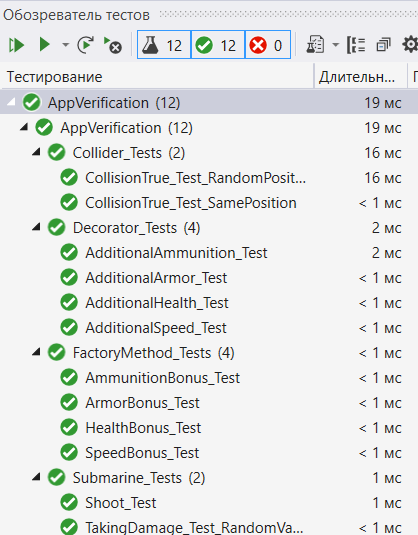


Рисунок 3.1 – Результат выполнения модульных тестов

Чтобы определить, какая часть кода проекта в действительности тестируется модульными тестами был проведён анализ покрытия кода в *Visual* *Studio*.

На рисунке 3.2 демонстрируется результат анализа покрытия кода.

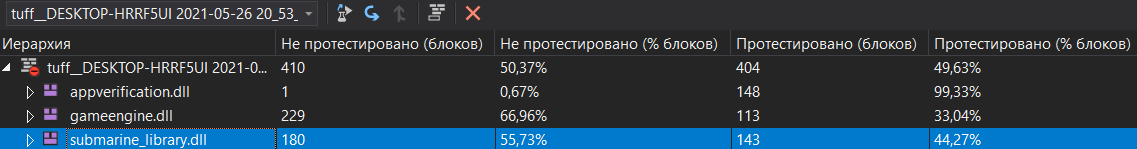


Рисунок 3.2 – Результат анализа покрытия кода

Описание модульных тестов библиотеки *AppVerification*:

– *AdditionalHealth\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением жизней лодки. Тестирование реализуется путём преобразования лодки в лодку с декорированными жизнями;

– *AdditionalArmor\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением брони лодки. Тестирование реализуется путём преобразования лодки в лодку с декорированной бронёй;

– *AdditionalSpeed\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением скорости лодки. Тестирование реализуется путём преобразования лодки в лодку с декорированной скоростью;

– *AdditionalAmmunition\_Test* – модульный тест класса *Decorator*\_*Tests*, тестирующий класс декоратора с изменением амуниции лодки. Тестирование реализуется путём преобразования лодки в лодку с декорированной амуницией;

– *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. Тестирование реализуется путём создания двух лодко с одинаковыми позициями и проверкой на столкновение;

– *CollisionTrue\_Test\_RandomPosition* – модульный тест класса *Collider\_Tests*, тестирующий столкновений объектов со случайными позициями. Тестирование реализуется путём создания двух лодко со случайными позициями и проверкой на столкновение;

– *TakingDamage\_Test\_RandomValue* – модульный тест класса *Submarine\_Tests*, тестирующий метод *TakingDamage* класса *Submarine*, который отвечает за получение урона лодкой. Тестирование реализуется путём нанесения лодке случайного урона и проверки на то, чтобы лодка правильно принимала урон;

– *Shoot\_Test* – модульный тест класса *Submarine\_Tests*, тестирующий метод *Shoot* класса *Submarine*, который отвечает за выстрел лодки. Тестирование реализуется путём выстрела лодкой случайной ракеты и сравнение количества ракет данного типа до выстрела и после.

**3.3 Эксплуатация игрового приложения**

После запуска игрового приложения появляется форма с начальным меню игры, которая содержит заставку, название игры и кнопку «Начать игру».

На рисунке 3.3 демонстрируется начальное меню игрового приложения.



Рисунок 3.3 – Начальное меню игрового приложения

После нажатия кнопки «Начать играть» на форме появляется конструктор снаряжения лодок.

На рисунке 3.4 демонстрируется конструктор снаряжения лодок.



Рисунок 3.4 – Конструктор снаряжения лодок

После того как двое игроков снарядили свои лодки, происходит проверка введённых данный. Если данный были введены корректно, то игровое приложение запускается.

В начале игры создаются только основные игровые объекты, включая подводные лодки. После истечения определённого времени начинают добавляться динамические объекты игры – это бонусы, миноносец и мины. Или же ракеты, которые могут запускать сами игроки.

Цель игры – уничтожить вражескую подводную лодку. Для уничтожения лодки, игроки используют ракеты разного типа. Так же нанести урон подводной лодке может миноносец, скинув на неё мину, миноносец появляется каждые 10 секунд. Чтобы получить преимущество перед соперником игрок должен подпирать бонусы, которые генерируются каждые 12 секунд.

Подводная лодка первого игрока окрашена синим цветом, а подводная лодка второго игрока окрашен в красный цвет соответственно. Каждый игрок имеет одинаковые характеристик при старте и различные клавиши для управления лодкой, характеристики и раскладка клавиатуры будут указаны ниже.

На рисунке 3.5 демонстрируется сцена игрового приложения в процессе игры.



Рисунок 3.5 – Сцена игрового приложения в процессе игры

Начальные характеристики подводной лодки:

–жизни – 30 единиц;

– броня– 30 единиц;

– скорость – 18;

– амуниция – 15 ракет.

Раскладка клавиатуры, отвечающая за управление подводными лодками демонстрируется в таблице 3.1.

Таблица 3.1 – Раскладка клавиатуры

|  |  |  |
| --- | --- | --- |
|  | Синяя лодка | Красная лодка |
| Движение вверх | W | Up |
| Движение вниз | S | Down |
| Движение влево | A | Left |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Движение вправо | D | Right |
| Переключение на огненную ракету | Number1 | Keypad1 |
| Переключение на ледяную ракету | Number2 | Keypad2 |
| Переключение на атомную ракету | Number3 | Keypad2 |
| Выстрел | Space | KeypadEnter |

На рисунке 3.6 демонстрируется вид подводных лодок для двух игроков.



Рисунок 3.6 – Вид подводных лодок

Типы ракет:

– огненная ракета – данная ракета наносит 45 ед. урона по жизням, 35 ед. урона по броне, имеет скорость 40 и скорострельность 750 мс;

– ледяная ракета – данная ракета наносит 60 ед. урона по жизням, 55 ед. урона по броне, имеет скорость 30 и скорострельность 1000 мс;

– атомная ракета – данная ракета наносит 80 ед. урона по жизням, 60 ед. урона по броне, имеет скорость 25 и скорострельность 1250 мс.

На рисунке 3.7 демонстрируются ракеты различного типа.



Рисунок 3.7 – Ракеты различного типа

Типы бонусов:

– бонус здоровья – добавляет к здоровью 20 единиц;

– бонус брони – добавляет к броне 20 единиц;

– бонус скорости – добавляет к скорости 2 единицы;

– бонус амуниции – добавляет к каждому типу ракет по одной ракете.

На рисунке 3.8 демонстрируются бонусы различного типа.



Рисунок 3.8 – Бонусы различного типа

Характеристики игрового объекта мина:

– урон по жизням – 30 единиц;

– урон по броне – 30 единиц;

– скорость – 20.

На рисунке 3.9 демонстрируется игровой объект мина.



Рисунок 3.9 – Игровой объект мина

Игра завершается в том случае, елсли один из игроков уничтожает вражескую лодку. При завершении игры останавливаются все таймера, которые отвечают за создание объектов, со сцены удаляется проигравшая лодка и выводится надпись с названием победившей лодки.

На рисунке 3.10 демонстрируется процесс завершения игры в результате уничтожения одной из подводных лодок.



Рисунок 3.10 – Завершение игры

**ЗАКЛЮЧЕНИЕ**

За время работы над проектом было изучено и реализовано на практике множество технологий, методов и подходов программирования.

Для того чтобы изучит все нужные технология была подобрана литература в области разработки игровых приложений, использования OpenGL в приложениях Windows Forms и различных паттернов программирования.

При реализации игрового приложения использованы средства языка программирования C# Windows Forms и шаблоны проектирования «декоратор» и «фабричный метод». Для отображения объектов на игровой сцене использовалась спрайтовая графика и средства библиотека OpenTK.

В ходе разработки игрового приложения решены следующие задачи:

– Произведён обзор литературы и средств по программно-техническому обеспечению для разработки игровых приложения;

– Разработан игровой движок и игровая сцена, включающая в себя весь функционал игры «Сражение подводных лодок».

– Реализованы шаблоны проектирования «декоратор» для декорирования характеристик подводной лодки, и паттерн «фабричный метод» для генерации бонусов;

– Проведено тестирование библиотек игрового приложения, посредством создания модульных тестов;

– Предоставлена вся информация о подходах и методах создания игрового приложения «Сражение подводных лодок»;

– Предоставлены данные для эксплуатации игрового приложения.

В результате работы было реализовано многопользовательское игровое приложение «Сражение подводных лодок». В игровом приложении игрокам необходимо уничтожить вражескую подводную лодку. Для этого лодки игроков снаряжены ракетами различного вида. Для того чтобы игра была более динамичная и захватывающая в игру были добавлены различные бонусы и миноносец, который атакует игроков.

Данная игра подойдет игрокам любой возрастной категории. Игра имеет понятное управление и простую, но достаточно приятную спрайтовую графику.

Больше всего игра понравится людям, любящим поиграть в многопользовательские игры с динамичными и яркими баталиями, так же людям, которым по душе соревновательные игры с возможностью стратегически мыслить.

