**СОДЕРЖАНИЕ**

[Введение 4](#_Toc72829759)

[1 Игровые приложения и средства их разработки 5](#_Toc72829760)

[1.1 Графическая библиотека OpenGL 5](#_Toc72829761)

[1.2 Графическая библиотека DirectX 6](#_Toc72829762)

[1.3 Результат сравнения графических библиотек OpenGL и DirectX 7](#_Toc72829763)

[1.4 Технология WPF, как средство реализации игрового приложения 8](#_Toc72829764)

[1.5 Игровой жанр «Симулятор морских сражений» 9](#_Toc72829765)

[1.6 Игры жанра «Симулятор морских сражений» 9](#_Toc72829766)

[1.7 Актуальность компьютерных игр в настоящее время 11](#_Toc72829767)

[2 Программная реализация приложения «Морской бой» 14](#_Toc72829768)

[2.1 Основные элементы игрового приложения «Морской бой» 14](#_Toc72829769)

2.2 Архитектура игрового приложения «Морской бой»…………………..14

[2.3 Структура игрового приложения «Морской бой» 14](#_Toc72829770)

[2.4 Принципы взаимодействия классов приложения 19](#_Toc72829771)

[3 Верификация и апробация игрового приложения «Морской бой» 23](#_Toc72829772)

[3.1 Принцип работы игрового приложения 23](#_Toc72829773)

[3.2 Тестирование приложения 24](#_Toc72829774)

[3.3 Пользовательский интерфейс 25](#_Toc72829775)

[Заключение 30](#_Toc72829776)

[Список используемых источников 31](#_Toc72829777)

[Приложение А Схема использования паттерна «Декоратор» 32](#_Toc72829778)

[Приложение Б Листинг программы «Морской бой» 33](#_Toc72829779)

[Приложение В Руководство пользователя 91](#_Toc72829780)

[Приложение Г Руководство программиста 93](#_Toc72829781)

[Приложение Д Руководство системного программиста 94](#_Toc72829782)

## ВВЕДЕНИЕ

В современном мире наличие своего персонального копмьютера является чуть ли не обязательным, а возможности этих устройств уже сполна покрывают все требования пользователй. Одним из наиболее популярных применений компьютерной техники в настоящие время являются компьютерные видеоигры. Они плотно вошли в повседневную жизнь многих людей и стали крупной самостоятельной отраслью со своими специфическими особенностями.

Высокие темпы развития игровой индустрии обусловлены прежде всего наличием большого потребительского спроса на различные игровые решения, а растущая заинтересованность пользователй в игровых продуктах породила большое мноогобразие игровых жанров. Каждый игровой жанр в отдельности представляет собой сложную систему, для которой свойственны отличия в таких аспектах, как сложность разработки, охват игроков и время производства игры.

Игровое приложение «Морской бой» является игрой в жанре симулятор морских сражений. Родоначальником данного жанра был симулятор танковых сражений, посвященный управлению боевым танком. История игр такого жанра начинается ещё в 1985 году. Целью таких игр является точное попадание по определённым объектам.

Об актуальности разработки нового игрового приложения в описываемом жанре свидетельствует большое количество положительных статистических данных. Симулятор танковых сражений плотно заняла свою нишу и имеет большое количество поклонников.

Целью текущего проекта является создание компьютерной игры в жанре танкового сражения, которая представляет собой реализацию основных аспектов выбранного игрового жанра, а также соотвествует всем современным пользовательским требованиям в отношении использования конечного продукта.

Важным задачами при разработке игры является: формулирование идеи и разработка концепта игры, созданние первого рабочего прототипа, получение минимально возможную полноценную версию игры, заполнение проекта игровым контентом.

На основе сказанного можно сделать вывод, что появление игр в жанре симулятора танковых сражений на сегодняшний момент соотвестсвует современным тенденциям в сфере компьютерных игр, а их разработка является актульной и востребованной среди широкой аудитории.

## ИГРОВЫЕ ПРИЛОЖЕНИЯ И СРЕДСТВА ИХ РАЗРАБОТКИ

## 1.1 Графическая библиотека *OpenGL*

*OpenGL* – спецификация, определяющая независимый от языка программирования программный интерфейс для написания приложений, использующих 2D и 3D компьютерную графику. Недостатки: определенная сложность освоения данной технологии, сложность интеграции с некоторыми языками программирования и средствами разработки. Достоинства: кроссплатформенность, высокая производительность, огромные возможности в плане анимации, а также работы с двухмерной и трехмерной графикой. Возможное использование в разработке игр: предназначена для разработки игр на разных платформах. Целевые платформы: разные [4, c. 7].

Спецификация *OpenGL* составляется соглашением *ARB (Architecture Review Board)*, который был сформирован в 1992 году компаниями *SGL, IBM, Intel, Microsoft* и *Digital Equipment Corporation*. Сейчас в него входят такие компании, как *NVIDIA, AMD, Intel, Apple* и другие.

Функции спецификации *OpenGL* реализуются производителями видеокарт, согласно спецификации *ARB*. Прежде чем продукт может быть классифицирован как *OpenGL*-совместимый, он должен протестироваться. Если оборудование не поддерживает какую-либо функцию из спецификации, то она должна быть эмулирована программно.

 Главная цель *OpenGL* – задание механизма описания процесса. визуализации сложных объектов, а не способов их построения. Например, в стандарт не включены средства построения невыпуклых многоугольников, так как алгоритмы их визуализации намного сложнее, чем те, которые используются для выпуклых.

Схема работы *OpenGL* представлена на рисунке 1.1.

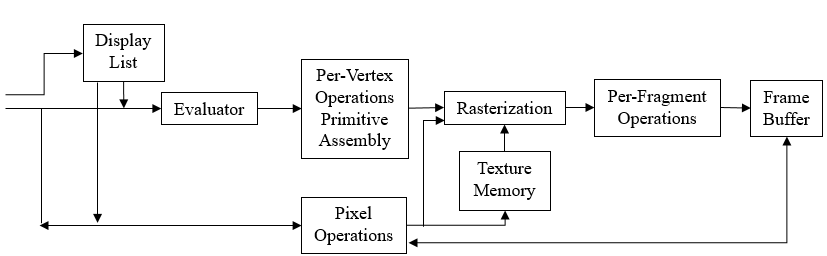
******

Рисунок 1.1 – Схема работы *OpenGL*

## 1.2 Графическая библиотека *DirectX*

*DirectX* – набор *API*, разработанных для программирования графики под *Microsoft Windows* [1, с. 26]. Достоинства: высокая производительность, огромные возможности в плане анимации, а также работы с *2D* и *3D* графикой. Возможное использование в разработке игр: предназначена для разработки игр на разных платформах. Целевые платформы: основная – *Windows*, но возможна реализация игр и под другие [4, c. 7].

*DirectX* основан на идее графического конвейера, как и *OpenGL* [2, с. 56]. Программист может настроить отдельные этапы этого конвейера, а передать ему входные данные. Структура *DirectX* представлена на рисунке 1.2.



Рисунок 2.2 – Структура *DirectX*

*DirectX* реализует модель *COM (Component Object Mode)*, что позволяет использовать его с любым языком, который поддерживает эту модель.

В целом *DirectX* делится на:

* *Direct2D* – интерфейс вывода двухмерной графики;
* *Graphics* – вывод графики. Этот компонент включает в себя функции обработки как *3D*, так и *2D* графики, а также за инициализацию и установку разрешения;
* *DirectInput* – отслеживание ввода пользователя. Поддерживаются различные девайсы, такие как мышь, клавиатура, джойстик или геймпад;
* *DirectPlay* – интерфейс сетевой коммуникации игр;
* *DirectSound* – работа со звуком;
* *DirectMusic* – воспроизведение фоновой музыки;
* *DirectShow* – ввод/вывод аудио или видео данных;
* *DirectSetup* – функционал для установки *DirectX* последней версии на компьютер пользователя.

## 1.3 Результат сравнения графических библиотек *OpenGl* и *DirectX*

Основные отличия библиотек приведены в таблице 1.1.

Таблица 1.1 – Сравнение *OpenGL* и *DirectX*

|  |  |  |
| --- | --- | --- |
|  | *OpenGL* | *DirectX* |
| Кроссплатформенность | Библиотеку *OpenGL* можно использовать на большинстве операционных сиcтем | Разрабатывается компанией *Microsoft* для операционой системы *Windows* |
| Влияние версий библиотеки | Слабая привязка приложений к версии *OpenGL*. Не важно какая версия *OpenGL* стоит на компьютере пользователя, главное, чтобы она поддерживала все необходимые приложению расширения | У DirectX каждая версия имеет набор жёстко заданных функций. Часто для перехода на новую версию требуется полностью переписать код приложения |
| Сложность установки | Драйвер *OpenGL* придётся устанавливать самостоятельно | Драйвер *DirectX* поставляется вместе с системой *Windows* и в большинстве случаев пользователю не нужно что-либо устанавливать |

Как можно заметить из таблицы, обе библиотеки имеют некоторые различия, но нельзя назвать одну из них лучше другой. Выбор необходимо осуществлять исходя из требований к вашему приложению.

В разрабатываемом приложении выберем *OpenGL*, реализованный через *OpenTK*, так как этого требует задание.

Для поставленной задачи требуется использовать *OpenGL*, и для реализации графики *OpenGL* в курсовом проекте будем пользоваться *Open Toolkit*.

*Open Toolkit* - это продвинутая кроссплатформенная оболочка *C# OpenGL*, *OpenAL* и *OpenCL* для *Mono / .Net*. Он особенно подходит для разработки *RAD* и может использоваться в играх, графических интерфейсах *(WinForms, WPF, GTK #)* и научных приложениях.

*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*.

Для нашего проекта будем использовать *OpenTK* 3.1.0.

## 1.4 Технология *WPF,* как средство реализации игрового приложения

*Windows Presentation Foundation (WPF)* – система для построения клиентских приложений *Windows* с визуально привлекательными возможностями взаимодействия с пользователем, графическая (презентационная) подсистема в составе *.NET Framework* (начиная с версии 3.0), использующая язык *XAML* [4, c. 7]. Технология *Windows Presentation Foundation (WPF)* является частью платформы *.NET* и представляет собой подсистему для проектирования графических интерфейсов. Используя *WPF*, Значительная часть работы по отрисовке графики ложиться на графический процессор видеокарты. В своей основе технология *WPF* содержит язык разметки на базе *XAM*. В отдельных случаях на *XAML* можно написать целую программу, однако чаще приложения строятся из программного кода и из кода разметки. Обычно *XAML* используется для изготовления визуального оформления приложения. Можно создавать красивый графический интерфейс, используя код на управляемых языках *C#* и *VB.NET*, или декларативное объявление интерфейса, либо совмещать тот и другой способ. Код обработки событий пишется отдельно.

К основным достоинствам языка можно отнести:

– использование языков платформы .*NET* – *VB.NET* и *C#* для создания логики приложения;

– возможность декларативного определения графического интерфейса с помощью *XAML*;

– качество внешнего вида не зависит от разрешения экрана;

– по сравнению с *Windows Forms* упрощено: привязка данных, создание трехмерных моделей, использование таких элементов, как стили, шаблоны, темы и др;

– возможность взаимодействовать с *Windows Forms*, благодаря чему, в приложениях *WPF* можно использовать традиционные элементы управления из *Windows Forms*;

– аппаратное ускорение графики за счет графического процессора

– видеокарты – вне зависимости от того, работаете ли вы с *2D* или *3D*;

– создание приложений под ОС семейства *Windows XP*/7/8/10.

## Игровой жанр «Симулятор морских сражений»

Симулятор морских сражений – жанр компьютерных игр, разновидность симуляторов, посвященная управлению боевым кораблями.

Игровой процесс таких игр заключается, прежде всего в управлении кораблём, которое осуществляется при помощи клавиатуры (в случае с играми, предназначенными для персонального компьютера) или геймпада (игры, созданные для игровых приставок). Нередко имеются тактические элементы, например, командование собственным подразделением (с помощью назначенных клавиш игрок отдает команды вида «следовать за мной», «атаковать ближайшую цель», «остановиться» и т.д).

Задания обычно заключаются в уничтожении вражеских кораблей и выдаются игроку в начале каждого эпизода. В некоторых играх задания проходят как в дневное, так и в ночное время суток. Также может присутствовать возможность разрушаемости окружения, возникающая впоследствии выстрелов. В некоторых реалистичных симуляторах разрешено принимать различные специальности в экипаже корабля (в аркадных симуляторах игрок обычно держит контроль над всем транспортным средством).

В играх этого жанра игроку предоставляется обзор из кабины корабля, в иных случаях – виртуальная камера находится над кораблём.

## 1.6 Игры жанра «Симулятор морских сражений»

Родоначальником жанра был жанр с похожим названием – симулятор танковых сражений. Первая игра про танковые сражения является *Battle City* в 1985 году. Полигон действий виден сверху. Игрок должен, управляя своим танком, уничтожить все вражеские танки. Данная игра изображена на рисунке 1.3.



Рисунок 1.3 – Игра *Battle City*

*World of Tanks* – самая популярная онлайн игра в жанре аркадного танкового симулятора, разработаная белорусской студией *Wargaming*, выпущенная в 2010 году. Концепция *World of Tanks* базируется на командных танковых сражениях игроков между собой (*PvP*), в которых каждый игрок управляет бронированной боевой машиной. Игра сделана с использованием 3D графики и управлением танка от третьего лица. Данная игра изображена на рисунке 1.4.



Рисунок 1.4 – игра *World of Tanks*

В 2015 году студией *Wargaming*, после успеха своего проекта *World of Tanks*, выпускает игру про морские бои в игровом жанре «симулятор морских сражений» под названием *World of Warships*, где под управлением игрока находится корабль, у которого есть три типа вооружения: корабельные орудия, торпеды и авиация. Команда игрока может сражаться против других игроков (*PvP*). Данная игра изображена на рисунке 1.5.



Рисунок 1.4 – игра *World of Warships*

## 1.7 Актуальность компьютерных игр в настоящее время

Сегодня индустрия компьютерных игр представляет собой быстро развивающийся сектор глобальной экономики, связанный с разработкой, продвижением и продажей игр. А сами игры теперь позиционируются и воспринимаются пользователями как качественное многожанровое интерактивное развлечение, которое постепенно становится все популярнее.

По мнению экспертов, именно игровая индустрия становится проводником идей современного творчества и новаторства. Компьютерные игры сейчас присутствуют в различных сегментах массовой культуры, а некоторые даже стали ее феноменами (например, *Mario* или *Final Fantasy*). В некоторых странах, например в США, компьютерные игры были признаны отдельным видом искусства. Кроме того, было положено начало новой субкультуре со своим языком, иконами, уникальными событиями и мероприятиями. Формируется мощная экосистема. Гейминг становится высокооплачиваемой профессией.

Эксперты и участники отрасли отмечают развитие ряда сильных тенденций, которые определяют развитие глобальной игровой индустрии. Так, большую роль продолжают играть новые технологии (виртуальные, мобильные, облачные и др.), наблюдается слияние виртуальных и физических сред, социализация игр и т.д. И такая популярность неизбежно ведёт к увеличению денежного оборота в индустрии. Всё это является весьма весомым доводом для разработки игр как для крупных компаний, так и для инди разработчиков или небольших групп энтузиастов.

В настоящий момент наиболее прибыльными являются мобильные платформы, в связи с их относительной новизной [3]. На втором месте находятся консоли, а на третьем – ПК (рисунок 1.7).

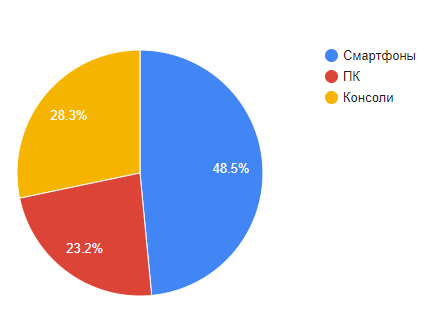


Рисунок 1.5 – Диаграмма прибыли индустрии на 2020 год

Сегодня на игровом рынке можно выделить две ключевые категории участников – это непосредственно разработчики компьютерных игр и компании-издатели. Разработчиками компьютерных игр являются *IT*-компании разного масштаба, специализирующиеся на создании игрового контента. При этом разработчиком может быть как единственный человек, так и большое предприятие с разделением обязанностей между сотрудниками по областям деятельности – программирование, дизайн, художественная составляющая, тестирование и т.д.

Крупные компании-разработчики с большим бюджетом, сами контролируют свою деятельность. Но большинство компаний-разработчиков получают финансовую и/или маркетинговую поддержку со стороны компаний-издателей. В этом случае, как правило, разработчики вынуждены упрощать свои игры в целях ее трансформации в более массовый продукт. Также существуют другая категория разработчиков, которые реализуют свои креативные проекты за счет собственных ресурсов или используют инструменты коллективных финансов – независимые или инди-разработчики.

Таким образом можно отметить, что разработка игр в данный момент является очень перспективным направлением разработки ПО и будет оставаться таковым ещё продолжительное время. Но не смотря на финансовую прибыль, которую несёт разработка игр, стоит помнить, что это в первую очередь искусство, и если в игре нет продуманного сюжета или чего-то, что способно захватить игроков, заставить их полюбить игру, то не стоит рассчитывать на большую финансовую прибыль.

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «МОРСКОЙ БОЙ»

## Основные элементы игрового приложения «Морской бой»

В процессе разработки игрового приложения «Морской бой» необходимо реализовать основные возможности игр такого жанра. Игровое поле будет представлять карту с непроходимыми скалами и мелководьями, замедляющие корабли, которые замедляют игроков. В пределах карты два пользователя на своих кораблях учувствуют в перестрелке против другу друга. В верхних углах экрана игрового пространства располагаются характеристики игроков, такие как количество боеприпасов и очки здоровья. В начале игры игроки должны закупиться снарядами и торпедами на определённую сумму, после чего начинается игра. Проигравшим является тот игрок, который был уничтожен другим игроком. Так же проигравшим можно считать игрока, у которого закончились боеприпасы.

Для реализации приложения необходимо использовать язык программирования *C#*, технологию *Windows Presentation Foundation (WPF)* и шаблоны проектирования:

* «Фабричный метод» – для реализации генерации снарядов и торпед;
* «Декоратор» – для изменения характеристик корабля и для генерации торпед с различными пересекающими характеристиками.

Для отображения игровых объектов необходимо использовать спрайтовую графику и средства графической библиотеки *OpenGL*.

* 1. **Архитектура игрового приложения «Морской бой»**

При проектировании архитектуры игрового приложения метод декомпозиции используется для разделения основной задачи на модули, которые, в свою очередь, представляют собой небольшие программы, взаимодействующие друг с другом. С помощью методов декомпозиции определяется так называемая иерархическая структура работы. Эта структура делит проект на иерархически связанные, управляемые и контролируемые подзадачи.

Для реализации игровой логики необходимо создать игровой движок. Основные функции, которые игровой движок предоставляет для решения текущей задачи, включают рендеринг (реализация 2*D*-графики), физику (обнаружение столкновений), анимацию, управление памятью, потоки, поддержку локализации и сцену. Исходя из вышеизложенного, необходимо создать проект, разделяющий центральные компоненты игры, такие как окно отображения и его настройки, управление, осуществляемое вводом с клавиатуры, игровые объекты, имеющие определенный набор компонентов для отображения, и правильное поведение на сцене, исходя из ее логики. Проект движка определяется библиотекой классов и не имеет графические компоненты для визуальной разработки, содержит технологии перемещения по вертикали и горизонтали, отображения графики и анимации игры, обнаружения столкновений.

Для отображения статистики необходимо интегрировать её в проект *WPF* и создать элементы для отображения количества боеприпасов и здоровья игроков.

Игровое приложение «Морской бой» содержать в себе графику и весь необходимый функционал для обработки событий, что будут происходить на игровой сцене. В результате должно быть разработано игровое приложение, которое будет выполнять следующий функционал: игровое меню с покупкой снарядов и торпед, отображение игровых объектов, перемещение игроков по игровому полю, регистрация попаданий снарядов в другие объекты, завершение игры после уничтожения техники одного из игроков и вывод сообщения об этом, возможность начать новую игру после завершения игровой партии (после завершения игры вернуться в главное меню). На рисунке 2.1 представлена обобщённая схема функционирования разрабатываемого приложения.

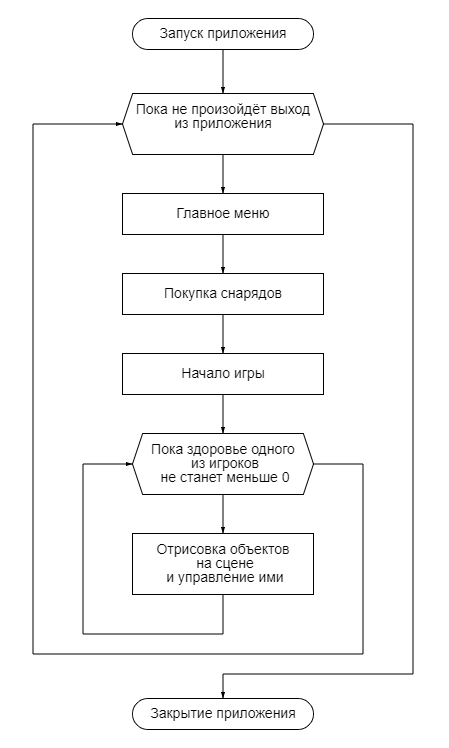


Рисунок 2.1 – Обобщённая схема работы приложения

## Структура игрового приложения «Морской бой»

Основу любого проекта информационной системы составляют методологии проектирования. Перед тем как начать разработку, необходимо выб-рать – какой методологией предстоит пользоваться. Сделанный выбор определит дальнейший процесс разработки. Под методологией понимается выбор структуры, способов реализации, логической организации, методов и средств, необходимых для создания еще не существующей системы, которые обеспечивают выполнение процессов жизненного цикла. Процесс проектирования рассматривается как последовательность технологических операций.

На первом этапе проектирования выделяются основные сущности игрового мира, которые будут присутствовать на игровом поле. Далее идет проектирование иерархии этих сущностей. После этого следует определить функциональность данных классов, описывая соответствующие методы. Сам объект игрового поля содержит методы, описывающие физическую модель игры, пользовательского ввода, отрисовки объектов.

В данном курсовом проекте основной сущностью, которая описывает поведение всех игровых объектов, является класс *GameObject*. Экземпляр этого класса хранит список всех объектов, находящихся на сцене, описывает их поведение во время игры, определяет физическую модель игры.

Данная модель позволяет упростить разработку основных элементов приложения, за счет повторного использования кода, благодаря механизму наследования и полиморфизма.

Из всех игровых объектов свойство передвижения имеют лишь игроки и снаряды. Перемещение игроков будет происходить по горизонтали и вертикали. Каждый кадр будет происходить проверка, на доступность передвижения игрока. При столкновении игрока с объектом (скалы или вражеский корабль) на игровом поле будет происходить блокировка движения в заданном направлении.

Другой объект, имеющий способность передвигаться, это боеприпасы корабля: торпеды и снаряды. Передвижение боеприпасов происходит по прямой траектории. Боеприпасы при запуске имеют статическую скорость, которая уникальна для каждого вида боеприпаса. Во время полёта боеприпаса каждый новый кадр будет происходить проверка доступности движения боеприпаса и всех объектов, что будут находиться на сцене. При обнаружении коллизии боеприпаса и игрока, будет происходить нанесения урона той технике, с которой произошло столкновение. боеприпас уничтожается при взаимодействии с одним из объектов. боеприпас, который был выпущен игроком, может нанести урон своей технике, так как имеет область поражения.

Паттерны или шаблоны проектирования позволяют оптимально решать часто встречающиеся проблемы проектировании различного рода программ. При написании программ проблема формализуется в виде классов, объектов и связей между ними, и далее применить один из существующих паттернов для решения данной проблемы. Существует множество различных паттернов, которые решают разные задачи проектирования. В данном курсовом проекте используются порождающий и структурный паттерны проектирования [5, с. 138].

Фабричный метод (*Factory Method*) представляет собой 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 [6, c. 215].

Данный паттерн применяется тогда, когда:

* заранее неизвестно, объекты каких типов необходимо создавать;
* система должна быть независимой от процесса создания новых объектов и расширяемой: в неё можно легко вводить новые классы, объекты которых система должна создавать;
* создание новых объектов необходимо делегировать из базового класса классам наследникам.

Данный паттерн проектирования используется в курсовом проекте для генерации снарядов и торпед с различными характеристиками. Схема паттерна для игрового приложения представлен на рисунке 2.2.

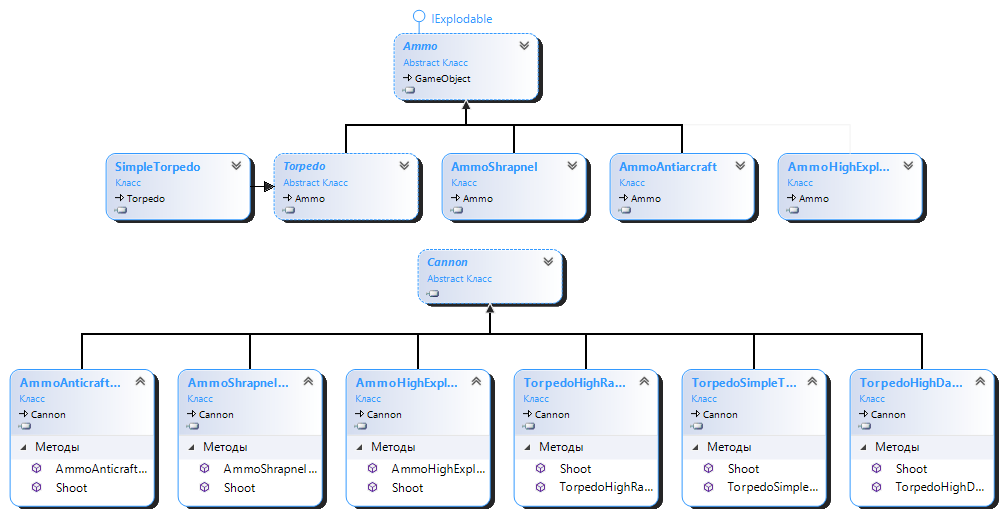


Рисунок 2.2 – Схема реализации паттерна «фабричный метод».

Для определения нового функционала в классах нередко используется наследование. Декоратор же представляет наследованию более гибкую альтернативу, поскольку позволяют динамически в процессе выполнения определять новые возможности у объектов [7].

Так как торпеды корабля могут иметь различные характеристики: дальность действия, скорости перемещения и силе причиняемых повреждений. Необходимо реализовать паттерн «Декоратор».

Декоратор (*Decorator*) – структурный шаблон проектирования, который позволяет динамически подключать к объекту дополнительную функциональность. Данный паттерн необходимо использовать, когда объекту нужно добавить новый функционал. Важным преимуществом данного паттерна от обычного наследования является возможность снять с объекта добавленный функционал, прямо во время выполнения программы. Т.е. декоратор позволяет снимать декорации, но снятие декораций должно происходить в порядке, обратном наложению. Схема паттерна представлена на рисунке 2.3 [5, c. 201].

Участники паттерна декоратор:

– класс *Component*. Это абстрактный класс, определяющий интерфейс для наследуемых объектов;

– класс *ConcreateComponent*. Это конкретная реализация объекта, в который и будет добавляться новый функционал при помощи декоратора;

– класс *Decorator*. Класс декоратора, который реализуется в виде абстрактного класса и имеет тот же базовый класс, что и объекты, что подлежат декорации. Именно поэтому базовый класс должен определять лишь базовый интерфейс. В этом классе храниться ссылка на декорируемый элемент в виде объекта базового класса *Component* и реализует связь с базовым классом как через наследование, так и через отношение агрегации.

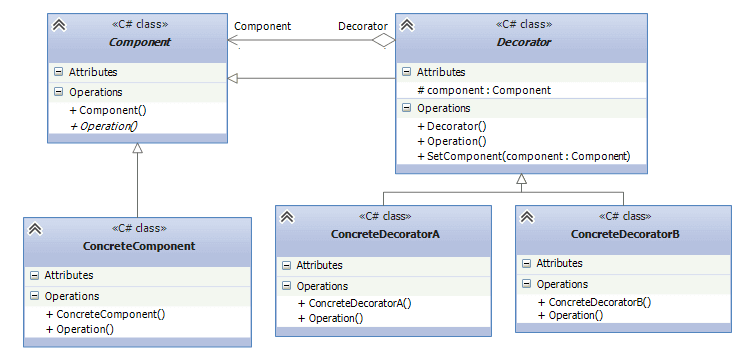


Рисунок 2.3 – Схема паттерна «Декоратор»

Применение различных паттернов проектирования может упростить разработку продукта, сэкономить время, а также внести ясности в код и сделать его репрезентативным. Данный паттерн, реализуемый в курсовом проекте, позволит динамически, по мере выполнения программы, изменять характеристики объектов. В приложении А представлена Схема использования паттерна «Декоратор».

Чтобы приложение было удобным для восприятия и понимания, необходимо разделить крупные задачи проекта на более мелкие. Эти мелкие подзадачи должны решать какую-то проблему или ряд проблем, они должны реализовывать какой-то функционал и продолжать до тех пор, пока не будет достигнута простота восприятия.

Основными элементами разделения выступили объекты игрового поля и их функциональные возможности. Вначале создаются статические объекты, которые решают свой спектр задач. Далее идут динамические объекты, а после них объекты, управляемые игроками. После этого реализуются методы взаимодействия между всеми объектами игрового поля.

## Принципы взаимодействия классов приложения

В процессе разработки игрового приложения была сформирована иерархическая структура классов, которые в полной мере описывают возможности своих сущностей. В игре могут присутствовать движущиеся объекты и статические. Движущиеся объекты имеют направление и скорость своего движения. Так же игровые объекты могут управляемые клавиатурой и неуправляемые.

Класс *GameObject* – основной класс, содержащий в себе необходимые поля и методы, присущие любому игровому объекту. Ему присущие такие свойства, как позиция, в которой он находится, его размер, метод CheckCollision, который отвечает определение столкновений с другими объектами на игровом поле, абстрактный метод метод отрисовки *Draw*, который реализовывается в конкретных игровых объектах.

Класс *Ship* наследуется от *GameObject*, содержащий в себе необходимые поля и методы, присущие кораблю. Ему присущие такие свойства, как позиция, в которой он находится, его размер, количество здоровья, количество боезапаса, скорость передвижения. У данного класса есть абстрактный метод *Move*, который отвечает за движение корабля, protected метод *FindAllowedDirections*, который определяет доступные направления для корабя и protected метод *InLowWater*, который определяет находится ли корабль на отмели.

В игровом приложении реализуется шаблон проектирования «Декоратор», который обеспечивает добавление игрокам нового функционала, такого как замедление корабля, когда тот попадает на мелководье, и который обеспечивает изменения характеристик торпед. Для этого был создан абстрактный класс *ShipDecorator*, который наследуется от класса Ship. От класса ShipDecorator наследуется класс *SlowShip*, который замедляет корабль. Также был создан абстрактный класс *TorpedoDecorator*, который наследуется от Torpedo. От класса *TorpedoDecorator* наследуются классы *HighRange*- *TorpedoDecorator* и *HighDamageTorpedoDecorator*, которые изменяют характеристики торпеды, такие как урон и дальность применения торпеды.

Шаблон проектирования «декоратор» применяется, когда необходимо динамически добавлять новые компоненты отдельному объекту во время выполнения программы, а также когда использование наследования в частном случае не представлялось возможным, поскольку это решение являлось статическим и распространялось целиком на весь класс [6].

Паттерн «декоратор» используется для изменения характеристик корабля. В «декораторе» реализовано изменение скорости корабля, когда тот попадает на отмель. Какими методами и свойствами реализуется паттерн «декоратор» для корабля продемонстрировано на рисунке 2.4 (Приложение А).

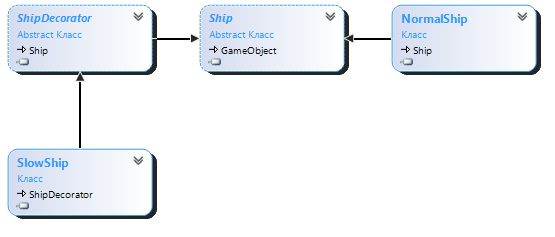


Рисунок 2.4 – Паттерн «декоратор» для корабля

«Декоратор» для изменения характеристик торпеды продемонстрировано на рисунке 2.5 (Приложение А).

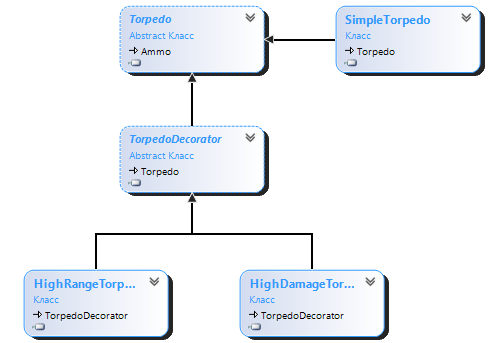


Рисунок 2.5 – Паттерн «декоратор» для торпед

Класс *Ammo* наследуется от *GameObject* и является классом абстрактного боеприпаса. Данный класс имеет свойства дальности выстрела, зона поражения боеприпаса, наносимый урон боеприпасом, ссылку на родительский корабль, а также методы *Move*, который отвечает за отрисовку боеприпаса

Класс *Bullet* наследуется от *GameObject* и содержит в себе информацию о патронах. Данный класс имеет свойство, которое определяет радиус патрона, а также метод *Update*, который отвечает за обновление патрона.

Класс *Ammo* наследуется от *GameObject* и представляет класс, который создает боеприпасы. Он содержат в себе метод *Move,* который отвечает за функцию перемещения боеприпаса, метод *Explode*, отвечающий за взрыв боезапаса, метод *Draw*, который отвечает за отрисовку боеприпаса.

Присутствует так же реализация паттерна «Фабричный метод», который применяется для генерации снарядов и торпед. В начале был создан абстрактный класс *Ammo*, который является родительским для конкретных cнарядов: снаряд *AmmoAntiarcraft*, снаряд *HighExplosive,* снаряд *AmmoShrapnel*, а также отдельные фабрики для создания определённого типа снаряда: *AmmoAnticraftCannon*, *AmmoHighExplosiveCannon*, *AmmoShrapnelCannon*.

Абстрактный класс Torpedo наследуется от класса Ammo, который является родительским для конкретных торпед: торпеда *SimpleTorpedo*, а также отдельные фабрики для создания определённого типа торпеды: *TorpedoHighDamageTube, TorpedoHighRangeTube, TorpedoSimpleTube.* Сами снаряды генерируются во время выстрела корабля.