Из интересных особенностей игры можно выделить динамический и захватывающий сюжет, так же интересный механизм, позволяющий игрокам переключаться между ракетами разных типов.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Программирование компьютерной графики. Современный OpenGL / Боресков А.В. – ДМК-Пресс, 2019. – 373с.
2. Паттерны проектирования в C# и .NET[Электронный ресурс]. – 2017. – Режим доступа: https://metanit.com/sharp/patterns/. – Дата доступа: 04.03.2021.
3. LearnOpenTK [Электронный ресурс]. – Режим доступа: https://opentk.net/learn/index.html. – Дата доступа: 28.03.2021.
4. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. – СПб: Питер, 2020. – 896c.
5. OpenGL SuperBible / Graham Sellers, Richard S. Wright Jr., Nicholas Haemel. – Addison-Wesley, 2015. –105c.

6. LearnOpenGL [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/310790/. – Дата доступа: 02.04.2021.

7. Factory Method C# [Электронный ресурс]. – Режим доступа: https://shwanoff.ru/factory-method/. – Дата доступа: 10.04.2021.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы «Сражение подводных лодок»**

Листинг класса Program.cs:

using System;

using System.Windows.Forms;

namespace BattleOfTheSubmarinesGames

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new CustomizationSubmarines(1920, 1080));

}

}

}

Листинг класса CustomizationSubmarines.cs:

using System;

using System.Drawing;

using System.Windows.Forms;

using GameLogic;

namespace BattleOfTheSubmarinesGames

{

/// <summary>

/// Игровое меню.

/// </summary>

public partial class CustomizationSubmarines : Form

{

RocketAmmunition blueAmmunition;

RocketAmmunition redAmmunition;

int ammunitionCount;

int submarines = 0;

/// <summary>

/// Инициализатор игрового меню.

/// </summary>

/// <param name="width"> Ширина. </param>

/// <param name="height"> Высота. </param>

public CustomizationSubmarines(int width, int height)

{

InitializeComponent();

Width = width;

Height = height;

blueAmmunition = new RocketAmmunition();

redAmmunition = new RocketAmmunition();

var submarine = new BlueSubmarine(new RocketAmmunition());

ammunitionCount = submarine.AmmunitionCount;

CangePosition(NameGame, (Width - NameGame.Size.Width) / 2, (Height / 2) - 200 );

CangePosition(StartGame\_Button, (Width - StartGame\_Button.Size.Width) / 2, Height / 2);

CangePosition(Customization\_panel, (Width - Customization\_panel.Size.Width) / 2, (Height - Customization\_panel.Size.Width) / 2);

}

/// <summary>

/// Открытие панели с настройкой лодки.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StartGame\_Button\_Click(object sender, EventArgs e)

{

Customization\_panel.Visible = true;

StartGame\_Button.Visible = false;

NameGame.Visible = false;

var fieryRocket = new FieryRocket(Direction.Up, typeof(Rocket));

AddRocketPropertys(fieryRocketPanel, fieryRocket);

var iceRocket = new IceRocket(Direction.Up, typeof(Rocket));

AddRocketPropertys(iceRocketPanel, iceRocket);

var atomicRocket = new AtomicRocket(Direction.Up, typeof(Rocket));

AddRocketPropertys(atomicRocketPanel, atomicRocket);

SubmarineAmunitionPanel.Text = $"Выберите {ammunitionCount} снарядов";

FieryRocketSum.Maximum = ammunitionCount;

IceRocketSum.Maximum = ammunitionCount;

AtomicRocketSum.Maximum = ammunitionCount;

}

/// <summary>

/// Вывод информации о ракетах.

/// </summary>

/// <param name="panel"></param>

/// <param name="rocket"></param>

private void AddRocketPropertys(Control panel, Rocket rocket)

{

int positionY = 0;

for (var i = 0; i < 4; i++)

{

var label1 = new Label();

var label = new Label()

{

ForeColor = Color.Black,

Width = 300,

Font = new Font(label1.Font.FontFamily, 15, label1.Font.Style),

Location = new Point(0, positionY)

};

switch (i)

{

case 0:

label.Text = $"Урон по жизням = {rocket.LifeDamage}";

break;

case 1:

label.Text = $"Бронепробиваемость = {rocket.ArmorDamage}";

break;

case 2:

label.Text = $"Скорость = {rocket.Speed}";

break;

case 3:

label.Text = $"Скорострельность = {rocket.FiringRate}";

break;

default:

break;

}

panel.Controls.Add(label);

positionY += 20;

}

}

/// <summary>

/// Измненение позиции элементов.

/// </summary>

/// <param name="element"></param>

/// <param name="x"></param>

/// <param name="y"></param>

private void CangePosition(Control element, int x, int y)

{

element.Location = new Point(x, y);

}

/// <summary>

/// Выбор боекоплекта для лодок.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void EquipSubmarinePanel\_Click(object sender, EventArgs e)

{

var sumAmmo = FieryRocketSum.Value + IceRocketSum.Value + AtomicRocketSum.Value;

if (sumAmmo == ammunitionCount)

{

if (submarines == 1)

{

redAmmunition.AddRockets(RocketType.FieryRocket, (int)FieryRocketSum.Value);

redAmmunition.AddRockets(RocketType.IceRocket, (int)IceRocketSum.Value);

redAmmunition.AddRockets(RocketType.AtomicRocket, (int)AtomicRocketSum.Value);

this.Close();

var mainWindow = new MainWindow(1920, 1080, blueAmmunition, redAmmunition);

mainWindow.Run(60);

}

else

{

blueAmmunition.AddRockets(RocketType.FieryRocket, (int)FieryRocketSum.Value);

blueAmmunition.AddRockets(RocketType.IceRocket, (int)IceRocketSum.Value);

blueAmmunition.AddRockets(RocketType.AtomicRocket, (int)AtomicRocketSum.Value);

SubmarineType.Text = "Снарядите второго игрока";

SubmarineType.ForeColor = Color.Red;

submarines++;

}

}

}

}

}

Листинг класса MainWindow.cs:

using System;

using OpenTK;

using System.Drawing;

using System.Collections.Generic;

using OpenTK.Graphics.OpenGL;

using OpenTK.Input;

using GameLogic;

using GameEngine;

namespace BattleOfTheSubmarinesGames

{

public class MainWindow : GameWindow

{

List<GameObject> gameObjects;

Dictionary<string, Texture2D> textures;

List<string> sprites;

Timerr bonusTimer = new Timerr();

Timerr blueSubmarineCooldown = new Timerr();

Timerr redSubmarineCooldown = new Timerr();

Timerr destroyerTimer = new Timerr();

RocketAmmunition blueSubAmmunition;

RocketAmmunition redSubAmmunition;

RocketType blueRocketType = RocketType.FieryRocket;

RocketType redRocketType = RocketType.FieryRocket;

bool activeRocket\_1 = true;

bool activeRocket\_2 = true;

bool activeMina = false;

/// <summary>

/// Инициализатор окна OpenTK.

/// </summary>

/// <param name="width"> Ширина. </param>

/// <param name="height"> Высота. </param>

public MainWindow(int width, int height, RocketAmmunition blueAmmo, RocketAmmunition redAmmo ) : base(width, height)

{

this.Location = new Point(-10, 0);

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Blend);

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

blueSubAmmunition = blueAmmo;

redSubAmmunition = redAmmo;

gameObjects = new List<GameObject>();

textures = new Dictionary<string, Texture2D>();

sprites = new List<string>()

{

"seaFloor.jpg",

"blueSubmarine.png",

"redSubmarine.png",

"health.png",

"armor.png",

"speed.png",

"ammunition.png",

"destroyer.png",

"mina.png",

"fieryRocket.png",

"iceRocket.png",

"atomicRocket.png",

"win\_1.png",

"win\_2.png"

};

LoadingGameObjects();

bonusTimer.Start(12000, CreateBonus);

destroyerTimer.Start(10000, CreateDestroyer);

}

/// <summary>

/// Загрузка игровых объектов.

/// </summary>

private void LoadingGameObjects()

{

//Загрузка текстур.

for (var i = 0; i < sprites.Count; i++)

{

textures.Add(sprites[i], TextureProcessing.LoadTexture2D(sprites[i]));

}

// Фон.

var background = new Background();

background.Components.Add(textures["seaFloor.jpg"]);

gameObjects.Add(background);

// Правая граница.

var rightBorder = new Border(Width, Height, Direction.Right);

gameObjects.Add(rightBorder);

// Левая граница.

var leftBorder = new Border(Width, Height, Direction.Left);

gameObjects.Add(leftBorder);

// Верхняя граница.

var upBorder = new Border(Width, Height, Direction.Up);

gameObjects.Add(upBorder);

// Нижняя граница.

var bottomBorder = new Border(Width, Height, Direction.Down);

gameObjects.Add(bottomBorder);

// Синяя лодка.

var blueSubmarine = new BlueSubmarine(blueSubAmmunition);

var sub2\_Texture = textures["blueSubmarine.png"];

blueSubmarine.Transform.Position = new Vector2(-(ClientSize.Width / 2) + sub2\_Texture.Width, 0);

blueSubmarine.Components.Add(sub2\_Texture);

blueSubmarine.Components.Add(new BoxCollider(sub2\_Texture.Width, sub2\_Texture.Height));

gameObjects.Add(blueSubmarine);

// Красная лодка.

var redSubmarine = new RedSubmarine(redSubAmmunition);

var sub1\_Texture = textures["redSubmarine.png"];

redSubmarine.Transform.Position = new Vector2((ClientSize.Width / 2) - sub1\_Texture.Width, 0);

redSubmarine.Transform.Scale = new Vector2(-redSubmarine.Transform.Scale.X, redSubmarine.Transform.Scale.Y);

redSubmarine.Components.Add(sub1\_Texture);

redSubmarine.Components.Add(new BoxCollider(sub1\_Texture.Width, sub1\_Texture.Height));

gameObjects.Add(redSubmarine);

}

/// <summary>

/// Завершение игры.

/// </summary>

/// <param name="typeSubmarine"> Тип победившей лодки.</param>

public void GameWin(Type typeSubmarine)

{

var texture = string.Empty;

if (typeSubmarine.Name.ToString() == nameof(BlueSubmarine))

{

texture = "win\_2.png";

}

else if (typeSubmarine.Name.ToString() == nameof(RedSubmarine))

{

texture = "win\_1.png";

}

var stopGame = new Timerr();

stopGame.Start(9000, StopGame);

bonusTimer.Stop();

destroyerTimer.Stop();

activeMina = true;

var win = new Background();

win.Components.Add(textures[texture]);

gameObjects.Add(win);

}

/// <summary>

/// Закрыть окно.

/// </summary>

private void StopGame()

{

this.Close();

}

/// <summary>

/// Удаление игровых объектов.

/// </summary>

/// <param name="go"></param>

private void RemoveGameObjects(GameObject go)

{

for (var i = 0; i < gameObjects.Count; i++)

{

if (gameObjects[i].GetType() == go.GetType())

{

gameObjects.RemoveAt(i);

}

}

}

/// <summary>

/// Контроль перезарядки синей лодки.

/// </summary>

private void ActivateCooldown\_1()

{

blueSubmarineCooldown.Stop();

activeRocket\_1 = true;

}

/// <summary>

/// Контроль перезарядки красной лодки

/// </summary>

private void ActivateCooldown\_2()

{

redSubmarineCooldown.Stop();

activeRocket\_2 = true;

}

/// <summary>

/// Создание мины.

/// </summary>

/// <param name="transform"></param>

private void CreateMina(Transform transform)

{

var mina = new Mina();

//RemoveGameObjects(mina);

mina.Transform.Scale = new Vector2(-transform.Scale.X, transform.Scale.Y);

mina.Transform.Position = new Vector2(transform.Position.X, transform.Position.Y);

mina.Components.Add(textures["mina.png"]);

mina.Components.Add(new BoxCollider(textures["mina.png"].Width, textures["mina.png"].Height));

gameObjects.Add(mina);

}

/// <summary>

/// Создание эсминца.

/// </summary>

private void CreateDestroyer()

{

var random = new Random();

int side = random.Next(0, 2);

Direction direction = (side == 0) ? (Direction.Right) : (Direction.Left);

side = (side == 0) ? (1) : (-1);

var destroyer = new Destroyer(direction);

RemoveGameObjects(destroyer);

destroyer.Transform.Scale = new Vector2(destroyer.Transform.Scale.X \* side, destroyer.Transform.Scale.Y);

destroyer.Transform.Position = new Vector2((-(ClientSize.Width + textures["destroyer.png"].Width) / 2) \* side, (ClientSize.Height - textures["destroyer.png"].Height) / 2);

destroyer.Components.Add(textures["destroyer.png"]);

gameObjects.Add(destroyer);

}

/// <summary>

/// Генерация бонусов.

/// </summary>

private void CreateBonus()

{

var bonus = new BonusGenerator();

RemoveGameObjects(bonus);

Texture2D bonusTexture = null;

switch (bonus.RandomBonus)

{

case 0:

bonusTexture = textures["health.png"];

break;

case 1:

bonusTexture = textures["armor.png"];

break;

case 2:

bonusTexture = textures["speed.png"];

break;

case 3:

bonusTexture = textures["ammunition.png"];

break;

default:

break;

}

var random = new Random();

int x = random.Next(-(ClientSize.Width - bonusTexture.Width) / 2, (ClientSize.Width - bonusTexture.Width) / 2);

int y = random.Next(-((ClientSize.Height - bonusTexture.Height) / 2), (ClientSize.Height - bonusTexture.Height) / 2);

bonus.Transform.Position = new Vector2(x, y);

bonus.Components.Add(bonusTexture);

bonus.Components.Add(new BoxCollider(bonusTexture.Width, bonusTexture.Height));

gameObjects.Add(bonus);

}

/// <summary>

/// Создание ракеты.

/// </summary>

/// <param name="transform"> Позиция стреляющей лодки. </param>

/// <param name="owner"> Тип стреляющей лодки. </param>

/// <param name="rocketType"> Тип ракеты. </param>

/// <returns> Скорострельность запускаемой ракеты. </returns>

private int CreateRocket(Transform transform, Type owner, RocketType rocketType)

{

int sign = (transform.Scale.X < 0) ? (-1) : (1);

Direction direction = (transform.Scale.X < 0) ? (Direction.Left) : (Direction.Right);

Rocket rocket = null;

switch (rocketType)

{

case RocketType.FieryRocket:

rocket = new FieryRocket(direction, owner);

rocket.Components.Add(textures["fieryRocket.png"]);

rocket.Components.Add(new BoxCollider(textures["fieryRocket.png"].Width, textures["fieryRocket.png"].Height));

break;

case RocketType.IceRocket:

rocket = new IceRocket(direction, owner);

rocket.Components.Add(textures["iceRocket.png"]);

rocket.Components.Add(new BoxCollider(textures["iceRocket.png"].Width, textures["iceRocket.png"].Height));

break;

case RocketType.AtomicRocket:

rocket = new AtomicRocket(direction, owner);

rocket.Components.Add(textures["atomicRocket.png"]);

rocket.Components.Add(new BoxCollider(textures["atomicRocket.png"].Width, textures["atomicRocket.png"].Height));

break;

default:

break;

}

rocket.Transform.Position = new Vector2(transform.Position.X, transform.Position.Y - 50);

rocket.Transform.Scale = new Vector2(rocket.Transform.Scale.X \* sign, rocket.Transform.Scale.Y);

gameObjects.Add(rocket);

return rocket.FiringRate;

}

/// <summary>

/// Проверка на столкновение объектов.

/// </summary>

/// <param name="go"> Игровой объект. </param>

/// <returns> Итог проверги на столкновения. </returns>

private bool CheckingColliders(GameObject go)

{

if (go.GetComponent<Collider>() == null)

{

throw new Exception($"Игровой объект {go.GetType().Name} не имеет коллайдера!"); ;

}

for (var i = 0; i < gameObjects.Count; i++)

{

if (!gameObjects[i].Equals(go))

{

if (gameObjects[i].GetComponent<Collider>() != null)

{

if (Collider.CheckCollision(gameObjects[i], go))

{

// При столкновении лодки с лодкой, стеной.

if (go is Submarine && gameObjects[i] is Submarine || gameObjects[i] is Border)

{

return true;

}

// При столкновении ракеты с лодкой.

if (go is Rocket && gameObjects[i] is Submarine)

{

var sub = (Submarine)gameObjects[i];

var rocket = (Rocket)go;

if (rocket.Owner != sub.BasicType)

{

sub.TakingDamage(rocket.LifeDamage, rocket.ArmorDamage);

gameObjects[i] = sub;

return true;

}

return false;

}

// При столкновении бонуса с лодкой.

if (go is BonusGenerator && gameObjects[i] is Submarine)

{

gameObjects[i] = DecorationSubmarine(gameObjects[i], go);

return true;

}

// При столкновении мины с лодкой.

if (go is Mina)

{

if (gameObjects[i] is Submarine)

{

var sub = (Submarine)gameObjects[i];

var mina = (Mina)go;

sub.TakingDamage(mina.LifeDamage, mina.ArmorDamage);

gameObjects[i] = sub;

return true;

}

if (gameObjects[i] is Rocket)

{

return true;

}

}

}

}

}

}

return false;

}

/// <summary>

/// Декарирование лодки.

/// </summary>

/// <param name="goSub"> Лодка. </param>

/// <param name="goBonus"> Бонус. </param>

/// <returns> Декарированную лодку.</returns>

private Submarine DecorationSubmarine(GameObject goSub, GameObject goBonus)

{

var sub = (Submarine)goSub;

var bonusGenerator = (BonusGenerator)goBonus;

var oldPosition = sub.Transform.Position;

var oldScale = sub.Transform.Scale;

var bonus = bonusGenerator.GenerateBonus();

sub = bonus.Activation(sub);

sub.Transform.Position = oldPosition;

sub.Transform.Scale = oldScale;

return sub;

}

/// <summary>

/// OnRenderFrame.

/// </summary>

/// <param name="e"> Данные события. </param>

protected override void OnLoad(EventArgs e)

{

base.OnLoad(e);

SpriteRendering.Begin(Width, Height);

}

/// <summary>

/// Покадровая отрисовка.

/// </summary>

/// <param name="e"> Данные события. </param>

protected override void OnRenderFrame(FrameEventArgs e)

{

GL.Clear(ClearBufferMask.ColorBufferBit);

base.OnRenderFrame(e);

try

{

foreach (GameObject gameObject in gameObjects)

{

if (gameObject.GetComponent<Texture2D>() != null)

{

SpriteRendering.Draw((Texture2D)gameObject.GetComponent<Texture2D>(), gameObject.Transform);

}

}

}

catch { }

SwapBuffers();

}

/// <summary>

/// Покадровая логика.