Паттерн «фабричный метод» используется для реализации генерации «снарядов и торпед». На рисунке 2.6 представлен реализованный шаблон проектирования «фабричный метод»:

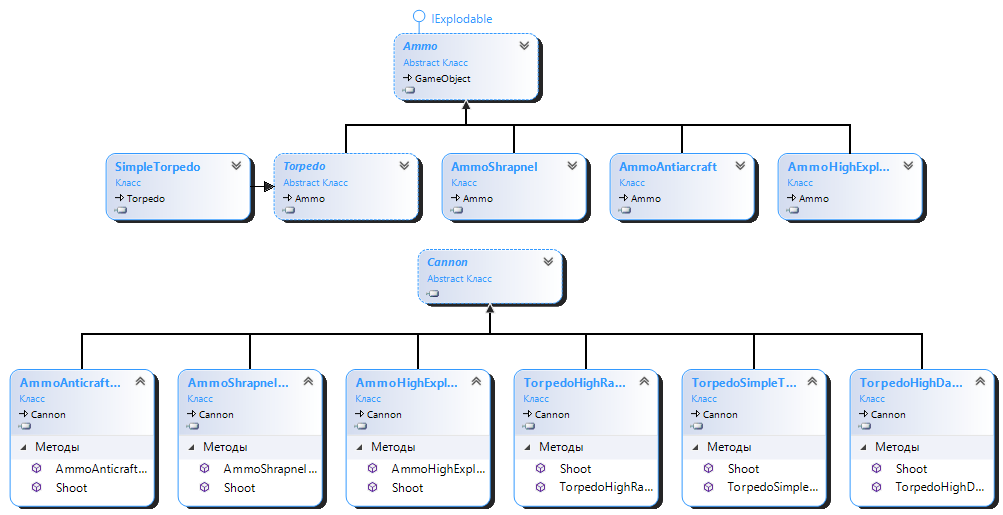


Рисунок 2.6 – Реализация шаблона проектирования «фабричный метод»

Класс *GameField* является классом игрового поля и хранит в себе информацию об игровых объектах и статусе игры.

Класс *Level* наследуется от *List<GameObject>*,является класс уровня. Представляет собой коллекцию препятствий на карте.

Перечисление *GameStatus* хранит в себе статусы игры: *InGame* – в игре, *PlayerFirstWins* – победа первого игрока, *PlayerSecondWins* – победа второго игрока, *Standoff* – ничья.

Класс *TextureRepository* является классом, который имеет метод *Add*, для добавления текстуры в словарь, и свойство *Get*, получения текстуры объекту.

Класс *GLTexture* имеет статический метод *LoadTexture*, который предназначен для загрузки текстуры.

Класс *TextureCreator* предназначен для создания текстур и спрайтов.

Класс *Explosian* наследуется от *GameObject* и отвечает за взрыв объекта.

Класс *Fire* наследуется от *GameObject* и отвечает за отрисовку огня на корабле, когда у того остается 50 и меньше единиц здоровья.

Класс *Hidrance* наследуется от *GameObjetct*, который представляет класс абстрактного препятствия. В свою очередь от класса *Hidrance* наследуется класс *Rock*, который является непроходимым препятствием для корабля и боеприпасов, и класс *Shallow*, которая является классом отмели и замедляет корабль, когда тот попадет в область действия отмели.

Таким образом, был реализован игровой процесс для приложения «Морской бой».

1. **ВЕРИФИКАЦИЯ И АПРОБАЦИЯ ИГРОВОГО ПРИЛОЖЕНИЯ «МОРСКОЙ БОЙ»**
   1. **Принцип работы игрового приложения**

Т.к. в данном курсовом проекте создается приложение *WPF* под определенный *.Net Framework*, то интеграция *OpenGL* с ним напрямую невозможна. Поэтому для разработки *.Net*-приложений необходимо использовать различные *.Net*-совместимые «обертки», такие как *OpenTK* или *SharpGL*. Это библиотеки, которые предоставляют классы для работы с *OpenGL*, причем названия их методов практически идентичны аналогичным функциям *OpenGL*. Для работы был выбран *OpenTK*, так как он является более простым в подключении к решению проекта.

Для того, чтобы отобразить картинку в окне *WPF,* необходимо скачать библиотеки *OpenTK* и подключить её к проекту. Далее следует разместить на форме элемент *OpenGLControl*.

В методе *MainWindow*() происходит инициализация основных компонентов игры, таких как игровое поле, его основные объекты, игроки, потоки параллельных вычислений. Потоки необходимы для того, чтобы обеспечить параллельное вычисление координатных компонент, не нагружая при этом поток отрисовки этими вычислениями. В данном случае это потоки просчета столкновений между объектами: игрока с игроком, столкновение игрока с непроходимыми объектами, столкновение боеприпасов с препятствиями и игроками. На рисунке 3.1 показан процесс создания игрового поля.

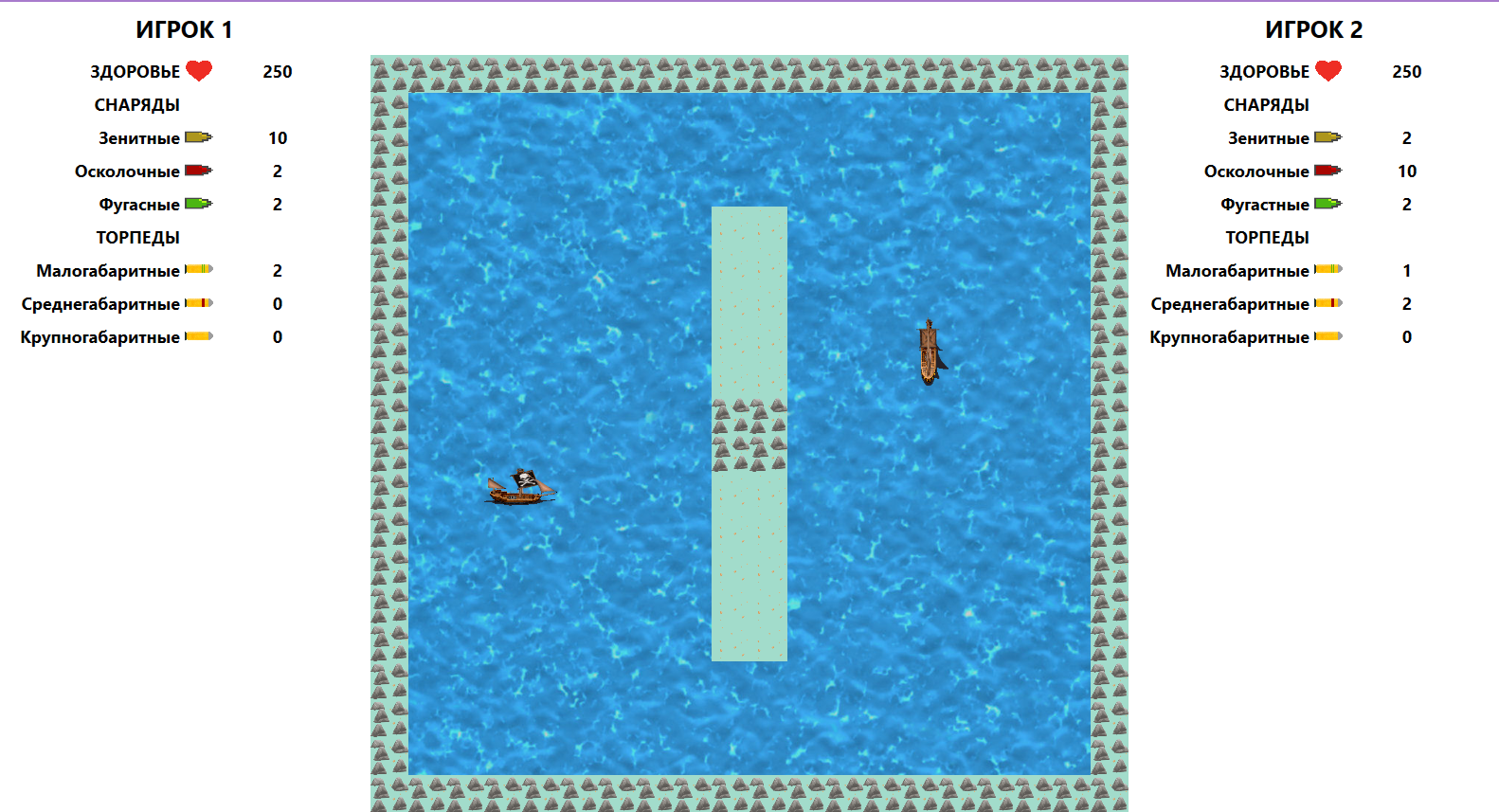


Рисунок 3.1 – Игрового поля

Как отмечалось ранее, для генерации различных снарядов и торпед был использован паттерн «фабричный метод». В начале игры игроки на одинаковое количество очков должны купить торпеды и снаряды, которые имеют различные характеристики: урон от попадания, дальность выстрела, скорость и зона поражения снаряда.

## 3.2 Тестирование приложения

Одним из важнейших этапов разработки игровых приложений является тестирование разрабатываемого программного обеспечения. Необходимо проверить правильность взаимодействия всех элементов системы, их поведение и работоспособность.

Модульные тесты должны соответствовать функциональности программного продукта, основное требование – скорость работы. Если после запуска набора тестов разработчик может сделать перерыв, то подобные запуски будут происходить всё реже и реже. В результате может получиться так, что модульные тесты вообще не будут запускаться и, как следствие, потеряется смысл их написания. Программист должен иметь возможность запустить весь набор тестов в любой момент времени. И этот набор должен выполниться настолько быстро, насколько это возможно. Исходный код всех вышеописанных классов находится в приложении А.

Описание модульных тестов игрового приложения (приложение В):

– *TestHighDamageTorpedoDecoratorDamage* – тестовый метод, проверяющий правильность работы декорации торпеды;

– *TestHighDamageTorpedoDecoratorRange* – тестовый метод, проверяющий правильность работы декорации торпеды;

– *TestHighDamageTorpedoDecoratorDamageArea* – тестовый метод, проверяющий правильность работы декорации торпеды;

*– TestHighRangeTorpedoDecoratorRange* – тестовый метод, проверяющий правильность работы декорации торпеды;

*– TestSlowShipDecorator* – тестовый метод, проверяющий правильность работы декорации корабля;

– *TestAmmoAnticraftCannon –* тестовый метод, проверяющий правильность работы фабричного метода на то что она возвращает;

*– TestAmmoHighExplosiveCannon –* тестовый метод, проверяющий правильность работы фабричного метода на то что она возвращает;

*– TestAmmoShrapnelCannon –* тестовый метод, проверяющий правильность работы фабричного метода на то что она возвращает;

*– TestTorpedoHighDamageTube –* тестовый метод, проверяющий правильность работы фабричного метода на то что она возвращает;

*– TestTorpedoHighRangeTube –* тестовый метод, проверяющий правильность работы фабричного метода на то что она возвращает;

*– TestTorpedoSimpleTube –* тестовый метод, проверяющий правильность работы фабричного метода на то что она возвращает.

На рисунке 3.1 представлены результаты прохождения тестов.

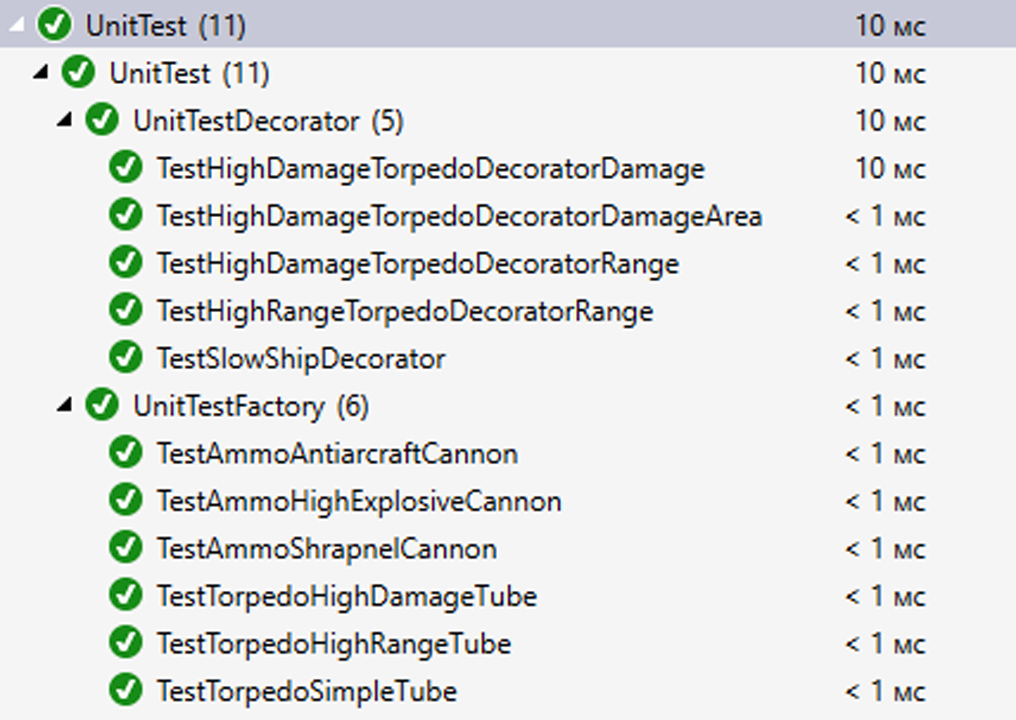


Рисунок 3.2 – Результаты прохождения тестов

Вышеперечисленные тесты позволяют корректность функционирования компонентов программы.

## 3.3 Пользовательский интерфейс

После запуска разработанного игрового приложения появляется окно ла-унчера, с помощью которого можно запустить игру. Лаунчер представляет собой простую форму с кнопкой, при нажатии на которую происходит запуск игры. На рисунке 3.3 представлен скриншот лаунчера.

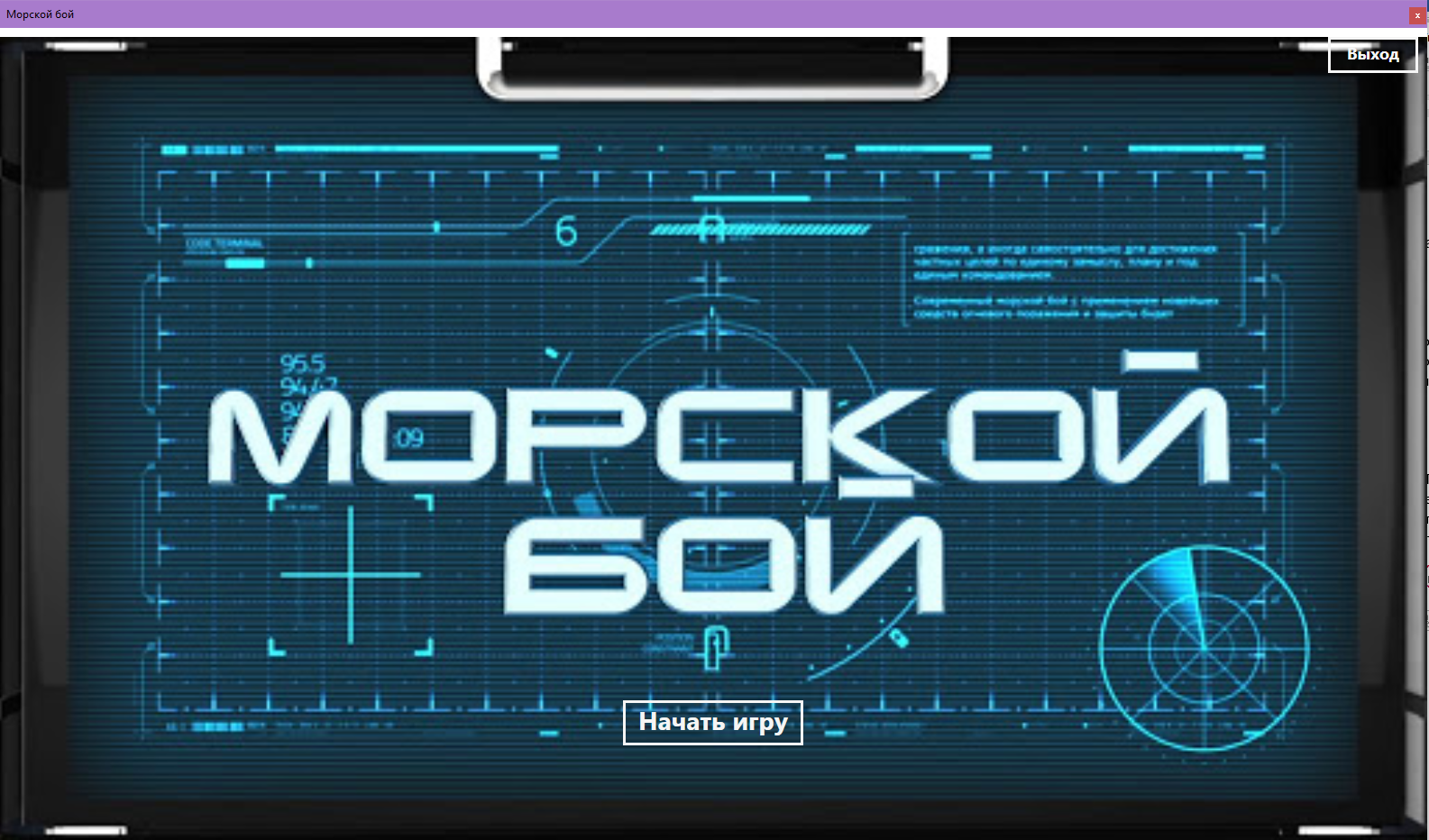


Рисунок 3.3 – Окно лаунчера

После нажатия на кнопку происходит инициализация и запуск магазина для покупки боеприпасов. На рисунке 3.4 представлен скриншот магазина.

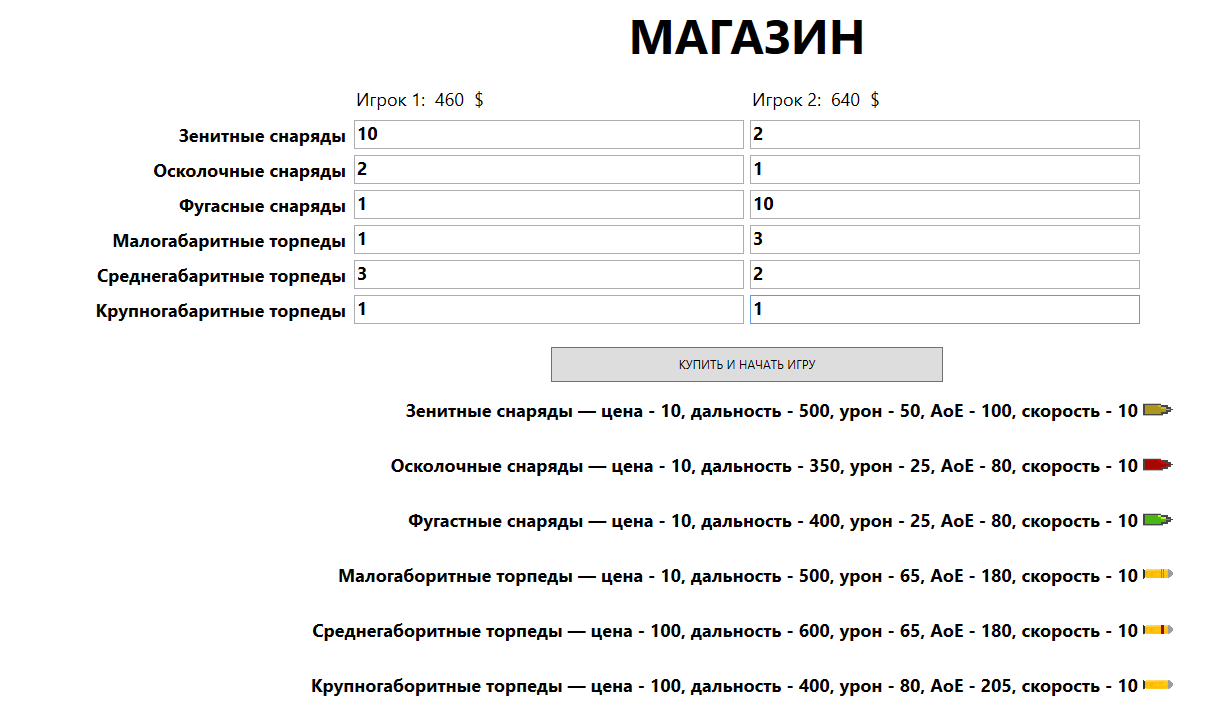


Рисунок 3.4 – Окно магазина

Игроки на сумму 1000 очков в начале игры должны купить определённое количество боеприпасов, при помощи которых они будут уничтожать друг друга. У игроков есть на выбор три типа снарядов и три типа торпед с различными характеристиками. На рисунке 3.5 показан внешний вид снарядов.



Рисунок 3.5 – Внешний вид снарядов

На рисунке 3.6 показан внешний вид торпед.

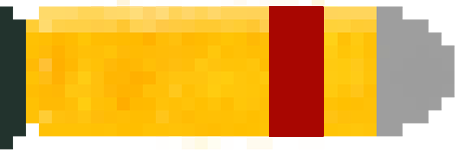


Рисунок 3.6 – Внешний вид снарядов

После покупки боеприпасов, происходит инициализация игрового поля. При старте оба игрока получают под свой контроль один из двух кораблей. Корабль первого игрока находится в левом нижем углу окна, а корабль второго расположен вверху справа игрового поля. С помощью клавиатуры игроки могут перемещаться по экрану вверх и вниз, влево и вправо и стрелять снарядами и торпедами в друг друга. На рисунке 3.7 представлен скриншот начала игры. Клавиши на клавиатуре, отвечающие за управление кораблями представлены в таблице 3.1.

Таблица 3.1 – Управление кораблями

|  |  |  |
| --- | --- | --- |
| Номер игрока | *Ship 1* | *Ship 2* |
| Передвижение вперед/назад | *W*/*S* | *I/K* |
| Передвижение влево/вправо | *A/D* | *J/L* |
| Стрельба снарядами | *Q/E/R* | *U/O/P* |
| Стрельба торпедами | *Z/X/C* | *B/N/M* |

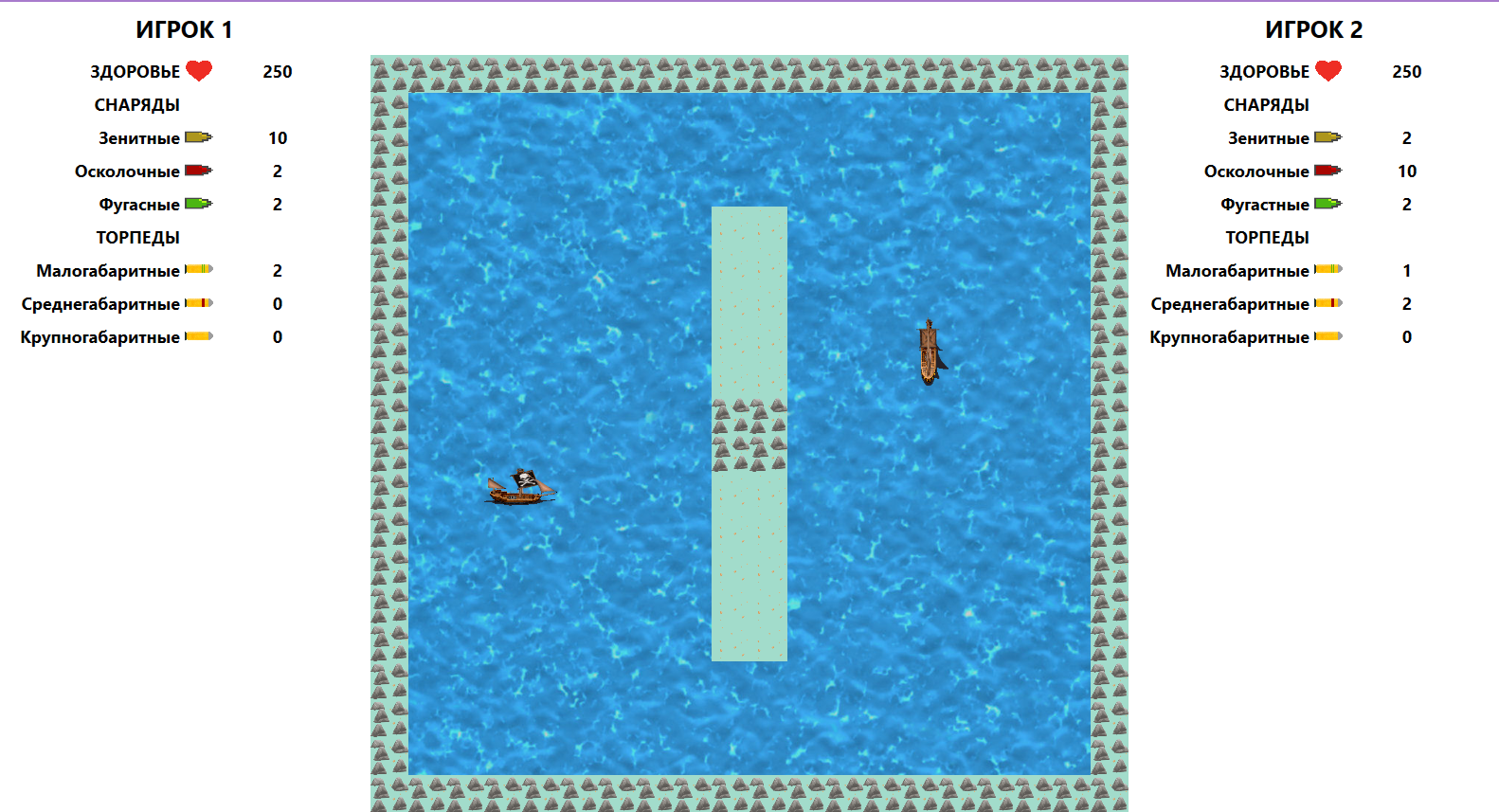


Рисунок 3.7 – Скриншот начала игры

Цель каждого игрока – уничтожить корабль противника. Для этого игрок должен попасть в противника снарядом или торпедой. У игроков ограниченный боезапас, поэтому его необходимо использовать экономно. Если у одного из кораблей кончаются боеприпасы, то игра окончится поражением для корабля без боеприпасов. На рисунке 3.7 представлен вид кораблей.



Рисунок 3.7 – Внешний вид кораблей

Характеристики каждого из игроков указываются в верхних углах экрана: слева вверху для первого игрока, справа вверху для второго игрока. В характеристиках также содержится информация о количестве здоровья игрока, его запас боеприпасов, что показано на рисунке 3.8.



Рисунок 3.8 – Характеристики игрока

Сообщение о завершении игры представлено на рисунке 3.9.



Рисунок 3.9 – Сообщение о конце игры

По окончанию игры, игроки могут начать игру заново, нажав на кнопку «Выйти в меню», переместив игроков в окно лаунчера.

# **ЗАКЛЮЧЕНИЕ**

При выполнении курсового проекта первоначально был проведен аналитический обзор литературы в области разработки игровых приложений, а также изучена предметная область и средства её реализации.

В соответствии с заданием была разработана 2*D*-игра жанра «Морской бой» под операционную систему *Windows* с использованием языка программирования *C#.* Для этого были рассмотрены различные программные интерфейсы для графического отображения в приложении *WPF*, с использованием графики *OpenGL*. Для этого при разработке приложения была использована графическая библиотека *OpenTK.* Данная графическая библиотека отвечает за визуализацию игрового пространства и его объектов, а также за использование спрайтовой графики.

Во время разработки игрового приложения был анализ подходов к решению поставленной задачи, выделены основные сущности игрового мира, их взаимосвязи, составлена иерархия классов. При реализации игрового проекта были использованы шаблоны проектирования: «декоратор» – для изменения характеристик торпед и корабля и «фабричный метод» – для реализации генерации торпед и снарядов. Они позволили не только упростить процесс разработки проекта, но и благодаря чему полученная библиотека классов имеет возможность расширения, путем добавления нового функционала, что способствует развитию и улучшению полученного программного продукта.

Разработанное игровое приложение состоит из меню покупки снаряжения, игрового поля с препятствиями, а также двух кораблей, управляемых пользователями, которые должны уничтожить друг друга.

Игра имеет минимальный порог вхождения, интуитивно понятна и, как следствие, не требует особых навыков. Соревновательные элементы игры грарантируют полное погружение в игровой процесс, получение интересного опыта, оказание положительного влияния на такие спектры развития, как логическое мышление и внимательность, что в совокупности повышает значимость и ценность игры как программного продукта.

Курсовой проект выполнен самостоятельно, проверен в системе «Антиплагиат». Процент оригинальности составляет 82,17%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников»

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гончаров, Д. А. *DirectX* 7.0 для программистов / Д. А. Гончаров, Т. Т. Салихов. – СПб: Питер, 2001. – 528 с.
2. *OpenGL* Библиотека программиста. 4-е издание / Д. Шрайнер [и др.]. – СПб: Питер, 2006. – 311с.
3. Диаграмма прибыли [Электронный ресурс]. – 2020. – Режим доступа: *https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience*-2020-2023/ – Дата доступа: 07.03.2021.
4. Воронина, В. В. Программирование игр: алгоритмы и технологии: учебное пособие / В. В. Воронина. – Ульяновск: УлГТУ, 2017. – 305 с.
5. Тепляков С. Паттерны проектирования на платформе .NET. – СПб: Питер, 2015. – 320 с.
6. Приемы объектно-ориентированного программирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. – СПб: Питер, 2001. – 368 с.
7. Декоратор [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/patterns/4.1.php– Дата доступа: 11.05.2020.

**ПРИЛОЖЕНИЕ A**

(обязательное)

**Схема использования паттерна «Декоратор»**

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Листинг программы «Морской бой»**

**Код программы для класса GameObject.cs:**

using System.Drawing;

namespace GameEngine.Game

{

/// <summary>

/// Базовый класс всех объектов на игровом поле

/// </summary>

public abstract class GameObject

{

/// <summary>

/// Коллекция текстур

/// </summary>

public abstract int[] Textures{ get; set; }

/// <summary>

/// Текущая текстура

/// </summary>

protected int current\_texture = 0;

/// <summary>

/// Ширина

/// </summary>

public abstract int Width { get; }

/// <summary>

/// Высота

/// </summary>

public abstract int Height { get; }

/// <summary>

/// Координаты на игровом поле

/// </summary>

public abstract Point Location { get; set; }

/// <summary>

/// Границы объекта

/// </summary>

public Rectangle Bounds

{

get

{

return new Rectangle(Location.X - Width / 2, Location.Y - Height / 2, Width, Height);

}

}

/// <summary>

/// Функция определения столкновений с другими объектами на карте

/// </summary>

/// <param name="obj"></param>

/// <returns></returns>

public bool CheckCollision(GameObject obj)

{

if (obj == this)

return false;

return Bounds.IntersectsWith(obj.Bounds);

}

/// <summary>

/// Функция рисования

/// </summary>

public abstract void Draw();

}

}

**Код программы для класса GameField:**

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using OpenTK;

using OpenTK.Graphics.OpenGL;

using GameEngine.Helpers;

using GameEngine.Objects;

using System;

namespace GameEngine.Game

{

/// <summary>

/// Класс игрового поля

/// </summary>

public class GameField

{

/// <summary>

/// Текущий уровень

/// </summary>

private Level \_currentLevel;

/// <summary>

/// Текстура заднего фона

/// </summary>

private int background\_texture;

/// <summary>

/// Размеры вьюпорта

/// </summary>

public Size ViewportSize { get; }

/// <summary>

/// Статус игры

/// </summary>

public GameStatus GameStatus

{

get

{

if ((\_ship1.Health > 0 && \_ship2.Health > 0 &&

\_ship1.Ammunition.Where(i => i > 0).Count() == 0 &&

\_ship2.Ammunition.Where(i => i > 0).Count() == 0 &&

Shots.Count() == 0) ||

(\_ship1.Health <= 0 && \_ship2.Health <= 0))

return GameStatus.Standoff;

if (\_ship1.Health <= 0 && \_ship2.Health > 0 || (\_ship1.Ammunition.Where(i => i > 0).Count() == 0 && Shots.Count() == 0))

return GameStatus.PlayerSecondWins;

if (\_ship1.Health > 0 && \_ship2.Health <= 0 || (\_ship2.Ammunition.Where(i => i > 0).Count() == 0 && Shots.Count() == 0))

return GameStatus.PlayerFirstWins;

return GameStatus.InGame;

}

}

/// <summary>

/// Корабли игроков. Поля

/// </summary>

private Ship \_ship1;

private Ship \_ship2;

public event EventHandler Ship1Changed;

public event EventHandler Ship2Changed;

/// <summary>

/// Корабли игроков. Свойства

/// </summary>

public Ship ShipFirst

{

get { return \_ship1; }

set

{

\_ship1 = value;

foreach (Cannon cannon in \_ship1.Cannons)

cannon.Parent = \_ship1;

Ship1Changed?.Invoke(\_ship1, null);

}

}

public Ship ShipSecond

{

get { return \_ship2; }

set

{

\_ship2 = value;

foreach (Cannon cannon in \_ship2.Cannons)

cannon.Parent = \_ship2;

Ship2Changed?.Invoke(\_ship2, null);

}

}

/// <summary>

/// Коллекция всех выстрелов

/// </summary>

public List<GameObject> Shots { get; }

/// <summary>

/// Коллекция непроходимых препятствий

/// </summary>

public IEnumerable<GameObject> AllObjects

{

get

{

return \_currentLevel.OfType<Rock>()

.Union(new List<GameObject>() { \_ship1, \_ship2 });

}

}

/// <summary>

/// Коллекция кораблей

/// </summary>

public IEnumerable<Ship> Ships

{

get

{

return new List<Ship>() { \_ship1, \_ship2 };

}

}

/// <summary>

/// Коллекция отмелей на карте

/// </summary>

public IEnumerable<Shallow> Shallows

{

get

{

return \_currentLevel.OfType<Shallow>();

}

}

/// <summary>

/// Конструктор

/// </summary>

public GameField(Level level, Ship ship1, Ship ship2)

{

ViewportSize = new Size(1000, 1000);

\_currentLevel = level;

Shots = new List<GameObject>();

\_ship1 = ship1;

\_ship2 = ship2;

background\_texture = TextureRepository.Get("background")[0];

}

/// <summary>

/// Функция рисования

/// </summary>

public void Draw()

{

drawBackground();

foreach (var hindrance in \_currentLevel)

hindrance.Draw();

var \_shots = Shots.ToArray();

foreach (var shot in \_shots)

shot.Draw();