/// </summary>

/// <param name="e"> Данные события. </param>

protected override void OnUpdateFrame(FrameEventArgs e)

{

base.OnUpdateFrame(e);

var property = String.Empty;

var gameObjectOrderRemove = new List<GameObject>();

for (var i = 0; i < gameObjects.Count; i++)

{

if (gameObjects[i] is Submarine)

{

var sub = (Submarine)gameObjects[i];

if (sub.BasicType.Name.ToString() == nameof(BlueSubmarine))

{

ControllingBluePlayer(e, sub);

}

else if (sub.BasicType.Name.ToString() == nameof(RedSubmarine))

{

ControllingRedPlayer(e, sub);

}

if (sub.Health <= 0)

{

GameWin(sub.BasicType);

gameObjectOrderRemove.Add(gameObjects[i]);

}

property += sub.ToString() +

" " +

" ";

}

if (gameObjects[i] is Rocket)

{

var rocket = (Rocket)gameObjects[i];

rocket.Move(e.Time);

if (CheckingColliders(rocket))

{

gameObjectOrderRemove.Add(gameObjects[i]);

}

}

if (gameObjects[i] is BonusGenerator)

{

var bonus = (BonusGenerator)gameObjects[i];

if (CheckingColliders(bonus))

{

gameObjectOrderRemove.Add(gameObjects[i]);

}

}

if (gameObjects[i] is Destroyer)

{

var destroyer = (Destroyer)gameObjects[i];

destroyer.Move(e.Time);

var random = new Random();

var subId = SearchSubmarineId(random.Next(1, 3));

if (subId != -1)

{

if (destroyer.Transform.Position.X <= gameObjects[subId].Transform.Position.X && !activeMina)

{

CreateMina(destroyer.Transform);

activeMina = true;

}

}

}

if (gameObjects[i] is Mina)

{

var mina = (Mina)gameObjects[i];

mina.Move(e.Time);

if (CheckingColliders(mina))

{

gameObjectOrderRemove.Add(gameObjects[i]);

activeMina = false;

}

}

}

foreach (GameObject go in gameObjectOrderRemove)

{

gameObjects.Remove(go);

}

Title = property;

}

/// <summary>

/// Поиск лодки в коллекции.

/// </summary>

/// <param name="numberOject"></param>

/// <returns></returns>

private int SearchSubmarineId(int numberOject)

{

for (var i = 0; i < gameObjects.Count; i++)

{

if (gameObjects[i] is Submarine)

{

var sub = (Submarine)gameObjects[i];

if (sub.BasicType.Name.ToString() == nameof(BlueSubmarine) && numberOject == 1)

{

return i;

}

else if (sub.BasicType.Name.ToString() == nameof(RedSubmarine) && numberOject == 2)

{

return i;

}

}

}

return -1;

}

/// <summary>

/// Управление синего игрока.

/// </summary>

/// <param name="e"></param>

private void ControllingBluePlayer(FrameEventArgs e, Submarine submarine)

{

KeyboardState kb = Keyboard.GetState();

var scaleX = submarine.Transform.Scale.X < 0 ? submarine.Transform.Scale.X \* -1 : submarine.Transform.Scale.X;

var scaleY = submarine.Transform.Scale.Y;

Vector2 oldPosition = submarine.Transform.Position;

if (kb.IsKeyDown(Key.Number1))

{

blueRocketType = RocketType.FieryRocket;

}

if (kb.IsKeyDown(Key.Number2))

{

blueRocketType = RocketType.IceRocket;

}

if (kb.IsKeyDown(Key.Number3))

{

blueRocketType = RocketType.AtomicRocket;

}

if (kb.IsKeyDown(Key.Space) && activeRocket\_1 && submarine.Ammunition.GetRockets(blueRocketType) > 0)

{

var colldown = CreateRocket(submarine.Transform, submarine.BasicType, blueRocketType);

blueSubmarineCooldown.Start(colldown, ActivateCooldown\_1);

submarine.Shoot(blueRocketType);

activeRocket\_1 = false;

}

if (kb.IsKeyDown(Key.W) ^ kb.IsKeyDown(Key.S))

{

if (kb.IsKeyDown(Key.W))

{

submarine.Move(Direction.Up, e.Time);

}

else

{

submarine.Move(Direction.Down, e.Time);

}

if (CheckingColliders(submarine))

{

submarine.Transform.Position = oldPosition;

}

}

if (kb.IsKeyDown(Key.A) ^ kb.IsKeyDown(Key.D))

{

if (kb.IsKeyDown(Key.A))

{

submarine.Transform.Scale = new Vector2(-scaleX, scaleY);

submarine.Move(Direction.Left, e.Time);

}

else

{

submarine.Transform.Scale = new Vector2(scaleX, scaleY);

submarine.Move(Direction.Right, e.Time);

}

if (CheckingColliders(submarine))

{

submarine.Transform.Position = oldPosition;

}

}

}

/// <summary>

/// Управление красного игрока.

/// </summary>

/// <param name="e"> Данные события. </param>

private void ControllingRedPlayer(FrameEventArgs e, Submarine submarine)

{

KeyboardState kb = Keyboard.GetState();

var scaleX = submarine.Transform.Scale.X < 0 ? submarine.Transform.Scale.X \* -1 : submarine.Transform.Scale.X;

var scaleY = submarine.Transform.Scale.Y;

Vector2 oldPosition = submarine.Transform.Position;

if (kb.IsKeyDown(Key.Keypad1))

{

redRocketType = RocketType.FieryRocket;

}

if (kb.IsKeyDown(Key.Keypad2))

{

redRocketType = RocketType.IceRocket;

}

if (kb.IsKeyDown(Key.Keypad3))

{

redRocketType = RocketType.AtomicRocket;

}

if (kb.IsKeyDown(Key.KeypadEnter) && activeRocket\_2 && submarine.Ammunition.GetRockets(redRocketType) > 0)

{

var colldown = CreateRocket(submarine.Transform, submarine.BasicType, redRocketType);

redSubmarineCooldown.Start(colldown, ActivateCooldown\_2);

submarine.Shoot(redRocketType);

activeRocket\_2 = false;

}

if (kb.IsKeyDown(Key.Up) ^ kb.IsKeyDown(Key.Down))

{

if (kb.IsKeyDown(Key.Up))

{

submarine.Move(Direction.Up, e.Time);

}

else

{

submarine.Move(Direction.Down, e.Time);

}

if (CheckingColliders(submarine))

{

submarine.Transform.Position = oldPosition;

}

}

if (kb.IsKeyDown(Key.Left) ^ kb.IsKeyDown(Key.Right))

{

if (kb.IsKeyDown(Key.Left))

{

submarine.Transform.Scale = new Vector2(-scaleX, scaleY);

submarine.Move(Direction.Left, e.Time);

}

else

{

submarine.Transform.Scale = new Vector2(scaleX, scaleY);

submarine.Move(Direction.Right, e.Time);

}

if (CheckingColliders(submarine))

{

submarine.Transform.Position = oldPosition;

}

}

}

}

}

Листинг класса BoxCollider.cs:

using OpenTK;

using System;

using System.Collections.Generic;

namespace GameEngine

{

/// <summary>

/// Коллайдер прямоугольной формы.

/// </summary>

public class BoxCollider : Collider

{

private int width;

private int height;

/// <summary>

/// Инициализатор коллайдера.

/// </summary>

/// <param name="width"> Ширина. </param>

/// <param name="height"> Высота. </param>

public BoxCollider(int width, int height)

{

if (width <= 0)

{

throw new ArgumentException($"Ширина коллайдера не может быть {width}", nameof(width));

}

if (height <= 0)

{

throw new ArgumentException($"Высота коллайдера не может быть {height}", nameof(height));

}

this.width = width;

this.height = height;

}

/// <summary>

/// Получаем координаты сторон прямоугольного объекта, оболочка объекта.

/// </summary>

/// <param name="gameObject"> Игровой объект. </param>

/// <returns> Коллекция координат. </returns>

protected override List<Vector2> GetField(GameObject gameObject)

{

if (!gameObject.GetComponent<Collider>().Equals(this))

{

throw new Exception($"Тип коллайдера {gameObject.GetType().Name} не cоответствует {this.GetType()}!");

}

float x = gameObject.Transform.Position.X;

float y = gameObject.Transform.Position.Y;

var halfWidth = width / 2;

var halfHeight = height / 2;

var vertices = new List<Vector2>()

{

new Vector2(x - halfWidth, y + halfHeight), // top left

new Vector2(x + halfWidth, y + halfHeight), // top right

new Vector2(x + halfWidth, y - halfHeight), // bottom right

new Vector2(x - halfWidth, y - halfHeight), // bottom left

};

return vertices;

}

}

}

Листинг класса Collider.cs:

using System;

using System.Collections.Generic;

using OpenTK;

namespace GameEngine

{

/// <summary>

/// Коллайдер.

/// </summary>

public abstract class Collider : GameComponents

{

/// <summary>

/// Получаем координаты сторон объекта, оболочка объекта.

/// </summary>

/// <param name="gameObject"> Игровой объект. </param>

/// <returns> Коллекция координат. </returns>

protected abstract List<Vector2> GetField(GameObject gameObject);

/// <summary>

/// Получение вершин объекта.

/// </summary>

/// <param name="gameObject"> Игровой объект. </param>

/// <returns> Вершины. </returns>

private static List<Vector2> GetVertices(GameObject gameObject)

{

return (gameObject.GetComponent<Collider>() as Collider).GetField(gameObject);

}

/// <summary>

/// Проверка на столкновение игровых объектов.

/// </summary>

/// <param name="gameObject\_1"> Первый объект. </param>

/// <param name="gameObject\_2"> Второй объект. </param>

/// <returns> Логическо значение, обозначающее произошло ли столкновение. </returns>

public static bool CheckCollision(GameObject gameObject\_1, GameObject gameObject\_2)

{

if (gameObject\_1.GetComponent<Collider>() == null)

{

throw new ArgumentNullException($"Игровой объект {gameObject\_1.GetType().Name} не имеет коллайдера!", nameof(gameObject\_1));

}

if (gameObject\_2.GetComponent<Collider>() == null)

{

throw new ArgumentNullException($"Игровой объект {gameObject\_2.GetType().Name} не имеет коллайдера!", nameof(gameObject\_2));

}

if (gameObject\_1 == null)

{

throw new ArgumentNullException($"Игровой объект {gameObject\_1.GetType().Name} является null!", nameof(gameObject\_1));

}

if (gameObject\_2 == null)

{

throw new ArgumentNullException($"Игровой объект {gameObject\_2.GetType().Name} является null!", nameof(gameObject\_2));

}

List<Vector2> vertices\_1 = GetVertices(gameObject\_1);

List<Vector2> vertices\_2 = GetVertices(gameObject\_2);

foreach (Vector2 point in vertices\_2)

{

if (CheckPointCollision(gameObject\_1, point))

{

return true;

}

}

foreach (Vector2 point in vertices\_1)

{

if (CheckPointCollision(gameObject\_2, point))

{

return true;

}

}

return false;

}

/// <summary>

/// Проверка нахождения игрового объекта на определённой позиции.