\_ship1.Draw();

\_ship2.Draw();

}

/// <summary>

/// Функция отрисовки заднего фона

/// </summary>

private void drawBackground()

{

GL.BindTexture(TextureTarget.Texture2D, background\_texture);

GL.Begin(BeginMode.Polygon);

GL.TexCoord2(0, 0);

GL.Vertex2(0, 0);

GL.TexCoord2(1, 0);

GL.Vertex2(ViewportSize.Width, 0);

GL.TexCoord2(1, 1);

GL.Vertex2(ViewportSize.Width, ViewportSize.Height);

GL.TexCoord2(0, 1);

GL.Vertex2(0, ViewportSize.Height);

GL.End();

}

}

}

**Код программы для класса *GameStatus.cs*:**

namespace GameEngine.Game

{

/// <summary>

/// Перечисление статусов игры

/// </summary>

public enum GameStatus

{

InGame,

PlayerFirstWins,

PlayerSecondWins,

Standoff

}

}

**Код программы для класса *TextureRepository.cs*:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameEngine.Game

{

public static class TextureRepository

{

private static Dictionary<string, int[]> \_textures;

static TextureRepository()

{

\_textures = new Dictionary<string, int[]>();

}

public static void Add(string name, int[] textures)

{

\_textures.Add(name, textures);

}

public static int[] Get(string name)

{

return \_textures[name];

}

}

}

**Код программы для класса *Level.cs*:**

using System.Collections.Generic;

namespace GameEngine.Game

{

/// <summary>

/// Класс уровня. Представляет собой коллекцию препятствий на карте

/// </summary>

public class Level : List<GameObject>

{

}

}

**Код программы для класса *GLTexture.cs*:**

using System.Drawing;

using System.Drawing.Imaging;

using OpenTK.Graphics.OpenGL;

namespace GameEngine.Helpers

{

public class GLTexture

{

/// <summary>

/// Функция загрузки текстуры

/// </summary>

public static int LoadTexture(Bitmap bitmap)

{

int tex;

GL.Hint(HintTarget.PerspectiveCorrectionHint, HintMode.Nicest);

GL.GenTextures(1, out tex);

GL.BindTexture(TextureTarget.Texture2D, tex);

BitmapData data = bitmap.LockBits(new Rectangle(0, 0, bitmap.Width, bitmap.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);

bitmap.UnlockBits(data);

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

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

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

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

return tex;

}

}

}

**Код программы для класса *TextureCreator.cs*:**

using GameEngine.Helpers;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BattleShips

{

public static class TextureCreator

{

public static int[] CreateFire()

{

const int SPRITE\_WIDTH = 64;

const int SPRITE\_HEIGHT = 128;

int[] textures = new int[32];

Bitmap tex = Resources.Fire;

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

{

for (int j = 0; j < 8; j++)

{

Bitmap sprite = new Bitmap(SPRITE\_WIDTH, SPRITE\_HEIGHT);

Graphics g = Graphics.FromImage(sprite);

g.DrawImage(tex, new Rectangle(0, 0, SPRITE\_WIDTH, SPRITE\_HEIGHT),

new Rectangle(j \* SPRITE\_WIDTH, i \* SPRITE\_HEIGHT, SPRITE\_WIDTH, SPRITE\_HEIGHT), GraphicsUnit.Pixel);

textures[i \* 8 + j] = GLTexture.LoadTexture(sprite);

}

}

return textures;

}

public static int[] CreateExplosion()

{

const int SPRITE\_WIDTH = 192;

const int SPRITE\_HEIGHT = 192;

int[] textures = new int[7];

Bitmap tex = Resources.Explosion;

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 5; j++)

{

if (i > 0 && j > 1)

break;

Bitmap sprite = new Bitmap(SPRITE\_WIDTH, SPRITE\_HEIGHT);

Graphics g = Graphics.FromImage(sprite);

g.DrawImage(tex, new Rectangle(0, 0, SPRITE\_WIDTH, SPRITE\_HEIGHT),

new Rectangle(j \* SPRITE\_WIDTH, i \* SPRITE\_HEIGHT, SPRITE\_WIDTH, SPRITE\_HEIGHT), GraphicsUnit.Pixel);

textures[i \* 5 + j] = GLTexture.LoadTexture(sprite);

}

}

return textures;

}

public static int[] CreateShip()

{

int[] textures = new int[4];

Bitmap tex = Resources.shipFirst;

Bitmap sprite = new Bitmap(126, 224);

Graphics g = Graphics.FromImage(sprite);

g.DrawImage(tex, new Rectangle(0, 0, 126, 224),

new Rectangle(0, 0, 126, 224), GraphicsUnit.Pixel);

textures[0] = GLTexture.LoadTexture(sprite);

sprite = new Bitmap(310 - 126, 160 - 10);

g = Graphics.FromImage(sprite);

g.DrawImage(tex, new Rectangle(0, 0, 310 - 126, 160 - 10),

new Rectangle(126, 10, 310 - 126, 160 - 10), GraphicsUnit.Pixel);

textures[1] = GLTexture.LoadTexture(sprite);

sprite = new Bitmap(526 - 337, 160 - 10);

g = Graphics.FromImage(sprite);

g.DrawImage(tex, new Rectangle(0, 0, 526 - 337, 160 - 10),

new Rectangle(337, 10, 526 - 337, 160 - 10), GraphicsUnit.Pixel);

textures[2] = GLTexture.LoadTexture(sprite);

sprite = new Bitmap(126, 224);

g = Graphics.FromImage(sprite);

g.DrawImage(tex, new Rectangle(0, 0, 126, 224),

new Rectangle(550, 0, 126, 224), GraphicsUnit.Pixel);

textures[3] = GLTexture.LoadTexture(sprite);

return textures;

}

public static int[] CreateAmmo1()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.AmmoFirst);

return textures;

}

public static int[] CreateAmmo2()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.AmmoSecond);

return textures;

}

public static int[] CreateAmmo3()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.AmmoThird);

return textures;

}

public static int[] CreateTorpedo1()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.TorpedoFirst);

return textures;

}

public static int[] CreateTorpedo2()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.TorpedoSecond);

return textures;

}

public static int[] CreateTorpedo3()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.TorpedoThird);

return textures;

}

public static int[] CreateBackground()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.water);

return textures;

}

public static int[] CreateRock()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.Mountain);

return textures;

}

public static int[] CreateShallow()

{

int[] textures = new int[1];

textures[0] = GLTexture.LoadTexture(Resources.Мелководье);

return textures;

}

}

}

**Код программы для класса *Direction.cs*:**

using System;

namespace GameEngine.Input

{

/// <summary>

/// Перечисление направлений

/// </summary>

[Flags]

public enum Direction

{

Left = 1,

Right = 2,

Up = 4,

Down = 8

}

}

**Код программы для класса *IController.cs*:**

using System;

using GameEngine.Game;

using GameEngine.Objects;

namespace GameEngine.Input

{

/// <summary>

/// Интерфейс управляющего контроллера

/// </summary>

public interface IController

{

/// <summary>

/// Событие начала движения

/// </summary>

event EventHandler StartMoving;

/// <summary>

/// Событие остановки

/// </summary>

event EventHandler StopMoving;

/// <summary>

/// События стрельбы

/// </summary>

event EventHandler Shooting1;

event EventHandler Shooting2;

event EventHandler Shooting3;

event EventHandler Shooting4;

event EventHandler Shooting5;

event EventHandler Shooting6;

/// <summary>

/// Текущее направление движения

/// </summary>

Direction CurrentDirection { get; set; }

/// <summary>

/// Ссылка на управляемый корабль

/// </summary>

Ship Ship { get; set; }

/// <summary>

/// Ссылка на игровое поле

/// </summary>

GameField GameField { get; set; }

}

}

**Код программы для класса *KeyboardController.cs*:**

using System;

using GameEngine.Game;

using GameEngine.Objects;

namespace GameEngine.Input

{

/// <summary>

/// Класс контроллера клавиатуры

/// </summary>

public class KeyboardController : IController

{

/// <summary>

/// Текущее направление

/// </summary>

public Direction CurrentDirection { get; set; } = Direction.Up;

/// <summary>

/// Ссылка на корабль

/// </summary>

public Ship Ship { get; set; }

/// <summary>

/// Ссылка на поле

/// </summary>

public GameField GameField { get; set; }

/// <summary>

/// Событие начала движения

/// </summary>

public event EventHandler StartMoving;

/// <summary>

/// Событие остановки

/// </summary>

public event EventHandler StopMoving;

/// <summary>

/// Событие стрельбы

/// </summary>

public event EventHandler Shooting1;

public event EventHandler Shooting2;

public event EventHandler Shooting3;

public event EventHandler Shooting4;

public event EventHandler Shooting5;

public event EventHandler Shooting6;

/// <summary>

/// Конструктор

/// </summary>

public KeyboardController(GameField field)

{

GameField = field;

}

/// <summary>

/// Метод начала движения

/// </summary>

public void StartMove(Direction direction)

{

CurrentDirection = direction;

StartMoving?.Invoke(this, null);

}

/// <summary>

/// Остановка

/// </summary>

public void StopMove()

{

StopMoving?.Invoke(this, null);

}

/// <summary>

/// Стрельба

/// </summary>

public void Shoot1()

{

Shooting1?.Invoke(this, null);

}

public void Shoot2()

{

Shooting2?.Invoke(this, null);

}

public void Shoot3()

{

Shooting3?.Invoke(this, null);

}

public void Shoot4()

{

Shooting4?.Invoke(this, null);

}

public void Shoot5()

{

Shooting5?.Invoke(this, null);

}

public void Shoot6()

{

Shooting6?.Invoke(this, null);

}

}

}

**Код программы для класса *MainWindow.xaml.cs:***

using System;

using System.Windows.Input;

using System.Windows;

using System.Drawing;

using System.Windows.Threading;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

using GameLibrary.Cannons;

using GameLibrary.Ammunition;

using OpenTK.Graphics.OpenGL;

using Rectangle = System.Drawing.Rectangle;

using Point = System.Drawing.Point;

using System.ComponentModel;

using System.Runtime.CompilerServices;

using GameLibrary.Ships;

using System.Text.RegularExpressions;

namespace BattleShips

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window, INotifyPropertyChanged

{

/// <summary>

/// Метод оповещения при изменении свойства

/// </summary>

/// <param name="prop"></param>

public void OnPropertyChanged([CallerMemberName] string prop = "")

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));

}

/// <summary>

/// Событие изменения свойства

/// </summary>

public event PropertyChangedEventHandler PropertyChanged;

/// <summary>

/// Таймер

/// </summary>

private readonly DispatcherTimer \_animationTimer;

/// <summary>

/// Игровое поле

/// </summary>

private GameField gameField;

/// <summary>

/// Контроллеры управления

/// </summary>

private KeyboardController \_controller1;

private KeyboardController \_controller2;

/// <summary>

/// Показывает начата ли игра

/// </summary>

private bool gameStarted;

/// <summary>

/// Свойства для привязки GUI к модели

/// </summary>

#region GUI Properties

private void NumberValidationTextBox

(object sender, TextCompositionEventArgs e)

{

Regex regex = new Regex(@"^[^\d\s] + $");

e.Handled = regex.IsMatch(e.Text);

}

private string \_result;

public string Result

{

get { return \_result; }

set

{

\_result = value;

OnPropertyChanged();

}

}

private int \_healthFirstShip;

public int HealthFirst

{

get { return \_healthFirstShip; }

set

{

\_healthFirstShip = value;

OnPropertyChanged();

}

}

private int \_healthSecondShip;

public int HealthSecond

{

get { return \_healthSecondShip; }

set

{

\_healthSecondShip = value;

OnPropertyChanged();

}

}

private int \_ammoAntiaircraftFirstShip;

public int AmmoAntiaircraftFirstShip

{

get { return \_ammoAntiaircraftFirstShip; }

set

{

\_ammoAntiaircraftFirstShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal1();

}

}

private int \_ammoShrapnelFirstShip;

public int AmmoShrapnelFirstShip

{

get { return \_ammoShrapnelFirstShip; }

set

{

\_ammoShrapnelFirstShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal1();

}

}

private int \_ammoHighExplosiveFirstShip;

public int AmmoHighExplosiveFirstShip

{

get { return \_ammoHighExplosiveFirstShip; }

set

{

\_ammoHighExplosiveFirstShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal1();

}

}

private int \_torpedoSmallSizedFirstShip;

public int TorpedoSmallSizedFirstShip

{

get { return \_torpedoSmallSizedFirstShip; }

set

{

\_torpedoSmallSizedFirstShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal1();

}

}

private int \_torpedoMediumSizedFirstShip;

public int TorpedoMediumSizedFirstShip

{

get { return \_torpedoMediumSizedFirstShip; }

set

{

\_torpedoMediumSizedFirstShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal1();

}

}

private int \_torpedoOversizeFirstShip;

public int TorpedoOversizeFirstShip

{

get { return \_torpedoOversizeFirstShip; }

set

{

\_torpedoOversizeFirstShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal1();

}

}

private int \_ammoAntiaircraftSecondShip;

public int AmmoAntiaircraftSecondShip

{

get { return \_ammoAntiaircraftSecondShip; }

set

{

\_ammoAntiaircraftSecondShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal2();

}

}

private int \_ammoShrapnelSecondShip;

public int AmmoShrapnelSecondShip

{

get { return \_ammoShrapnelSecondShip; }

set

{

\_ammoShrapnelSecondShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal2();

}

}

private int \_ammoHighExplosiveSecondShip;

public int AmmoHighExplosiveSecondShip

{

get { return \_ammoHighExplosiveSecondShip; }

set

{

\_ammoHighExplosiveSecondShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal2();

}

}

private int \_torpedoSmallSizedSecondShip;

public int TorpedoSmallSizedSecondShip

{

get { return \_torpedoSmallSizedSecondShip; }

set

{

\_torpedoSmallSizedSecondShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal2();

}

}

private int \_torpedoMediumSizedSecondShip;

public int TorpedoMediumSizedSecondShip

{

get { return \_torpedoMediumSizedSecondShip; }

set

{

\_torpedoMediumSizedSecondShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal2();

}

}

private int \_torpedoOversizeSecondShip;

public int TorpedoOversizeSecondShip

{

get { return \_torpedoOversizeSecondShip; }

set

{

\_torpedoOversizeSecondShip = value;

OnPropertyChanged();

if (!gameStarted)

CalculateTotal2();

}

}

private int \_playerFirstTotal = 1000;

public int PlayerFirstTotal

{

get { return \_playerFirstTotal; }

set

{

\_playerFirstTotal = value;

OnPropertyChanged();

}

}

private void CalculateTotal1()

{

PlayerFirstTotal = 1000 - (

AmmoAntiaircraftFirstShip \* AmmoAntiarcraft.Cost +

AmmoShrapnelFirstShip \* AmmoShrapnel.Cost +

AmmoHighExplosiveFirstShip \* AmmoHighExplosive.Cost +

TorpedoSmallSizedFirstShip \* SimpleTorpedo.Cost +

TorpedoMediumSizedFirstShip \* HighDamageTorpedoDecorator.Cost +

TorpedoOversizeFirstShip \* HighRangeTorpedoDecorator.Cost);

if (PlayerFirstTotal < 0)

{

MessageBox.Show("Не хватает средств!");

btnStartGame.IsEnabled = false;

}

else

btnStartGame.IsEnabled = true;

}

private int \_playerSecondTotal = 1000;

public int PlayerSecondTotal

{

get { return \_playerSecondTotal; }

set

{

\_playerSecondTotal = value;

OnPropertyChanged();

}

}

private void CalculateTotal2()

{

PlayerSecondTotal = 1000 - (

AmmoAntiaircraftSecondShip \* AmmoAntiarcraft.Cost +

AmmoShrapnelSecondShip \* AmmoShrapnel.Cost +

AmmoHighExplosiveSecondShip \* AmmoHighExplosive.Cost +

TorpedoSmallSizedSecondShip \* SimpleTorpedo.Cost +

TorpedoMediumSizedSecondShip \* HighDamageTorpedoDecorator.Cost +

TorpedoOversizeSecondShip \* HighRangeTorpedoDecorator.Cost);

if (PlayerSecondTotal < 0)

{

MessageBox.Show("Не хватает средств!");

btnStartGame.IsEnabled = false;

}

else

btnStartGame.IsEnabled = true;

}

#endregion

/// <summary>

/// Конструктор

/// </summary>

public MainWindow()

{

InitializeComponent();

\_animationTimer = new DispatcherTimer(DispatcherPriority.ContextIdle, Dispatcher)

{

Interval = new TimeSpan(0, 0, 0, 0, 33)

};

TextureRepository.Add("fire", TextureCreator.CreateFire());

TextureRepository.Add("explosion", TextureCreator.CreateExplosion());

TextureRepository.Add("ship", TextureCreator.CreateShip());

TextureRepository.Add("ammo1", TextureCreator.CreateAmmo1());

TextureRepository.Add("ammo2", TextureCreator.CreateAmmo2());

TextureRepository.Add("ammo3", TextureCreator.CreateAmmo3());

TextureRepository.Add("torpedo1", TextureCreator.CreateTorpedo1());

TextureRepository.Add("torpedo2", TextureCreator.CreateTorpedo2());

TextureRepository.Add("torpedo3", TextureCreator.CreateTorpedo3());

TextureRepository.Add("background", TextureCreator.CreateBackground());