/// </summary>

/// <param name="gameObject"> Игровой объект. </param>

/// <param name="point"> Конкретная точка. </param>

/// <returns></returns>

private static bool CheckPointCollision(GameObject gameObject, Vector2 point)

{

bool collision = false;

float x = point.X;

float y = point.Y;

List<Vector2> vertices = GetVertices(gameObject);

for (int i = 0, j = vertices.Count - 1; i < vertices.Count; j = i++)

{

if ((((vertices[i].Y <= y) && (y < vertices[j].Y)) ||

((vertices[j].Y <= y) && (y < vertices[i].Y))) &&

(((vertices[j].Y - vertices[i].Y) != 0) &&

(x > ((vertices[j].X - vertices[i].X) \* (y - vertices[i].Y) / (vertices[j].Y - vertices[i].Y) + vertices[i].X))))

{

collision = !collision;

}

}

return collision;

}

}

}

Листинг класса GameComponents.cs:

using System;

namespace GameEngine

{

/// <summary>

/// Тип компонентов игрового объекта.

/// </summary>

public abstract class GameComponents { }

}

Листинг класса GameObject.cs:

using System.Collections.Generic;

namespace GameEngine

{

/// <summary>

/// Игровой объект.

/// </summary>

public abstract class GameObject

{

private Transform transform;

private List<GameComponents> components;

/// <summary>

/// Инициализатор игрового объекта.

/// </summary>

public GameObject()

{

transform = new Transform();

components = new List<GameComponents>();

}

/// <summary>

/// Позиция, вращение, масштаб объекта.

/// </summary>

public Transform Transform

{

get { return transform; }

}

/// <summary>

/// Компоненты объекта.

/// </summary>

public List<GameComponents> Components

{

get { return components; }

}

/// <summary>

/// Проверка типов компонентов.

/// </summary>

/// <typeparam name="TComponent"></typeparam>

/// <returns></returns>

public GameComponents GetComponent<TComponent>() where TComponent : GameComponents

{

foreach (GameComponents component in components)

{

if (component is TComponent)

{

return component as TComponent;

}

}

return null;

}

}

}

Листинг класса SpriteRendering.cs:

using OpenTK;

using OpenTK.Graphics.OpenGL;

namespace GameEngine

{

/// <summary>

/// Работа с текстурой, отрисовка.

/// </summary>

public class SpriteRendering

{

/// <summary>

/// Отрисовка текстуры.

/// </summary>

/// <param name="texture"> Класс со свойствами текстуры. </param>

/// <param name="transform"> Позиция. </param>

public static void Draw(Texture2D texture, Transform transform)

{

Vector2[] vertices = new Vector2[4]

{

new Vector2(0, 0),

new Vector2(1, 0),

new Vector2(1, 1),

new Vector2(0, 1),

};

GL.BindTexture(TextureTarget.Texture2D, texture.Id);

GL.Begin(PrimitiveType.Quads);

for (var i = 0; i < vertices.Length; ++i)

{

GL.TexCoord2(vertices[i]);

vertices[i].X \*= texture.Width;

vertices[i].Y \*= texture.Height;

vertices[i] \*= transform.Scale;

vertices[i] += transform.Position;

vertices[i] -= new Vector2(((transform.Scale.X < 0) ? -1 : 1) \* texture.Width / 2, ((transform.Scale.Y < 0) ? -1 : 1) \* texture.Height / 2);

GL.Vertex2(vertices[i]);

}

GL.End();

}

public static void Begin(int screenWidth, int screenHeight)

{

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

int halfScreenWidth = screenWidth / 2;

int halfScreenHeight = screenHeight / 2;

GL.Ortho(-halfScreenWidth, halfScreenWidth, -halfScreenHeight, halfScreenHeight, 0.0f, 1.0f);

}

}

}

Листинг класса Texture2D.cs:

using System;

namespace GameEngine

{

/// <summary>

/// Свойства текстуры.

/// </summary>

public class Texture2D : GameComponents

{

private int id;

private int width;

private int height;

/// <summary>

/// Инициализатор текстуры.

/// </summary>

/// <param name="id"> Номер текстуры. </param>

/// <param name="width"> Ширина. </param>

/// <param name="height"> Высота. </param>

public Texture2D(int id, int width, int height)

{

if (width <= 0)

{

throw new ArgumentException($"Ширина текстура не может быть {width}", nameof(width));

}

if (height <= 0)

{

throw new ArgumentException($"Высота текстура не может быть {height}", nameof(height));

}

this.id = id;

this.width = width;

this.height = height;

}

/// <summary>

/// Номер текстуры.

/// </summary>

public int Id

{

get { return id; }

}

/// <summary>

/// Ширина.

/// </summary>

public int Width

{

get { return width; }

set

{

if (value > 0)

{

width = value;

}

}

}

/// <summary>

/// Высота.

/// </summary>

public int Height

{

get { return height; }

set

{

if (value > 0)

{

height = value;

}

}

}

}

}

Листинг класса TextureProcessing.cs:

using System;

using System.IO;

using System.Drawing;

using System.Drawing.Imaging;

using OpenTK.Graphics.OpenGL;

namespace GameEngine

{

public static class TextureProcessing

{

/// <summary>

/// Создание текстуры.

/// </summary>

/// <param name="path"> Путь к изображению. </param>

/// <returns> Класс с параметрами текстуры. </returns>

public static Texture2D LoadTexture2D(string path)

{

if (!File.Exists("Sprites/" + path))

{

throw new ArgumentException($"Файл не найден, проверьте путь {path}");

}

int id = GL.GenTexture();

GL.BindTexture(TextureTarget.Texture2D, id);

Bitmap bmp = new Bitmap("Sprites/" + path);

bmp.RotateFlip(RotateFlipType.Rotate180FlipX);

BitmapData data = bmp.LockBits(new Rectangle(0, 0, bmp.Width, bmp.Height),

ImageLockMode.ReadOnly,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, data.Width, data.Height, 0,

OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, data.Scan0);

bmp.UnlockBits(data);

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

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

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

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

return new Texture2D(id, bmp.Width, bmp.Height);

}

}

}

Листинг класса Transform.cs:

using OpenTK;

namespace GameEngine

{

/// <summary>

/// Значения позиции, вращения, масштаба объекта.

/// </summary>

public class Transform

{

/// <summary>

/// Позиция.

/// </summary>

public Vector2 Position { get; set; }

/// <summary>

/// Вращение.

/// </summary>

public Vector2 Rotation { get; set; }

/// <summary>

/// Масштаб.

/// </summary>

public Vector2 Scale { get; set; }

/// <summary>

/// Инициализатор класса Transform.

/// </summary>

public Transform()

{

Position = new Vector2(0.0f, 0.0f);

Rotation = new Vector2(0.0f, 0.0f);

Scale = new Vector2(1.0f, 1.0f);

}

}

}

Листинг класса SubmarineDecorator.cs:

using System;

using GameEngine;

namespace GameLogic

{

public abstract class SubmarineDecorator : Submarine

{

protected Submarine submarine;

/// <summary>

/// Инициализатор.

/// </summary>

/// <param name="submarine"> Декорируемая лодка. </param>

protected SubmarineDecorator(Submarine submarine) : base(submarine.Ammunition)

{

if (submarine == null)

{

throw new ArgumentNullException();

}

this.submarine = submarine;

Health = submarine.Health;

Armor = submarine.Armor;

Speed = submarine.Speed;

BasicType = submarine.BasicType;

AmmunitionCount = submarine.AmmunitionCount;

foreach (GameComponents goc in submarine.Components)

{

this.Components.Add(goc);

}

}

}

}

Листинг класса AdditionalAmmunition.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Пополнение боеприпасов лодки.

/// </summary>

public class AdditionalAmmunition : SubmarineDecorator

{

private static int boostAmmunition = 1;

public AdditionalAmmunition(Submarine submarine) : base(submarine)

{

Ammunition.ChangeRockets(RocketType.FieryRocket, boostAmmunition);

Ammunition.ChangeRockets(RocketType.IceRocket, boostAmmunition);

Ammunition.ChangeRockets(RocketType.AtomicRocket, boostAmmunition);

}

/// <summary>

/// Значение дополнительных боеприпасов, свойство необходимо для тестов.

/// </summary>

public int BoostAmmunition

{

get { return boostAmmunition \* Ammunition.TypeRocketCount; }

}

}

}

Листинг класса AdditionalArmor.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Увеличение брони лодки.

/// </summary>

public class AdditionalArmor : SubmarineDecorator

{

private static int boostArmor = 20;

public AdditionalArmor(Submarine submarine) : base( submarine )

{

Armor += boostArmor;

}

/// <summary>

/// Значение дополнительной брони, свойство необходимо для тестов.

/// </summary>

public int BoostArmor

{

get { return boostArmor; }

}

}

}

Листинг класса AdditionalHealth.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Увеличение жизней лодки.

/// </summary>

public class AdditionalHealth : SubmarineDecorator

{

private static int boostHealth = 20;

public AdditionalHealth(Submarine submarine) : base(submarine)

{

Health += boostHealth;

}

/// <summary>

/// Значение дополнительных жизней, свойство необходимо для тестов.

/// </summary>

public int BoostHealth

{

get { return boostHealth; }

}

}

}

Листинг класса AdditionalSpeed.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Увеличение скорость лодки.

/// </summary>

public class AdditionalSpeed : SubmarineDecorator

{

private static int boostSpeed = 2;

public AdditionalSpeed(Submarine submarine) : base(submarine)

{

Speed += boostSpeed;

}

/// <summary>

/// Значение дополнительной скорости, свойство необходимо для тестов.

/// </summary>

public int BoostSpeed

{

get { return boostSpeed; }

}

}

}

Листинг класса AmmunitionBonus.cs:

using System;

namespace GameLogic

{

public class AmmunitionBonus : Bonus

{

public override Submarine Activation(Submarine submarine)

{

return new AdditionalAmmunition(submarine);

}

}

}

Листинг класса ArmorBonus.cs:

using System;

namespace GameLogic

{

public class ArmorBonus : Bonus

{

public override Submarine Activation(Submarine submarine)

{

return new AdditionalArmor(submarine);

}

}

}

Листинг класса Bonus.cs:

using System;

namespace GameLogic

{

public abstract class Bonus

{

public abstract Submarine Activation(Submarine submarine);

}

}

Листинг класса BonusCreator.cs:

using System;

namespace GameLogic

{

public abstract class BonusCreator

{

public abstract Bonus CreateBonus();

}

}

Листинг класса CreatorAmmunition.cs:

using System;

namespace GameLogic

{

public class CreatorAmmunition : BonusCreator

{

public override Bonus CreateBonus()

{

return new AmmunitionBonus();

}

}

}

Листинг класса CreatorArmor.cs:

using System;

namespace GameLogic

{

public class CreatorArmor : BonusCreator

{

public override Bonus CreateBonus()

{

return new ArmorBonus();

}

}

}

Листинг класса CreatorHealth.cs:

using System;

namespace GameLogic

{

public class CreatorHealth : BonusCreator

{

public override Bonus CreateBonus()

{

return new HealthBonus();

}

}

}

Листинг класса CreatorSpeed.cs:

using System;

namespace GameLogic

{

public class CreatorSpeed : BonusCreator

{

public override Bonus CreateBonus()

{

return new SpeedBonus();

}

}

}

Листинг класса HealthBonus.cs:

using System;

namespace GameLogic

{

public class HealthBonus : Bonus

{

public override Submarine Activation(Submarine submarine)

{

return new AdditionalHealth(submarine);

}

}

}

Листинг класса SpeedBonus.cs:

using System;

namespace GameLogic

{

public class SpeedBonus : Bonus

{

public override Submarine Activation(Submarine submarine)

{

return new AdditionalSpeed(submarine);

}

}

}

Листинг класса Submarine.cs:

using System;

using OpenTK;

using GameEngine;

namespace GameLogic

{

/// <summary>

/// Подводная лодка.

/// </summary>

public abstract class Submarine : GameObject, IMovableSubmarine

{

/// <summary>

/// Количество жизней.

/// </summary>

public int Health { get; protected set; }

/// <summary>

/// Запас брони.

/// </summary>

public int Armor { get; protected set; }

/// <summary>

/// Скорость.

/// </summary>

public float Speed { get; protected set; }

/// <summary>

/// Ракеты.

/// </summary>

public RocketAmmunition Ammunition { get; protected set; }

/// <summary>

/// Начальное количество ракет.

/// </summary>

public int AmmunitionCount { get; protected set; }

/// <summary>

/// Настоящий тип лодки, до декарирования.

/// </summary>

public Type BasicType { get; protected set; }

/// <summary>

/// Инициализатор лодки.

/// </summary>

public Submarine(RocketAmmunition ammo)

{

Health = 100;

Speed = 18;

Armor = 100;

AmmunitionCount = 15;

Ammunition = ammo;

BasicType = this.GetType();

}

/// <summary>

/// Получение урона.

/// </summary>

/// <param name="lifeDamage"> Урон по здоровью. </param>

/// <param name="armorDamage"> Урон по броне. </param>

public void TakingDamage(int lifeDamage, int armorDamage)

{

if (Armor == 0)

{

Health -= lifeDamage;

}

else

{

Armor -= armorDamage;

if (Armor < 0)

{

Health -= lifeDamage - (-Armor);

Armor = 0;

}

}

}

/// <summary>

/// Выстрел, расходует ракету.

/// </summary>

/// <param name="rocketType"> Тип ракеты. </param>

public void Shoot(RocketType rocketType)

{

Ammunition.ChangeRockets(rocketType, -1);

}

/// <summary>

/// Движение лодки.

/// </summary>

/// <param name="direction"> Направление. </param>

/// <param name="time"> Время. </param>

public void Move(Direction direction, double time)

{

float x = Transform.Position.X;

float y = Transform.Position.Y;

switch (direction)

{

case Direction.Up:

y += (float)(Math.Pow(Speed, 2) \* time);

break;

case Direction.Down:

y -= (float)(Math.Pow(Speed, 2) \* time);

break;

case Direction.Left:

x -= (float)(Math.Pow(Speed, 2) \* time);

break;

case Direction.Right:

x += (float)(Math.Pow(Speed, 2) \* time);

break;

default:

break;

}

Transform.Position = new Vector2(x, y);

}

/// <summary>

/// Вывод характеристик лодки.

/// </summary>

/// <returns> Характеристика лодки. </returns>

public override string ToString()

{

return $"Жизни: {Health} | Броня: {Armor} | Скорость: {Speed} | Снаряды: {Ammunition.Count()} - ({Ammunition.GetRockets(RocketType.FieryRocket)},{Ammunition.GetRockets(RocketType.IceRocket)},{Ammunition.GetRockets(RocketType.AtomicRocket)})";

}

}

}

Листинг класса RedSubmarine.cs:

using System;

namespace GameLogic

{

public class RedSubmarine : Submarine

{

public RedSubmarine(RocketAmmunition ammo) : base(ammo) { }

}

}

Листинг класса BlueSubmarine.cs:

using System;

namespace GameLogic

{

public class BlueSubmarine : Submarine

{

public BlueSubmarine(RocketAmmunition ammo) : base(ammo) { }

}

}

Листинг класса Destroyer.cs:

using System;

using OpenTK;

using GameEngine;

namespace GameLogic

{

/// <summary>

/// Миноносец.

/// </summary>

public class Destroyer : GameObject, IMovable

{

private float speed;

private Direction direction;

/// <summary>

/// Инициализатор миноносца.

/// </summary>

/// <param name="direction"></param>

public Destroyer(Direction direction)

{

speed = 20;

this.direction = direction;

}

/// <summary>

/// Движение миноносца.

/// </summary>

/// <param name="time"> Время. </param>

public void Move(double time)

{

float x = Transform.Position.X;

float y = Transform.Position.Y;

if (direction == Direction.Right)

{

x += (float)(Math.Pow(speed, 2) \* time);

}

if (direction == Direction.Left)

{

x -= (float)(Math.Pow(speed, 2) \* time);

}

Transform.Position = new Vector2(x, y);

}

}

}

Листинг класса Mina.cs:

using System;

using OpenTK;

using GameEngine;

namespace GameLogic

{

/// <summary>

/// Мина.

/// </summary>

public class Mina : GameObject, IMovable

{

private int lifeDamage;

private int armorDamage;

private float speed;

/// <summary>

/// Инициализатор мины.

/// </summary>

public Mina()

{

lifeDamage = 30;

armorDamage = 30;

speed = 20;

}

/// <summary>

/// Урон по здоровью.

/// </summary>

public int LifeDamage

{

get { return lifeDamage; }

}

/// <summary>

/// Бронепробиваемость.

/// </summary>

public int ArmorDamage

{

get { return armorDamage; }

}

/// <summary>

/// Движение мины.

/// </summary>

/// <param name="time"> Время. </param>

public void Move(double time)

{

float x = Transform.Position.X;

float y = Transform.Position.Y;

y -= (float)(Math.Pow(speed, 2) \* time);

Transform.Position = new Vector2(x, y);

}

}

}

Листинг интерфейса IMovable.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Движение ракеты.

/// </summary>

interface IMovable

{

void Move(double time);

}

}

Листинг интерфейса IMovableSubmarine.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Движение игровых объектов.

/// </summary>

public interface IMovableSubmarine

{

void Move(Direction direction, double time);

}

}

Листинг класса BonusGenerator.cs:

using System;

using GameEngine;

namespace GameLogic

{

/// <summary>

/// Генератор бонусов.

/// </summary>

public class BonusGenerator : GameObject

{

private int randomBonus;

/// <summary>

/// Инициализатор бонуса.

/// </summary>

public BonusGenerator()

{

Random random = new Random();

this.randomBonus = random.Next(0, 4);

}

/// <summary>

/// Номер бонуса.

/// </summary>

public int RandomBonus

{

get { return randomBonus; }

}

/// <summary>

/// Генерация бонуса.

/// </summary>

/// <returns> Бонус. </returns>

public Bonus GenerateBonus()

{

BonusCreator bonusCreator = null;

switch (randomBonus)

{

case 0:

bonusCreator = new CreatorHealth();

break;

case 1:

bonusCreator = new CreatorArmor();

break;

case 2:

bonusCreator = new CreatorSpeed();

break;

case 3:

bonusCreator = new CreatorAmmunition();

break;

default:

break;

}

return bonusCreator.CreateBonus();

}

}

}

Листинг класса Direction.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Направление движения.

/// </summary>

public enum Direction : int

{

Up,

Down,

Left,

Right

}

}

Листинг класса RocketAmmunition.cs:

using System.Collections.Generic;

namespace GameLogic

{

/// <summary>

/// Снаряжение подводной лодки.