TextureRepository.Add("rock", TextureCreator.CreateRock());

TextureRepository.Add("shallow", TextureCreator.CreateShallow());

}

/// <summary>

/// Обработчики изменения кораблей

/// </summary>

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

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

private void ShipFirst\_Changed(object sender, EventArgs e)

{

Ship s1 = (Ship)sender;

HealthFirst = s1.Health;

AmmoAntiaircraftFirstShip = s1.Ammunition[0];

AmmoShrapnelFirstShip = s1.Ammunition[1];

AmmoHighExplosiveFirstShip = s1.Ammunition[2];

TorpedoSmallSizedFirstShip = s1.Ammunition[3];

TorpedoMediumSizedFirstShip = s1.Ammunition[4];

TorpedoOversizeFirstShip = s1.Ammunition[5];

}

private void ShipSecond\_Changed(object sender, EventArgs e)

{

Ship s2 = (Ship)sender;

HealthSecond = s2.Health;

AmmoAntiaircraftSecondShip = s2.Ammunition[0];

AmmoShrapnelSecondShip = s2.Ammunition[1];

AmmoHighExplosiveSecondShip = s2.Ammunition[2];

TorpedoSmallSizedSecondShip = s2.Ammunition[3];

TorpedoMediumSizedSecondShip = s2.Ammunition[4];

TorpedoOversizeSecondShip = s2.Ammunition[5];

}

/// <summary>

/// Обработчик нажатия клавиши на клавиатуре

/// </summary>

protected override void OnKeyDown(KeyEventArgs e)

{

base.OnKeyDown(e);

if (!gameStarted)

return;

switch (e.Key)

{

case Key.W:

\_controller1.StartMove(Direction.Up);

break;

case Key.S:

\_controller1.StartMove(Direction.Down);

break;

case Key.A:

\_controller1.StartMove(Direction.Left);

break;

case Key.D:

\_controller1.StartMove(Direction.Right);

break;

case Key.Q:

\_controller1.Shoot1();

break;

case Key.E:

\_controller1.Shoot2();

break;

case Key.R:

\_controller1.Shoot3();

break;

case Key.Z:

\_controller1.Shoot4();

break;

case Key.X:

\_controller1.Shoot5();

break;

case Key.C:

\_controller1.Shoot6();

break;

case Key.I:

\_controller2.StartMove(Direction.Up);

break;

case Key.K:

\_controller2.StartMove(Direction.Down);

break;

case Key.J:

\_controller2.StartMove(Direction.Left);

break;

case Key.L:

\_controller2.StartMove(Direction.Right);

break;

case Key.U:

\_controller2.Shoot1();

break;

case Key.O:

\_controller2.Shoot2();

break;

case Key.P:

\_controller2.Shoot3();

break;

case Key.B:

\_controller2.Shoot4();

break;

case Key.N:

\_controller2.Shoot5();

break;

case Key.M:

\_controller2.Shoot6();

break;

}

}

/// <summary>

/// Обработчик отпускания клавиши на клавиатуре

/// </summary>

protected override void OnKeyUp(KeyEventArgs e)

{

base.OnKeyUp(e);

if (!gameStarted)

return;

switch (e.Key)

{

case Key.W:

case Key.S:

case Key.A:

case Key.D:

\_controller1.StopMove();

break;

case Key.I:

case Key.K:

case Key.J:

case Key.L:

\_controller2.StopMove();

break;

}

}

/// <summary>

/// Обработчик тика таймера

/// </summary>

private void OnAnimationTick(object sender, EventArgs e)

{

if (gameField.GameStatus != GameStatus.InGame)

{

gameStarted = false;

\_animationTimer.Stop();

\_animationTimer.Tick -= OnAnimationTick;

switch (gameField.GameStatus)

{

case GameStatus.PlayerFirstWins:

Result = "Игрок 1 победил!!!";

break;

case GameStatus.PlayerSecondWins:

Result = "Игрок 2 победил!!!";

break;

case GameStatus.Standoff:

Result = "Ничья!!!";

break;

}

GameScreen.Visibility = Visibility.Hidden;

EndGameScreen.Visibility = Visibility.Visible;

}

renderCanvas.Invalidate();

}

/// <summary>

/// Обработчик инициализации хоста элементов WinForms

/// </summary>

private void WindowsFormsHost\_Initialized(object sender, EventArgs e)

{

renderCanvas.MakeCurrent();

}

/// <summary>

/// Обработчик события загрузки холста

/// </summary>

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

{

GL.ClearColor(Color.White);

GL.Viewport(0, 0, renderCanvas.Width, renderCanvas.Height);

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(0, 1000, 1000, 0, 0.0, 1.0);

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Blend);

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

}

/// <summary>

/// Обработчик события рисования холста

/// </summary>

private void renderCanvas\_Paint(object sender, System.Windows.Forms.PaintEventArgs e)

{

GL.Clear(ClearBufferMask.ColorBufferBit);

int x = (renderCanvas.Width - 800) / 2;

int y = (renderCanvas.Height - 800) / 2;

GL.Viewport(x, y, 800, 800);

if (gameField != null)

gameField.Draw();

GL.Flush();

renderCanvas.SwapBuffers();

}

/// <summary>

/// Обработчик нажатия на кнопку "Начать игру"

/// </summary>

private void btnStart\_Click(object sender, RoutedEventArgs e)

{

StartScreen.Visibility = Visibility.Hidden;

ShopScreen.Visibility = Visibility.Visible;

AmmoAntiaircraftFirstShip = 0;

AmmoShrapnelFirstShip = 0;

AmmoHighExplosiveFirstShip = 0;

TorpedoSmallSizedFirstShip = 0;

TorpedoMediumSizedFirstShip = 0;

TorpedoOversizeFirstShip = 0;

AmmoAntiaircraftSecondShip = 0;

AmmoShrapnelSecondShip = 0;

AmmoHighExplosiveSecondShip = 0;

TorpedoSmallSizedSecondShip = 0;

TorpedoMediumSizedSecondShip = 0;

TorpedoOversizeSecondShip = 0;

PlayerFirstTotal = 1000;

PlayerSecondTotal = 1000;

}

/// <summary>

/// Обработчик нажатия на кнопку "купить снаряды и начать игру"

/// </summary>

private void btnStartGame\_Click(object sender, RoutedEventArgs e)

{

Level hindrances = new Level()

{

new Rock(){Location = new Point(25, 25)},

new Rock(){Location = new Point(25,75)},

new Rock(){Location = new Point(25,125)},

new Rock(){Location = new Point(25,175)},

new Rock(){Location = new Point(25,225)},

new Rock(){Location = new Point(25,275)},

new Rock(){Location = new Point(25,325)},

new Rock(){Location = new Point(25,375)},

new Rock(){Location = new Point(25,425)},

new Rock(){Location = new Point(25,475)},

new Rock(){Location = new Point(25,525)},

new Rock(){Location = new Point(25,575)},

new Rock(){Location = new Point(25,625)},

new Rock(){Location = new Point(25,675)},

new Rock(){Location = new Point(25,725)},

new Rock(){Location = new Point(25,775)},

new Rock(){Location = new Point(25,825)},

new Rock(){Location = new Point(25,875)},

new Rock(){Location = new Point(25,925)},

new Rock(){Location = new Point(25,975)},

new Rock(){Location = new Point(975,75)},

new Rock(){Location = new Point(975,125)},

new Rock(){Location = new Point(975,175)},

new Rock(){Location = new Point(975,225)},

new Rock(){Location = new Point(975,275)},

new Rock(){Location = new Point(975,325)},

new Rock(){Location = new Point(975,375)},

new Rock(){Location = new Point(975,425)},

new Rock(){Location = new Point(975,475)},

new Rock(){Location = new Point(975,525)},

new Rock(){Location = new Point(975,575)},

new Rock(){Location = new Point(975,625)},

new Rock(){Location = new Point(975,675)},

new Rock(){Location = new Point(975,725)},

new Rock(){Location = new Point(975,775)},

new Rock(){Location = new Point(975,825)},

new Rock(){Location = new Point(975,875)},

new Rock(){Location = new Point(975,925)},

new Rock(){Location = new Point(975,975)},

new Rock(){Location = new Point(75,25)},

new Rock(){Location = new Point(125,25)},

new Rock(){Location = new Point(175,25)},

new Rock(){Location = new Point(225,25)},

new Rock(){Location = new Point(275,25)},

new Rock(){Location = new Point(325,25)},

new Rock(){Location = new Point(375,25)},

new Rock(){Location = new Point(425,25)},

new Rock(){Location = new Point(475,25)},

new Rock(){Location = new Point(525,25)},

new Rock(){Location = new Point(575,25)},

new Rock(){Location = new Point(625,25)},

new Rock(){Location = new Point(675,25)},

new Rock(){Location = new Point(725,25)},

new Rock(){Location = new Point(775,25)},

new Rock(){Location = new Point(825,25)},

new Rock(){Location = new Point(875,25)},

new Rock(){Location = new Point(925,25)},

new Rock(){Location = new Point(975,25)},

new Rock(){Location = new Point(75,975)},

new Rock(){Location = new Point(125,975)},

new Rock(){Location = new Point(175,975)},

new Rock(){Location = new Point(225,975)},

new Rock(){Location = new Point(275,975)},

new Rock(){Location = new Point(325,975)},

new Rock(){Location = new Point(375,975)},

new Rock(){Location = new Point(425,975)},

new Rock(){Location = new Point(475,975)},

new Rock(){Location = new Point(525,975)},

new Rock(){Location = new Point(575,975)},

new Rock(){Location = new Point(625,975)},

new Rock(){Location = new Point(675,975)},

new Rock(){Location = new Point(725,975)},

new Rock(){Location = new Point(775,975)},

new Rock(){Location = new Point(825,975)},

new Rock(){Location = new Point(875,975)},

new Rock(){Location = new Point(925,975)},

new Rock(){Location = new Point(475,475)},

new Rock(){Location = new Point(525,475)},

new Shallow(){Location = new Point(475,425)},

new Shallow(){Location = new Point(475,375)},

new Shallow(){Location = new Point(475,325)},

new Shallow(){Location = new Point(475,275)},

new Shallow(){Location = new Point(525,425)},

new Shallow(){Location = new Point(525,375)},

new Shallow(){Location = new Point(525,325)},

new Shallow(){Location = new Point(525,275)},

new Shallow(){Location = new Point(525,225)},

new Shallow(){Location = new Point(475,225)},

new Rock(){Location = new Point(475,525)},

new Rock(){Location = new Point(525,525)},

new Shallow(){Location = new Point(475,575)},

new Shallow(){Location = new Point(475,625)},

new Shallow(){Location = new Point(475,775)},

new Shallow(){Location = new Point(475,725)},

new Shallow(){Location = new Point(475,675)},

new Shallow(){Location = new Point(525,575)},

new Shallow(){Location = new Point(525,625)},

new Shallow(){Location = new Point(525,675)},

new Shallow(){Location = new Point(525,725)},

new Shallow(){Location = new Point(525,775)},

};

\_controller1 = new KeyboardController(gameField);

\_controller2 = new KeyboardController(gameField);

Ship s1 = new NormalShip(\_controller1, new int[]

{

AmmoAntiaircraftFirstShip,

AmmoShrapnelFirstShip,

AmmoHighExplosiveFirstShip,

TorpedoSmallSizedFirstShip,

TorpedoMediumSizedFirstShip,

TorpedoOversizeFirstShip

});

s1.Changed += ShipFirst\_Changed;

s1.Location = new Point(120, 900);

s1.Cannons[0] = new AmmoAnticraftCannon(s1);

s1.Cannons[1] = new AmmoShrapnelCannon(s1);

s1.Cannons[2] = new AmmoHighExplosiveCannon(s1);

s1.Cannons[3] = new TorpedoSimpleTube(s1);

s1.Cannons[4] = new TorpedoHighRangeTube(s1);

s1.Cannons[5] = new TorpedoHighDamageTube(s1);

Ship s2 = new NormalShip(\_controller2, new int[]

{

AmmoAntiaircraftSecondShip,

AmmoShrapnelSecondShip,

AmmoHighExplosiveSecondShip,

TorpedoSmallSizedSecondShip,

TorpedoMediumSizedSecondShip,

TorpedoOversizeSecondShip

});

s2.Changed += ShipSecond\_Changed;

s2.Location = new Point(900, 120);

s2.Cannons[0] = new AmmoAnticraftCannon(s2);

s2.Cannons[1] = new AmmoShrapnelCannon(s2);

s2.Cannons[2] = new AmmoHighExplosiveCannon(s2);

s2.Cannons[3] = new TorpedoSimpleTube(s2);

s2.Cannons[4] = new TorpedoHighRangeTube(s2);

s2.Cannons[5] = new TorpedoHighDamageTube(s2);

gameField = new GameField(hindrances, s1, s2);

gameField.Ship1Changed += GameField\_Ship1Changed;

gameField.Ship2Changed += GameField\_ShipSecondChanged;

\_controller1.GameField = gameField;

\_controller2.GameField = gameField;

HealthFirst = s1.Health;

HealthSecond = s2.Health;

ShopScreen.Visibility = Visibility.Hidden;

GameScreen.Visibility = Visibility.Visible;

\_animationTimer.Tick += OnAnimationTick;

\_animationTimer.Start();

gameStarted = true;

}

private void GameField\_ShipSecondChanged(object sender, EventArgs e)

{

gameField.ShipSecond.Changed += ShipSecond\_Changed;

}

private void GameField\_Ship1Changed(object sender, EventArgs e)

{

gameField.ShipFirst.Changed += ShipFirst\_Changed;

}

/// <summary>

/// Обработчик нажатия на кнопку "выход"

/// </summary>

private void btnExit\_Click(object sender, RoutedEventArgs e)

{

Close();

}

/// <summary>

/// Обработчик нажатия на кнопку "Вернуться в меню"

/// </summary>

private void btnMenu\_Click(object sender, RoutedEventArgs e)

{

EndGameScreen.Visibility = Visibility.Hidden;

StartScreen.Visibility = Visibility.Visible;

}

}

}

**Код программы для класса *Ammo.cs:***

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GameEngine.Game;

using GameEngine.Input;

using OpenTK.Graphics.OpenGL;

namespace GameEngine.Objects

{

/// <summary>

/// Класс абстрактного боеприпаса

/// </summary>

public abstract class Ammo : GameObject, IExplodable

{

public override int[] Textures { get; set; }

/// <summary>

/// Дальность выстрела

/// </summary>

public abstract int Range { get; }

/// <summary>

/// Наносимый урон

/// </summary>

public abstract int Damage { get; }

/// <summary>

/// Зона поражения

/// </summary>

public abstract int DamageArea { get; }

/// <summary>

/// Направление выстрела

/// </summary>

public abstract Direction ShotDirection { get; set; }

/// <summary>

/// Ссылка на игровое поле

/// </summary>

public abstract GameField GameField { get; set; }

/// <summary>

/// Ссылка на корабль

/// </summary>

public abstract Ship Parent { get; set; }

/// <summary>

/// Скорость

/// </summary>

public abstract int Speed { get; }

/// <summary>

/// Текущая дальность

/// </summary>

private int currentRange;

/// <summary>

/// Ширина

/// </summary>

public override int Width

{

get

{

return ShotDirection == Direction.Up ||

ShotDirection == Direction.Down ? 10 : 20;

}

}

/// <summary>

/// Высота

/// </summary>

public override int Height

{

get

{

return ShotDirection == Direction.Up ||

ShotDirection == Direction.Down ? 20 : 10;

}

}

/// <summary>

/// Функция перемещения

/// </summary>

private void Move()

{

switch (ShotDirection)

{

case Direction.Up:

Location = new Point(Location.X, Location.Y - Speed);

break;

case Direction.Down:

Location = new Point(Location.X, Location.Y + Speed);

break;

case Direction.Left:

Location = new Point(Location.X - Speed, Location.Y);

break;

case Direction.Right:

Location = new Point(Location.X + Speed, Location.Y);

break;

}

GameObject go = (GameObject)this;

if (Location.X < 0 || Location.Y > GameField.ViewportSize.Width ||

Location.Y < 0 || Location.Y > GameField.ViewportSize.Height)

{

GameField.Shots.Remove(this);

return;

}

var objects = GameField.AllObjects.ToArray();

foreach (var obj in objects)

{

if (obj != Parent && CheckCollision(obj))

Explode();

}

currentRange += Speed;

if (currentRange > Range)

{

Explode();

return;

}

}

/// <summary>

/// Взрыв боезапаса

/// </summary>

public void Explode()

{

Explosion explosion = new Explosion(this);

GameField.Shots.Remove(this);

GameField.Shots.Add(explosion);

}

/// <summary>

/// Функция отрисовки

/// </summary>

public override void Draw()

{

Move();

GL.BindTexture(TextureTarget.Texture2D, Textures[current\_texture]);

GL.Begin(BeginMode.Polygon);

Point textureX = getStartCoord(ShotDirection);

GL.TexCoord2(textureX.X, textureX.Y);

GL.Vertex3(Bounds.X, Bounds.Y, 0);

textureX = getNextTextureCoord(textureX);

GL.TexCoord2(textureX.X, textureX.Y);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y, 0);

textureX = getNextTextureCoord(textureX);

GL.TexCoord2(textureX.X, textureX.Y);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y + Bounds.Height, 0);

textureX = getNextTextureCoord(textureX);

GL.TexCoord2(textureX.X, textureX.Y);