/// </summary>

public class RocketAmmunition

{

private Dictionary<RocketType, int> ammunition;

/// <summary>

/// Инициализатор снаряжения.

/// </summary>

public RocketAmmunition()

{

ammunition = new Dictionary<RocketType, int>();

}

/// <summary>

/// Количестов ракет заданного типа.

/// </summary>

public int GetRockets(RocketType rocketType)

{

if (ammunition.ContainsKey(rocketType))

{

return ammunition[rocketType];

}

return 0;

}

/// <summary>

/// Количество всех ракет.

/// </summary>

/// <returns> Количество всех ракет. </returns>

public int Count()

{

int count = 0;

foreach (KeyValuePair<RocketType, int> ammo in ammunition)

{

count += ammo.Value;

}

return count;

}

/// <summary>

/// Количество типов ракет.

/// </summary>

/// <returns> Количество типов ракет. </returns>

public int TypeRocketCount

{

get { return ammunition.Count; }

}

/// <summary>

/// Добвление ракет заданного типа.

/// </summary>

/// <param name="rocketType"></param>

/// <param name="rocketCount"></param>

public void AddRockets(RocketType rocketType, int rocketCount)

{

if (!ammunition.ContainsKey(rocketType))

{

ammunition.Add(rocketType, rocketCount);

}

}

/// <summary>

/// Изменение ракет заданного типа, (добавить/отнять).

/// </summary>

/// <param name="rocketType"></param>

public void ChangeRockets(RocketType rocketType, int countRockets)

{

if (ammunition.ContainsKey(rocketType))

{

ammunition[rocketType] += countRockets;

}

}

}

}

Листинг класса RocketType.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Типы рокет.

/// </summary>

public enum RocketType : int

{

FieryRocket,

IceRocket,

AtomicRocket

}

}

Листинг класса Timerr.cs:

using System;

using System.Timers;

using Timer = System.Timers.Timer;

namespace GameLogic

{

/// <summary>

/// Таймер.

/// </summary>

public class Timerr

{

private Timer aTimer;

private Action function = null;

/// <summary>

/// Запуск таймера.

/// </summary>

/// <param name="mlSecond"> Время в миллисекундах. </param>

/// <param name="function"> Вызывающийся метод. </param>

public void Start(int mlSecond, Action function)

{

if (mlSecond < 100)

{

throw new ArgumentException($"Таймер со скоростью {mlSecond} мс не используется", nameof(mlSecond));

}

if (function == null)

{

throw new ArgumentNullException("Ссылка на метод является пустой", nameof(function));

}

this.function = function;

aTimer = new Timer

{

Interval = mlSecond

};

aTimer.Elapsed += OnTimedEvent;

aTimer.AutoReset = true;

aTimer.Enabled = true;

}

/// <summary>

/// Остановка таймера.

/// </summary>

public void Stop()

{

aTimer.Enabled = false;

}

/// <summary>

/// Вызов метода.

/// </summary>

/// <param name="e"> Данные для события. </param>

private void OnTimedEvent(Object source, ElapsedEventArgs e)

{

function();

}

}

}

Листинг класса Background.cs:

using GameEngine;

namespace GameLogic

{

public class Background : GameObject { }

}

Листинг класса Border.cs:

using System;

using OpenTK;

using GameEngine;

namespace GameLogic

{

/// <summary>

/// Граница.

/// </summary>

public class Border : GameObject

{

private int width;

private int height;

Direction direction;

/// <summary>

/// Инициализатор границы.

/// </summary>

/// <param name="width"> Ширина. </param>

/// <param name="height"> Высота. </param>

public Border(int width, int height, Direction direction)

{

if (width <= 0)

{

throw new ArgumentException($"Ширина границы не может быть {width}", nameof(width));

}

if (height <= 0)

{

throw new ArgumentException($"Высота границы не может быть {height}", nameof(height));

}

this.width = width;

this.height = height;

this.direction = direction;

Components.Add(new BoxCollider(width, height));

AddProperties();

}

/// <summary>

/// Настройка позиции границы.

/// </summary>

private void AddProperties()

{

int x = width;

int y = height;

switch (direction)

{

case Direction.Up:

x = 0;

break;

case Direction.Down:

x = 0;

y = y \* -1;

break;

case Direction.Left:

x = x \* -1;

y = 0;

break;

case Direction.Right:

y = 0;

break;

default:

break;

}

Transform.Position = new Vector2(x, y);

}

}

}

Листинг класса Rocket.cs:

using System;

using OpenTK;

using GameEngine;

namespace GameLogic

{

/// <summary>

/// Ракета.

/// </summary>

public abstract class Rocket : GameObject, IMovable

{

/// <summary>

/// Скорость.

/// </summary>

public float Speed { get; }

/// <summary>

/// Урон.

/// </summary>

public int LifeDamage { get; }

/// <summary>

/// Бронепробиваемость.

/// </summary>

public int ArmorDamage { get; }

/// <summary>

/// Скорострельность.

/// </summary>

public int FiringRate { get; }

/// <summary>

/// Направление ракеты.

/// </summary>

public Direction Direction { get; }

/// <summary>

/// Хозяин ракеты.

/// </summary>

public Type Owner { get; }

/// <summary>

/// Инициализатор ракеты.

/// </summary>

/// <param name="speed"> Скорость. </param>

/// <param name="lifeDamage"> Урон по жизням. </param>

/// <param name="armorDamage"> Бронепробиваемость. </param>

/// <param name="firingRate"> Скорострельность </param>

/// <param name="direction"> Направление. </param>

/// <param name="owner"> Тип стреляющей лодки. </param>

public Rocket(float speed, int lifeDamage, int armorDamage, int firingRate, Direction direction, Type owner)

{

Speed = speed;

LifeDamage = lifeDamage;

ArmorDamage = armorDamage;

FiringRate = firingRate;

Direction = direction;

Owner = owner;

}

/// <summary>

/// Движение ракеты.

/// </summary>

/// <param name="time"> Время. </param>

public void Move(double time)

{

float x = Transform.Position.X;

float y = Transform.Position.Y;

if (Direction == Direction.Right)

{

x += (float)(Math.Pow(Speed, 2) \* time);

}

if (Direction == Direction.Left)

{

x -= (float)(Math.Pow(Speed, 2) \* time);

}

Transform.Position = new Vector2(x, y);

}

}

}

Листинг класса FieryRocket.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Огненная ракета.

/// </summary>

public class FieryRocket : Rocket

{

public FieryRocket(Direction direction, Type owner) : base(40, 45, 35, 750, direction, owner) { }

}

}

Листинг класса IceRocket.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Ледяная ракета.

/// </summary>

public class IceRocket : Rocket

{

public IceRocket(Direction direction, Type owner) : base(30, 60, 55, 1000, direction, owner) { }

}

}

Листинг класса AtomicRocket.cs:

using System;

namespace GameLogic

{

/// <summary>

/// Атомная ракета.

/// </summary>

public class AtomicRocket : Rocket

{

public AtomicRocket(Direction direction, Type owner) : base(25, 80, 60, 1250, direction, owner) { }

}

}

Листинг класса Decorator\_Tests.cs:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using GameLogic;

namespace AppVerification

{

/// <summary>

/// Тестирование классов декоратора.

/// </summary>

[TestClass()]

public class Decorator\_Tests

{

/// <summary>

/// Тестирование класса декоратора с изменением жизней лодки.

/// </summary>

[TestMethod()]

public void AdditionalHealth\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

// Act.

var decoratedSubmarine = new AdditionalHealth(submarine);

var difference = decoratedSubmarine.BoostHealth;

// Assert.

Assert.AreEqual(decoratedSubmarine.Health, submarine.Health + difference);

}

/// <summary>

/// Тестирование класса декоратора с изменением брони лодки.

/// </summary>

[TestMethod()]

public void AdditionalArmor\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

// Act.

var decoratedSubmarine = new AdditionalArmor(submarine);

var difference = decoratedSubmarine.BoostArmor;

// Assert.

Assert.AreEqual(decoratedSubmarine.Armor, submarine.Armor + difference);

}

/// <summary>

/// Тестирование класса декоратора с изменением скорости лодки.

/// </summary>

[TestMethod()]

public void AdditionalSpeed\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

// Act.

var decoratedSubmarine = new AdditionalSpeed(submarine);

var difference = decoratedSubmarine.BoostSpeed;

// Assert.

Assert.AreEqual(decoratedSubmarine.Speed, submarine.Speed + difference);

}

/// <summary>

/// Тестирование класса декоратора с изменением боеприпасов лодки.

/// </summary>

[TestMethod()]

public void AdditionalAmmunition\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

// Act.

var decoratedSubmarine = new AdditionalAmmunition(submarine);

var difference = decoratedSubmarine.BoostAmmunition;

// Assert.

Assert.AreEqual(decoratedSubmarine.Ammunition.Count(), submarine.Ammunition.Count() + difference);

}

}

}

Листинг класса FactoryMethod\_Tests.cs:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using GameLogic;

namespace AppVerification

{

/// <summary>

/// Тестирование классов фабричного метода.

/// </summary>

[TestClass()]

public class FactoryMethod\_Tests

{

/// <summary>

/// Создание бонуса здоровья.

/// </summary>

[TestMethod()]

public void HealthBonus\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

var expectedsubmarine = new AdditionalHealth(submarine);

// Act.

var creatorHealth = new CreatorHealth();

var healthBonus = creatorHealth.CreateBonus();

var newSubmarine = healthBonus.Activation(submarine);

// Assert.

Assert.AreEqual(newSubmarine.GetType(), expectedsubmarine.GetType());

}

/// <summary>

/// Создание бонуса брони.