GL.Vertex3(Bounds.X, Bounds.Y + Bounds.Height, 0);

GL.End();

}

/// <summary>

/// Функция получения начальных координат текстуры

/// </summary>

private static Point getStartCoord(Direction direction)

{

switch (direction)

{

case Direction.Up:

return new Point(1, 0);

case Direction.Right:

return new Point(0, 0);

case Direction.Down:

return new Point(0, 1);

default:

return new Point(1, 1);

}

}

/// <summary>

/// Функция получения последующих текстурных координат

/// </summary>

private static Point getNextTextureCoord(Point coord)

{

if (coord.X == 0 && coord.Y == 0)

{

coord.X = 1;

return coord;

}

if (coord.X == 1 && coord.Y == 0)

{

coord.Y = 1;

return coord;

}

if (coord.X == 1 && coord.Y == 1)

{

coord.X = 0;

return coord;

}

coord.X = 0;

coord.Y = 0;

return coord;

}

}

}

**Код программы для класса *Cannon.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameEngine.Objects

{

/// <summary>

/// Класс абстрактного орудия

/// </summary>

public abstract class Cannon

{

/// <summary>

/// Ссылка на родительский корабль

/// </summary>

public Ship Parent { get; set; }

/// <summary>

/// Выстрел

/// </summary>

public abstract Ammo Shoot();

}

}

**Код программы для класса *Explosion.cs:***

using System.Drawing;

using OpenTK.Graphics.OpenGL;

using GameEngine.Helpers;

using GameEngine.Game;

namespace GameEngine.Objects

{

/// <summary>

/// Класс взрыва

/// </summary>

public class Explosion : GameObject

{

public override int[] Textures { get; set; }

/// <summary>

/// Размеры спрайта

/// </summary>

private const int SPRITE\_WIDTH = 192;

private const int SPRITE\_HEIGHT = 192;

/// <summary>

/// Размер

/// </summary>

private int \_size;

/// <summary>

/// Ссылка на игровое поле

/// </summary>

private GameField \_gameField;

/// <summary>

/// Размеры

/// </summary>

public override int Width { get { return \_size; } }

public override int Height { get { return \_size; } }

public override Point Location { get; set; }

/// <summary>

/// Конструктор

/// </summary>

public Explosion(IExplodable ammo)

{

\_size = ammo.DamageArea;

Location = ((GameObject)ammo).Location;

\_gameField = ammo.GameField;

foreach (var ship in \_gameField.Ships)

if (CheckCollision(ship))

{

ship.Health -= ammo.Damage;

ship.OnChanged();

}

Textures = TextureRepository.Get("explosion");

}

/// <summary>

/// Функция рисования

/// </summary>

public override void Draw()

{

GL.BindTexture(TextureTarget.Texture2D, Textures[current\_texture++]);

GL.Begin(BeginMode.Polygon);

GL.TexCoord2(0, 0);

GL.Vertex3(Bounds.X, Bounds.Y, 0);

GL.TexCoord2(1, 0);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y, 0);

GL.TexCoord2(1, 1);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y + Bounds.Height, 0);

GL.TexCoord2(0, 1);

GL.Vertex3(Bounds.X, Bounds.Y + Bounds.Height, 0);

GL.End();

if(current\_texture >= Textures.Length)

{

\_gameField.Shots.Remove(this);

}

}

}

}

**Код программы для класса *Fire.cs***:

using System.Drawing;

using OpenTK.Graphics.OpenGL;

using GameEngine.Helpers;

using GameEngine.Game;

namespace GameEngine.Objects

{

/// <summary>

/// Класс огня

/// </summary>

public class Fire : GameObject

{

public override int[] Textures { get; set; }

/// <summary>

/// Ссылка на игровое поле

/// </summary>

private GameObject \_parent;

/// <summary>

/// Размеры

/// </summary>

public override int Width { get { return \_parent.Width / 2; } }

public override int Height { get { return \_parent.Height / 2; } }

/// <summary>

/// Положение

/// </summary>

public override Point Location

{

get { return \_parent.Location; }

set { \_parent.Location = value; }

}

/// <summary>

/// Границы

/// </summary>

public new Rectangle Bounds

{

get

{

return new Rectangle(Location.X - Width / 2, Location.Y - Height / 2, Width, Height);

}

}

/// <summary>

/// Конструктор

/// </summary>

public Fire(GameObject parent)

{

\_parent = parent;

Textures = TextureRepository.Get("fire");

}

/// <summary>

/// Функция рисования

/// </summary>

public override void Draw()

{

GL.BindTexture(TextureTarget.Texture2D, Textures[current\_texture]);

GL.Begin(BeginMode.Polygon);

GL.TexCoord2(0, 0);

GL.Vertex3(Bounds.X, Bounds.Y, 0);

GL.TexCoord2(1, 0);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y, 0);

GL.TexCoord2(1, 1);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y + Bounds.Height, 0);

GL.TexCoord2(0, 1);

GL.Vertex3(Bounds.X, Bounds.Y + Bounds.Height, 0);

GL.End();

current\_texture = ++current\_texture < Textures.Length ? current\_texture : 0;

}

}

}

**Код программы для класса *Hidrance.cs***:

using System.Drawing;

using OpenTK.Graphics.OpenGL;

using GameEngine.Helpers;

using GameEngine.Game;

namespace GameEngine.Objects

{

/// <summary>

/// Класс абстрактного препятствия

/// </summary>

public abstract class Hindrance : GameObject

{

public override int[] Textures { get; set; }

/// <summary>

/// Проходимое или нет

/// </summary>

public override Point Location { get; set; }

}

/// <summary>

/// Класс скала

/// </summary>

public class Rock : Hindrance

{

/// <summary>

/// Непроходимое

/// </summary>

public override int Width { get { return 50; } }

public override int Height { get { return 50; } }

public Rock()

{

Textures = TextureRepository.Get("rock"); ;

}

public override void Draw()

{

GL.BindTexture(TextureTarget.Texture2D, Textures[current\_texture]);

GL.Begin(BeginMode.Polygon);

GL.Color3(Color.White);

GL.TexCoord2(0, 0);

GL.Vertex3(Bounds.X, Bounds.Y, 0);

GL.TexCoord2(1, 0);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y, 0);

GL.TexCoord2(1, 1);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y + Bounds.Height, 0);

GL.TexCoord2(0, 1);

GL.Vertex3(Bounds.X, Bounds.Y + Bounds.Height, 0);

GL.End();

}

}

/// <summary>

/// Класс отмель

/// </summary>

public class Shallow : Hindrance

{

/// <summary>

/// Проходимое

/// </summary>

public override int Width { get { return 50; } }

public override int Height { get { return 50; } }

public Shallow()

{

Textures = TextureRepository.Get("shallow");

}

public override void Draw()

{

GL.BindTexture(TextureTarget.Texture2D, Textures[current\_texture]);

GL.Begin(BeginMode.Polygon);

GL.Color3(Color.White);

GL.TexCoord2(0, 0);

GL.Vertex3(Bounds.X, Bounds.Y, 0);

GL.TexCoord2(1, 0);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y, 0);

GL.TexCoord2(1, 1);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y + Bounds.Height, 0);

GL.TexCoord2(0, 1);

GL.Vertex3(Bounds.X, Bounds.Y + Bounds.Height, 0);

GL.End();

}

}

}

**Код программы для класса *IExplodable.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GameEngine.Game;

namespace GameEngine.Objects

{

/// <summary>

/// Интерфейс взрывающегося объекта

/// </summary>

public interface IExplodable

{

/// <summary>

/// Дальность выстрела

/// </summary>

int Range { get; }

/// <summary>

/// Наносимый урон

/// </summary>

int Damage { get; }

/// <summary>

/// Зона поражения

/// </summary>

int DamageArea { get; }

/// <summary>

/// Ссылка на игровое поле

/// </summary>

GameField GameField { get; }

/// <summary>

/// Ссылка на родительский корабль

/// </summary>

Ship Parent { get; }

/// <summary>

/// Взрыв

/// </summary>

void Explode();

}

}

**Код программы для класса *Ship.cs***:

using System;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using OpenTK.Graphics.OpenGL;

using GameEngine.Helpers;

using GameEngine.Game;

using GameEngine.Input;

namespace GameEngine.Objects

{

public abstract class Ship : GameObject

{

/// <summary>

/// Событие изменения параметров корабля

/// </summary>

public event EventHandler Changed;

/// <summary>

/// Метод для вызова события извне

/// </summary>

public void OnChanged()

{

Changed?.Invoke(this, null);

}

//находится ли в движении

public abstract bool IsMoving { get; set; }

//пожар

protected Fire \_fire;

//контроллер управления

public abstract IController Controller { get; }

/// <summary>

/// Орудия

/// </summary>

public abstract Cannon[] Cannons { get; set; }

/// <summary>

/// Ссылка на игровое поле

/// </summary>

public abstract GameField GameField { get; }

/// <summary>

/// Направление движения

/// </summary>

public abstract Direction Direction { get; }

/// <summary>

/// Скорость корабля

/// </summary>

public abstract int Speed { get; }

/// <summary>

/// Боезапас

/// </summary>

public abstract int[] Ammunition { get; set; }

/// <summary>

/// Пожар на корабле, если здоровье меньше 50

/// </summary>

public abstract Fire fire { get; }

/// <summary>

/// Текущая текстура корабля

/// </summary>

private new int current\_texture

{

get

{

switch (Controller.CurrentDirection)

{

case Direction.Down:

return 0;

case Direction.Left:

return 1;

case Direction.Right:

return 2;

default:

return 3;

}

}

}

/// <summary>

/// Ширина

/// </summary>

public override int Width

{

get

{

return Controller.CurrentDirection == Direction.Up ||

Controller.CurrentDirection == Direction.Down ? 50 : 100;

}

}

/// <summary>

/// Высота

/// </summary>

public override int Height

{

get

{

return Controller.CurrentDirection == Direction.Up ||

Controller.CurrentDirection == Direction.Down ? 100 : 50;

}

}

/// <summary>

/// Здоровье

/// </summary>

public abstract int Health { get; set; }

public override Point Location { get; set; }

/// <summary>

/// Движение корабля

/// </summary>

protected abstract void Move();

/// <summary>

/// Отрисовка корабля

/// </summary>

public override void Draw()

{

Move();

GL.BindTexture(TextureTarget.Texture2D, Textures[current\_texture]);

GL.Begin(BeginMode.Polygon);

GL.TexCoord2(0, 0);

GL.Vertex3(Bounds.X, Bounds.Y, 0);

GL.TexCoord2(1, 0);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y, 0);

GL.TexCoord2(1, 1);

GL.Vertex3(Bounds.X + Bounds.Width, Bounds.Y + Bounds.Height, 0);

GL.TexCoord2(0, 1);

GL.Vertex3(Bounds.X, Bounds.Y + Bounds.Height, 0);

GL.End();

if (fire != null)

fire.Draw();

}

/// <summary>

/// Определение доступных направлений

/// </summary>

protected Direction FindAllowedDirections()

{

Direction \_ad = Direction.Down | Direction.Left | Direction.Right | Direction.Up;

Rectangle \_inflatedBounds = Bounds;

\_inflatedBounds.Inflate(Speed, Speed);

if (\_inflatedBounds.X <= 0)

\_ad ^= Direction.Left;

if (\_inflatedBounds.Y <= 0)

\_ad ^= Direction.Up;

if (\_inflatedBounds.X + \_inflatedBounds.Width >= Controller.GameField.ViewportSize.Width)

\_ad ^= Direction.Right;

if (\_inflatedBounds.Y + \_inflatedBounds.Height >= Controller.GameField.ViewportSize.Height)

\_ad ^= Direction.Down;

var \_objects = Controller.GameField.AllObjects;

var collisions = \_objects.Where(obj => obj.Bounds.IntersectsWith(Bounds) && obj.Bounds != Bounds);

foreach (var c in collisions)

{

Rectangle \_intersect = \_inflatedBounds;

\_intersect.Intersect(c.Bounds);

if (\_inflatedBounds.X == \_intersect.X && \_intersect.Width < \_intersect.Height)

\_ad ^= Direction.Left;

else if (\_inflatedBounds.Y == \_intersect.Y && \_intersect.Width > \_intersect.Height)

\_ad ^= Direction.Up;

else if (\_inflatedBounds.X + \_inflatedBounds.Width == \_intersect.X + \_intersect.Width

&& \_intersect.Width < \_intersect.Height)

\_ad ^= Direction.Right;

else if (\_inflatedBounds.Y + \_inflatedBounds.Height == \_intersect.Y + \_intersect.Height

&& \_intersect.Width > \_intersect.Height)

\_ad ^= Direction.Down;

}

return \_ad;

}

/// <summary>

/// На мели корабль или нет

/// </summary>

/// <returns></returns>

protected bool InLowWater()

{

return GameField.Shallows.FirstOrDefault(obj => obj.Bounds.IntersectsWith(Bounds)) != null;

}

}

}

**Код программы для класса *Torpedo.cs:***

using System.Drawing;

using System.Linq;

using GameEngine.Game;

using GameEngine.Input;

namespace GameEngine.Objects

{

/// <summary>

/// Класс абстрактной торпеды

/// </summary>

public abstract class Torpedo : Ammo

{

/// <summary>

/// Ширина

/// </summary>

public override int Width

{

get

{

return ShotDirection == Direction.Up ||

ShotDirection == Direction.Down ? 10 : 40;

}

}

/// <summary>

/// Высота

/// </summary>

public override int Height

{

get

{

return ShotDirection == Direction.Up ||

ShotDirection == Direction.Down ? 40 : 10;

}

}

/// <summary>

/// Базовая скорость

/// </summary>

protected abstract int BaseSpeed { get; }

/// <summary>

/// Реальная скорость с учетом нахождения на отмели

/// </summary>

public override int Speed

{

get

{

return 10;

}

}

public abstract void SetTexture();

public override GameField GameField { get; set; }

public override Point Location { get; set; }

public override Ship Parent { get; set; }

public override Direction ShotDirection { get; set; }

}

}

**Код программы для класса *UnitTest.cs:***

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using BattleShips;

using GameLibrary.Ammunition;

using GameEngine.Objects;

namespace UnitTest

{

[TestClass]

public class UnitTest

{

[TestMethod]

public void TestHighDamageTorpedoDecoratorDamage()

{

SimpleTorpedo torpedo = new SimpleTorpedo(null, new System.Drawing.Point(100, 100), null, 0);

var torpedoDecorator = new HighDamageTorpedoDecorator(torpedo);

Assert.IsTrue(torpedo.Damage == torpedoDecorator.Damage - 15);

}

[TestMethod]

public void TestHighDamageTorpedoDecoratorRange()

{

SimpleTorpedo torpedo = new SimpleTorpedo(null, new System.Drawing.Point(100, 100), null, 0);

var torpedoDecorator = new HighDamageTorpedoDecorator(torpedo);

Assert.IsTrue(torpedo.Range == torpedoDecorator.Range + 100);

}

[TestMethod]

public void TestHighDamageTorpedoDecoratorDamageArea()

{

SimpleTorpedo torpedo = new SimpleTorpedo(null, new System.Drawing.Point(100, 100), null, 0);

var torpedoDecorator = new HighDamageTorpedoDecorator(torpedo);

Assert.IsTrue(torpedo.DamageArea == torpedoDecorator.DamageArea - 25);

}

[TestMethod]

public void TestHighRangeTorpedoDecoratorRange()

{

SimpleTorpedo torpedo = new SimpleTorpedo(null, new System.Drawing.Point(100, 100), null, 0);

var torpedoDecorator = new HighRangeTorpedoDecorator(torpedo);

Assert.IsTrue(torpedo.Range == torpedoDecorator.Range - 100);

}

}

}

**Код программы для класса *AmmoAntiarcraft.cs:***

using System.Drawing;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

/// <summary>

/// Класс конкретного боезапаса 1

/// Классы Ammo2, Ammo3 идентичны Ammo1

/// </summary>

public class AmmoAntiarcraft : Ammo

{

/// <summary>

/// Стоимость снаряда в магазине

/// </summary>

public static int Cost

{

get { return 10; }

}

/// <summary>

/// Дальность выстрела

/// </summary>

public override int Range

{

get { return 500; }

}

/// <summary>

/// Урон

/// </summary>

public override int Damage

{

get { return 50; }

}

/// <summary>

/// Зона поражения

/// </summary>

public override int DamageArea

{

get { return 100; }

}

/// <summary>

/// Скорость

/// </summary>

public override int Speed { get; } = 10;

public override GameField GameField { get; set; }

public override Point Location { get; set; }

public override Ship Parent { get; set; }

public override Direction ShotDirection { get; set; }

/// <summary>

/// Конструктор

/// </summary>

public AmmoAntiarcraft(GameField field, Point startLocation, Ship parent, Direction direction)

{

Location = startLocation;

GameField = field;

Parent = parent;

ShotDirection = direction;

Textures = TextureRepository.Get("ammo1");

}

}

}

**Код программы для класса *AmmoHighExplosive.cs:***

using System.Drawing;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

public class AmmoHighExplosive : Ammo

{

public static int Cost

{

get { return 10; }

}

public override int Range

{

get { return 400; }

}

public override int Damage

{

get { return 25; }

}

public override int DamageArea

{

get { return 80; }

}

public override int Speed { get; } = 10;

public override GameField GameField { get; set; }

public override Point Location { get; set; }

public override Ship Parent { get; set; }

public override Direction ShotDirection { get; set; }

public AmmoHighExplosive(GameField field, Point startLocation, Ship parent, Direction direction)

{

Location = startLocation;

GameField = field;

Parent = parent;

ShotDirection = direction;

Textures = TextureRepository.Get("ammo3");

}

}

}

**Код программы для класса *AmmoShrapnel.cs:***

using System.Drawing;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

public class AmmoShrapnel : Ammo

{

public static int Cost

{

get { return 10; }

}

public override int Range

{

get { return 350; }

}

public override int Damage

{

get { return 25; }

}

public override int DamageArea

{

get { return 80; }

}

public override int Speed { get; } = 10;

public override GameField GameField { get; set; }

public override Point Location { get; set; }

public override Ship Parent { get; set; }

public override Direction ShotDirection { get; set; }

public AmmoShrapnel(GameField field, Point startLocation, Ship parent, Direction direction)

{

Location = startLocation;

GameField = field;

Parent = parent;

ShotDirection = direction;

Textures = TextureRepository.Get("ammo2");

}

}

}

**Код программы для класса *HighDamageTorpedoDecorator.cs:***

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

public class HighDamageTorpedoDecorator : TorpedoDecorator

{

public HighDamageTorpedoDecorator(Torpedo torpedo) : base(torpedo) { }

public static int Cost

{

get { return 100; }

}

public override int Range => torpedo.Range - 100;

public override int Damage => torpedo.Damage + 15;

public override int DamageArea => torpedo.DamageArea + 25;

public override void SetTexture()

{

Textures = TextureRepository.Get("torpedo2");

}

}

}

**Код программы для класса *HighRangeTorpedoDecorator.cs:***

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

public class HighRangeTorpedoDecorator : TorpedoDecorator

{

public HighRangeTorpedoDecorator(Torpedo torpedo) : base(torpedo)

{

}

public static int Cost

{

get { return 100; }

}

public override int Range => torpedo.Range + 100;

public override void SetTexture()

{

Textures = TextureRepository.Get("torpedo3");

}

}

}

**Код программы для класса *SimpleTorpedo.cs:***

using System.Drawing;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

public class SimpleTorpedo : Torpedo

{

public static int Cost

{

get { return 10; }

}

public override int Range

{

get { return 500; }

}

public override int Damage

{

get { return 65; }

}

public override int DamageArea

{

get { return 180; }

}

protected override int BaseSpeed

{

get { return 10; }

}

public SimpleTorpedo(GameField field, Point startLocation, Ship parent, Direction direction)

{

Location = startLocation;

GameField = field;

Parent = parent;

ShotDirection = direction;

}

public override void SetTexture()

{

Textures = TextureRepository.Get("torpedo1");

}

}

}

**Код программы для класса *TorpedoDecorator.cs:***

using System.Drawing;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ammunition

{

public abstract class TorpedoDecorator : Torpedo

{

protected Torpedo torpedo;

public TorpedoDecorator(Torpedo torpedo)

{

this.torpedo = torpedo;

}

public override int Range => torpedo.Range;

public override int Damage => torpedo.Damage;

public override int DamageArea => torpedo.DamageArea;

public override GameField GameField

{

get { return torpedo.GameField; }

set { torpedo.GameField = value; }

}

public override Point Location

{

get { return torpedo.Location; }

set { torpedo.Location = value; }

}

public override Ship Parent

{

get { return torpedo.Parent; }

set { torpedo.Parent = value; }

}

public override Direction ShotDirection

{

get { return torpedo.ShotDirection; }

set { torpedo.ShotDirection = value; }

}

protected override int BaseSpeed

{

get { return 10; }

}

}

}

**Код программы для класса *AmmoAnticraftCannon.cs:***

using GameEngine.Objects;

using GameLibrary.Ammunition;

namespace GameLibrary.Cannons

{

/// <summary>

/// Классы конкретных орудий. Все одинаковые. Только стреляют различными боеприпасами

/// </summary>

public class AmmoAnticraftCannon : Cannon

{

/// <summary>

/// Конструктор

/// </summary>

public AmmoAnticraftCannon(Ship parent)

{

Parent = parent;

}

/// <summary>

/// Выстрел. Может стрелять только если есть боезапас. Боезапас при выстреле уменьшается

/// </summary>

public override Ammo Shoot()

{

if (Parent.Ammunition[0] > 0)

{

Parent.Ammunition[0]--;

Parent.OnChanged();

return new AmmoAntiarcraft(Parent.GameField, Parent.Location, Parent, Parent.Direction);

}

return null;

}

}

}

**Код программы для класса *AmmoHighExplosiveCannon.cs:***

using GameEngine.Objects;

using GameLibrary.Ammunition;

namespace GameLibrary.Cannons

{

public class AmmoHighExplosiveCannon : Cannon

{

public AmmoHighExplosiveCannon(Ship parent)

{

Parent = parent;

}

public override Ammo Shoot()

{

if (Parent.Ammunition[2] > 0)

{

Parent.Ammunition[2]--;

Parent.OnChanged();

return new AmmoHighExplosive(Parent.GameField, Parent.Location, Parent, Parent.Direction);

}

return null;

}

}

}

**Код программы для класса *AmmoShrapnelCannon.cs:***

using GameEngine.Objects;

using GameLibrary.Ammunition;

namespace GameLibrary.Cannons

{

public class AmmoShrapnelCannon : Cannon

{

public AmmoShrapnelCannon(Ship parent)

{

Parent = parent;

}

public override Ammo Shoot()

{

if (Parent.Ammunition[1] > 0)

{

Parent.Ammunition[1]--;

Parent.OnChanged();

return new AmmoShrapnel(Parent.GameField, Parent.Location, Parent, Parent.Direction);

}

return null;

}

}

}

**Код программы для класса *TorpedoHighDamageTube.cs:***

using GameEngine.Objects;

using GameLibrary.Ammunition;

namespace GameLibrary.Cannons

{

public class TorpedoHighDamageTube : Cannon

{

public TorpedoHighDamageTube(Ship parent)

{

Parent = parent;

}

public override Ammo Shoot()

{

if (Parent.Ammunition[5] > 0)

{

Parent.Ammunition[5]--;

Parent.OnChanged();

SimpleTorpedo simpleTorpedo = new SimpleTorpedo(Parent.GameField, Parent.Location, Parent, Parent.Direction);

HighDamageTorpedoDecorator highDamageTorpedo = new HighDamageTorpedoDecorator(simpleTorpedo);

highDamageTorpedo.SetTexture();

return highDamageTorpedo;

}

return null;

}

}

}

**Код программы для класса *TorpedoHighRangeTube.cs:***

using GameEngine.Objects;

using GameLibrary.Ammunition;

namespace GameLibrary.Cannons

{

public class TorpedoHighRangeTube : Cannon

{

public TorpedoHighRangeTube(Ship parent)

{

Parent = parent;

}

public override Ammo Shoot()

{

if (Parent.Ammunition[4] > 0)

{

Parent.Ammunition[4]--;

Parent.OnChanged();

SimpleTorpedo simpleTorpedo = new SimpleTorpedo(Parent.GameField, Parent.Location, Parent, Parent.Direction);

HighRangeTorpedoDecorator highRangeTorpedo = new HighRangeTorpedoDecorator(simpleTorpedo);

highRangeTorpedo.SetTexture();

return highRangeTorpedo;

}

return null;

}

}

}

**Код программы для класса *TorpedoSimpleTube.cs:***

using GameEngine.Objects;

using GameLibrary.Ammunition;

namespace GameLibrary.Cannons

{

public class TorpedoSimpleTube : Cannon

{

public TorpedoSimpleTube(Ship parent)

{

Parent = parent;

}

public override Ammo Shoot()

{

if (Parent.Ammunition[3] > 0)

{

Parent.Ammunition[3]--;

Parent.OnChanged();

SimpleTorpedo simpleTorpedo = new SimpleTorpedo(Parent.GameField, Parent.Location, Parent, Parent.Direction);

simpleTorpedo.SetTexture();

return simpleTorpedo;

}

return null;

}

}

}

**Код программы для класса *NormalShip.cs***

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameLibrary.Ships

{

/// <summary>

/// Класс корабля

/// </summary>

public class NormalShip : Ship

{

public override int Speed

{

get { return 6; }

}

public override int Health { get; set; } = 250;

public override IController Controller { get; }

public override bool IsMoving { get; set; }

/// <summary>

/// Конструктор

/// </summary>

public NormalShip(IController controller, int[] ammo, bool isFirst = true)

{

Controller = controller;

Controller.Ship = this;

Ammunition = ammo;

Textures = TextureRepository.Get("ship");

Cannons = new Cannon[6];

Controller.StartMoving += \_controller\_StartMoving;

Controller.StopMoving += \_controller\_StopMoving;

Controller.Shooting1 += \_controller\_Shooting1;

Controller.Shooting2 += \_controller\_Shooting2;

Controller.Shooting3 += \_controller\_Shooting3;

Controller.Shooting4 += \_controller\_Shooting4;

Controller.Shooting5 += \_controller\_Shooting5;

Controller.Shooting6 += \_controller\_Shooting6;

}

/// <summary>

/// Обработчик остановки

/// </summary>

private void \_controller\_StopMoving(object sender, EventArgs e)

{

IsMoving = false;

}

/// <summary>

/// Обработчик начала движения

/// </summary>

private void \_controller\_StartMoving(object sender, EventArgs e)

{

IsMoving = true;

}

/// <summary>

/// Обработчики стрельбы

/// </summary>

private void \_controller\_Shooting1(object sender, EventArgs e)

{

Ammo ammo = Cannons[0].Shoot();

if (ammo != null)

GameField.Shots.Add(ammo);

}

private void \_controller\_Shooting2(object sender, EventArgs e)

{

Ammo ammo = Cannons[1].Shoot();

if (ammo != null)

GameField.Shots.Add(ammo);

}

private void \_controller\_Shooting3(object sender, EventArgs e)

{

Ammo ammo = Cannons[2].Shoot();

if (ammo != null)

GameField.Shots.Add(ammo);

}

private void \_controller\_Shooting4(object sender, EventArgs e)

{

Ammo ammo = Cannons[3].Shoot();

if (ammo != null)

GameField.Shots.Add(ammo);

}

private void \_controller\_Shooting5(object sender, EventArgs e)

{

Ammo ammo = Cannons[4].Shoot();

if (ammo != null)

GameField.Shots.Add(ammo);

}

private void \_controller\_Shooting6(object sender, EventArgs e)

{

Ammo ammo = Cannons[5].Shoot();

if (ammo != null)

GameField.Shots.Add(ammo);

}

protected override void Move()

{

if (!IsMoving)

return;

Direction \_allowedDirections = FindAllowedDirections();

if (!\_allowedDirections.HasFlag(Controller.CurrentDirection))

{

IsMoving = false;

return;

}

switch (Controller.CurrentDirection)

{

case Direction.Up:

Location = new Point(Location.X, Location.Y - Speed);

break;

case Direction.Down:

Location = new Point(Location.X, Location.Y + Speed);

break;

case Direction.Left:

Location = new Point(Location.X - Speed, Location.Y);

break;

case Direction.Right:

Location = new Point(Location.X + Speed, Location.Y);

break;

}

if (InLowWater())

{

if(GameField.ShipFirst == this)

GameField.ShipFirst = new SlowShip(this);

else

GameField.ShipSecond = new SlowShip(this);

}

}

public override int[] Textures { get; set; }

public override int[] Ammunition { get; set; }

public override Cannon[] Cannons { get; set; }

public override GameField GameField

{

get

{

return Controller.GameField;

}

}

public override Direction Direction

{

get

{

return Controller.CurrentDirection;

}

}

public override Fire fire

{

get

{

if (Health > 50)

return null;

return \_fire ?? (\_fire = new Fire(this));

}

}

}

}

**Код программы для класса *ShipDecorator.cs:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using GameEngine.Game;

using GameEngine.Input;

using GameEngine.Objects;

namespace GameLibrary.Ships

{

public abstract class ShipDecorator : Ship

{

protected NormalShip ship;

public ShipDecorator(NormalShip ship)

{

this.ship = ship;

}

public override IController Controller

{

get { return ship.Controller; }

}

public override bool IsMoving

{

get { return ship.IsMoving; }

set { ship.IsMoving = value; }

}

public override int Speed

{

get { return ship.Speed; }

}

public override int Health

{

get { return ship.Health; }

set { ship.Health = value; }

}

public override int[] Ammunition

{

get { return ship.Ammunition; }

set { ship.Ammunition = value; }

}

public override Cannon[] Cannons

{

get { return ship.Cannons; }

set { ship.Cannons = value; }

}

public override GameField GameField

{

get

{

return ship.GameField;

}

}

public override Direction Direction

{

get

{

return ship.Direction;

}

}

public override Fire fire

{

get

{

return ship.fire;

}

}

public override int[] Textures

{

get { return ship.Textures; }

set { ship.Textures = value; }

}

}

}

**Код программы для класса *SlowShip.cs:***

using GameEngine.Input;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GameLibrary.Ships

{

public class SlowShip : ShipDecorator

{

public override int Width => ship.Width;

public override int Height => ship.Height;

public SlowShip(NormalShip ship) : base(ship) { }

public override Point Location

{

get { return ship.Location; }

set { ship.Location = value; }

}

public override int Speed

{

get { return ship.Speed - 4; }

}

protected override void Move()

{

if (!IsMoving)

return;

Direction \_allowedDirections = FindAllowedDirections();

if (!\_allowedDirections.HasFlag(Controller.CurrentDirection))

{

IsMoving = false;

return;

}

switch (Controller.CurrentDirection)

{

case Direction.Up:

Location = new Point(Location.X, Location.Y - Speed);

break;

case Direction.Down:

Location = new Point(Location.X, Location.Y + Speed);

break;

case Direction.Left:

Location = new Point(Location.X - Speed, Location.Y);

break;

case Direction.Right:

Location = new Point(Location.X + Speed, Location.Y);

break;

}

if (!InLowWater())

{

if (GameField.ShipFirst == this)

GameField.ShipFirst = base.ship;

else

GameField.ShipSecond = base.ship;

}

}

}

}

# **ПРИЛОЖЕНИЕ В**

(обязательное)

**Руководство пользователя**

1. Введение.

Разработанное программное приложение предназначено для запуска на ОС не ниже *Windows* *7*. Игра обладает минимальным порогом вхождения для пользователей.

Игровое приложение обладает следующим функционалом:

– меню покупки боеприпасов;

* игровой уровень c различными препятствиями;
* передвижение игроков;
* стрельба между игроками.

Для использования программного приложения пользователь должен быть ознакомлен с:

* настоящим руководством пользователя;
* правилами использования ЭВМ.

1. Назначение и условия применения.

Разработанное программное приложение предназначено для игры двух игроков на одном экране. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel Core i5-8300H* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора *LG* с разрешением не менее 1640x750;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;

1. Подготовка к работе.

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

1. Описание операций.

При запуске приложения открывается интерфейс, который предлагает игрокам начать игру. Далее игроков переносят в игровое меню, где игроки должны на определённую сумму купить боеприпасов, после чего игроки могут начать игру. Цель каждого игрока заключается в уничтожении корабля врага. В процессе игры можно передвигаться по игровому полю и уклоняться от снарядов и торпед противника.

В процессе игры можно встретить два вида препядствия:

* непроходимые скалы;
* отмель, замедляющая игрока.

Информация о текущем количестве боеприпасов, количестве здоровья, находится информационных панелях по краям игрового окна.

1. Аварийные ситуации

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

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

1. Рекомендации по освоению.

Заранее изучить работу с клавиатурой персонального компьютера. Запомнить расположение клавиш, необходимых для игры.

# **ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство** **программиста**

1. Назначения и условия применения программы.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel Core i5-8300H* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора *LG* с разрешением не менее 1640x750;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;

1. Характеристики программы.

Для запуска приложения не требуется никаких дополнительных настроек. Для запуска решения необходимо среда разработки *Visual Studio* с установленным фреймворком .*NET* и библиотекой *OpenTK.*

1. Обращение к программе.

Приложение запускается путём открытия файла *BattleShip.exe*, находящегося в папке *BattleShip*.

1. Входные и выходные данные.

В данной программе в качестве входных данных используется ввод с клавиатуры кнопок управления игровым процессом. В качестве выходных выступает окно отображения игры.

1. Сообщения.

При окончании игры выводится победивший игрок. В процессе работы приложения игровая статистика выводится в информационные лейблы каждого игрока.

# **ПРИЛОЖЕНИЕ Д**

(обязательное)

**Руководство системного программиста**

1. Общие сведения о программе.

Разработанное игровое приложение предназначено для игры между двумя игроками на одном компьютере. Разработанное игровое приложение предназначено для развития внимания и реакции, а также для развлечения. Также развивает концентрацию и внимание и значительно улучшает память, позволяя запоминать всё большие объёмы информации.

Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

* центральный процессор *Intel Core i5-8300H* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора *LG* с разрешением не менее 1640x750;
* операционная система *Windows* 7 и выше;
* 100 Мб оперативной памяти;

1. Структура программы.

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

1. Настройка программы.

Для запуска приложения не требуется никаких дополнительных настроек. Для запуска решения необходимо среда разработки *Visual Studio* с установленным фреймворком .*NET* и библиотекой *OpenTK.*

1. Проверка программы.

Для верификации программного средства были реализованы модульные тесты. Для запуска отладки в среде разработки *Visual Studio* необходимо запустить выполнение модульных тестов. По окончанию процесса отладки будет выдан отчет о результатах тестирования приложения.