/// </summary>

[TestMethod()]

public void ArmorBonus\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

var expectedsubmarine = new AdditionalArmor(submarine);

// Act.

var creatorArmor = new CreatorArmor();

var armorBonus = creatorArmor.CreateBonus();

var newSubmarine = armorBonus.Activation(submarine);

// Assert.

Assert.AreEqual(newSubmarine.GetType(), expectedsubmarine.GetType());

}

/// <summary>

/// Создание бонуса скорости.

/// </summary>

[TestMethod()]

public void SpeedBonus\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

var expectedsubmarine = new AdditionalSpeed(submarine);

// Act.

var creatorSpeed = new CreatorSpeed();

var speedBonus = creatorSpeed.CreateBonus();

var newSubmarine = speedBonus.Activation(submarine);

// Assert.

Assert.AreEqual(newSubmarine.GetType(), expectedsubmarine.GetType());

}

/// <summary>

/// Создание бонуса боеприпасов.

/// </summary>

[TestMethod()]

public void AmmunitionBonus\_Test()

{

// Arrang.

var submarine = new BlueSubmarine(new RocketAmmunition());

var expectedsubmarine = new AdditionalAmmunition(submarine);

// Act.

var creatorAmmo = new CreatorAmmunition();

var ammoBonus = creatorAmmo.CreateBonus();

var newSubmarine = ammoBonus.Activation(submarine);

// Assert.

Assert.AreEqual(newSubmarine.GetType(), expectedsubmarine.GetType());

}

}

}

Листинг класса Collider\_Tests.cs:

using Microsoft.VisualStudio.TestTools.UnitTesting;

using GameLogic;

using GameEngine;

using OpenTK;

using System;

namespace AppVerification

{

/// <summary>

/// Тестирование коллайдера.

/// </summary>

[TestClass()]

public class Collider\_Tests

{

/// <summary>

/// Тест на проверку столкновений объектов, одинаковое положение объектов.

/// </summary>

[TestMethod()]

public void CollisionTrue\_Test\_SamePosition()

{

// Arrang.

var position = new Vector2(10, 10);

bool expected = true;

var obj1 = new BlueSubmarine(new RocketAmmunition());

obj1.Components.Add(new BoxCollider(100, 100));

obj1.Transform.Position = position;

var obj2 = new RedSubmarine(new RocketAmmunition());

obj2.Components.Add(new BoxCollider(100, 100));

obj2.Transform.Position = position;

// Act.

var actual = Collider.CheckCollision(obj1, obj2);

// Assert.

Assert.AreEqual(actual, expected);

}

/// <summary>

/// Тест на проверку столкновений объектов, рандомное положение объектов.

/// </summary>

[TestMethod()]

public void CollisionTrue\_Test\_RandomPosition()

{

// Arrang.

Random rn = new Random();

int limit = 1000;

var obj1 = new BlueSubmarine(new RocketAmmunition());

obj1.Components.Add(new BoxCollider(100, 100));

obj1.Transform.Position = new Vector2(rn.Next(-limit, limit), rn.Next(-limit, limit));

var obj2 = new RedSubmarine(new RocketAmmunition());

obj2.Components.Add(new BoxCollider(100, 100));

obj2.Transform.Position = new Vector2(rn.Next(-limit, limit), rn.Next(-limit, limit));

bool expected = (obj1.Transform.Position == obj2.Transform.Position) ? (true) : (false);

// Act.

var actual = Collider.CheckCollision(obj1, obj2);

// Assert.

Assert.AreEqual(actual, expected);

}

}

}

Листинг класса Submarine\_Tests.cs:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using GameLogic;

namespace AppVerification

{

/// <summary>

/// Тестирование методов стрельбы и получения урона у лодки.

/// </summary>

[TestClass()]

public class Submarine\_Tests

{

/// <summary>

/// Тестирование метода TakingDamage класса Submarine, который отвечает за получение урона лодкой.

/// </summary>

[TestMethod()]

public void TakingDamage\_Test\_RandomValue()

{

// Arrang.

var rn = new Random();

var submarine = new BlueSubmarine(new RocketAmmunition());

var testArmor = submarine.Armor;

var healthDamage = rn.Next(0, 100);

var armorDamage = rn.Next(0, 100);

// Act.

submarine.TakingDamage(healthDamage, armorDamage);

testArmor -= armorDamage;

// Assert.

Assert.AreEqual(submarine.Armor, testArmor);

}

/// <summary>

/// Тестирование метода Shoot класса Submarine, который отвечает за выстрел лодки.

/// После выстрела ракетой определённого типа, подсчитываем количество ракет данного типа.

/// </summary>

[TestMethod()]

public void Shoot\_Test()

{

// Arrang.

int limitRocket = 10;

var submarine = new BlueSubmarine(new RocketAmmunition());

submarine.Ammunition.AddRockets(RocketType.FieryRocket, limitRocket);

submarine.Ammunition.AddRockets(RocketType.IceRocket, limitRocket);

submarine.Ammunition.AddRockets(RocketType.AtomicRocket, limitRocket);

var rocketType = RocketType.AtomicRocket;

var atomicRocketCount = submarine.Ammunition.GetRockets(rocketType);

// Act.

submarine.Shoot(rocketType);

atomicRocketCount--;

// Assert.

Assert.AreEqual(submarine.Ammunition.GetRockets(rocketType), atomicRocketCount);

}

}

}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Архитектура паттерна «Декоратор»**

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Руководство программиста**

1. Назначение и условия применения.

Разработанное программное приложение предназначено для организации игрового процесса между двумя игроками. Игровое приложение «Сражение подводных лодок» предполагает наличие двух игроков, которые соревнуются в том, кто быстрее уничтожит вражескую лодку. Управление лодками осуществляется с помощью клавиатуры.

Для корректной работы программного средства необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие стандартной клавиатуры;
* наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

1. Характеристики программы.

Игровой процесс реализуется в одном окне.

1. Обращение к программе.

Приложение запускается путем открытия файла *BattleOfTheSubmarinesGames.exe*, находящимся в каталоге *bin/Debag*.

1. Выходные данные.

Выходной информацией является экран с результатом игрового процесса.

1. Сообщения.

По завершению игры на экране появляется сообщение о том, кто выиграл.

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство системного программиста**

1. Общие сведения о программе.

Разработанное программное приложение предназначено для организации игрового процесса между двумя игроками. Игровое приложение «Сражение подводных лодок» предполагает наличие двух игроков, которые соревнуются в том, кто быстрее уничтожит вражескую лодку. Управление лодками осуществляется с помощью клавиатуры.

Для корректной работы программного средства необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows XP* и выше;
* наличие стандартной клавиатуры;
* наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

1. Структура программы.

Игровой процесс происходит в одном окне, где отображаются все игровые объекты.

1. Настройка программы

Настройка программы осуществляется путем запуска исполняемого файла.

1. Проверка программы.

Отсутствие сообщения об ошибках свидетельствует, о корректной работе приложения.

1. Дополнительные возможности.

Дополнительных возможностей программа не имеет.

1. Сообщения системному программисту.

Игровой процесс происходит в одном окне сразу после запуска приложения.

**ПРИЛОЖЕНИЕ Д**

(обязательное)

**Руководство пользователя**

1. Введение.

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

Разработанное программное приложение предназначено для организации игрового процесса между двумя игроками. Игровое приложение «Сражение подводных лодок» предполагает наличие двух игроков, которые соревнуются в том, кто быстрее уничтожит вражескую лодку. Управление лодками осуществляется с помощью клавиатуры.

Программное средство обладает следующим функционалом:

* наличие счёта;
* игровое поле;
* управление игроком при помощи клавиатуры;
* наличие призов.

Ход использования приложения.

Для использования программного средство пользователь должен быть ознакомлен с:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

1. Назначение и условия применения.

Разработанное программное приложение предназначеное для сражений подводных лодок.

Для корректной работы программного средства необходимо соблюдение следующих требований:

* поддерживаемые операционные системы *Windows* *XP* и выше;
* наличие стандартной клавиатуры;
* наличие следующих устройств вывода: экран, подключаемый по *HDMI*.

1. Подготовка к работе.

Для установки приложения необходимо загрузить на компьютер каталог BattleOfTheSubmarinesGames. После этого в папке с программой зайти в каталог bin/Debag и запустить файл BattleOfTheSubmarinesGames.exe.

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

1. Описание операций.

В ходе разработки игрового приложения были реализовано окно игрового поля, на котором происходит основное действие игры.

Игровой процесс начинается сразу после запуска игры и нажатия кнопки «Начать игру».

Игровое пространство реализовано в виде сцены.

Раскладка клавиатуры, отвечающая за управление подводными лодками демонстрируется в таблице 1.

Таблица 1 – Раскладка клавиатуры

|  |  |  |
| --- | --- | --- |
|  | Синяя лодка | Красная лодка |
| Движение вверх | W | Up |
| Движение вниз | S | Down |
| Движение влево | A | Left |
| Движение вправо | D | Right |
| Переключение на огненную ракету | Number1 | Keypad1 |
| Переключение на ледяную ракету | Number2 | Keypad2 |
| Переключение на атомную ракету | Number3 | Keypad2 |
| Выстрел | Space | KeypadEnter |

Аварийные ситуации.

Чтобы избежать ошибок при использовании данного игрового приложения, необходимо соблюдать порядок действий и условия использования, описанные в главе 3 данного руководства пользователя.

В случае непредвиденного «зависания» программы рекомендуется завершить процесс в диспетчере задач и запустить снова. Или можно подождать некоторое время, так как существует возможность зависания программы за счет наличия библиотек, используемых в игровом приложении.

1. Рекомендации по освоению.

Игровое приложение рекомендуется запускать при наличии двух игроков для правильного функционирования.