МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-12 Информационные системы и технологии (в игровой индустрии)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

на тему: «Игра «Перестрелка», реализованная средствами WPF с использованием графики OpenGL»

Исполнитель: студент группы ИТИ-21

Кулешов И.В.

Руководитель: зав. кафедрой ИТ

Курочка К.С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2020

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc40209827)

[1 СРЕДСТВА РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ 5](#_Toc40209828)

[1.1 Графическая библиотека *DirectX* 5](#_Toc40209829)

[1.2 Графическая библиотека *OpenGL* 6](#_Toc40209830)

[1.3 Преимущества графических библиотек *OpenGL* и *DirectX* 7](#_Toc40209831)

[1.4 Игровой жанр «артиллерийская игра» 8](#_Toc40209832)

[1.5 Игры жанра «артиллерийская игра» 9](#_Toc40209833)

[1.6 Разработка компьютерных игр в настоящее время 12](#_Toc40209834)

[2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 15](#_Toc40209835)

[2.1 Данные для проектирования игрового приложения 15](#_Toc40209837)

[2.2 Структура программного обеспечения 16](#_Toc40209838)

[2.3 Схема данных игрового приложения 19](#_Toc40209839)

[3 ВЕРИФИКАЦИЯ И АПРОБАЦИЯ ПРИЛОЖЕНИЯ 24](#_Toc40209840)

[3.1 Принцип работы игрового приложения 24](#_Toc40209841)

[3.2 Результаты тестирования игрового приложения 29](#_Toc40209842)

[3.3 Результаты верификации игрового приложения 30](#_Toc40209843)

[ЗАКЛЮЧЕНИЕ 33](#_Toc40209844)

[Список используемых источников 34](#_Toc40209845)

[ПРИЛОЖЕНИЕ А Листинг программы «Перестрелка» 35](#_Toc40209846)

[ПРИЛОЖЕНИЕ Б Руководство пользователя 98](#_Toc40209847)

[ПРИЛОЖЕНИЕ В Руководство программиста 99](#_Toc40209848)

[ПРИЛОЖЕНИЕ Г Руководство системного программиста 100](#_Toc40209849)

[ПРИЛОЖЕНИЕ Д Схема использования паттерна «Декоратор» 102](#_Toc40209850)

# ВВЕДЕНИЕ

Компьютерные игры по своей сути являются предметом искусства. А искусством люди увлекаются с незапамятных времен благодаря тому, что оно позволяет снять напряжение, расслабиться и забыть о проблемах окружающего мира на какое-то время.

Индустрия видеоигр очень сильно развилась в наше время, что позволяет ей быстро и легко предоставить людям игры любого жанра и тем самым удовлетворить предпочтения практически каждого игрока. Вычислительная мощность современных цифровых устройств способна воспроизводить компьютерную графику практически любой сложности, что помогает играм конкурировать с киноиндустрией в качестве изображения.

В современном мире компьютерные игры способны нести помимо развлекательного характера ещё и обучающий. Существует множество игр, созданных специально для учебных учреждений, которые способны помочь людям изучить как базовые науки, вроде математики и естественного языка, так и игры, которые помогают в понимании более сложных физических или биологических процессов. Одни из первых подобных игр начали появляться в Японских школах и там они предназначались для увеличения заинтересованности учащихся в процессе обучения.

На данный момент, игры являются одним из наиболее популярных и легкодоступных источников удовольствий. Имеется возможность играть в них как одному, так и в компании других людей, что помогает людям совершенствовать свои социальные навыки. Видеоигры доступны практически на всех современных гаджетах: компьютерах, смартфонах, телевизорах, консолях, электромобилях и даже на электронных часах. В видеоигры нравится играть людям всех возрастов, так как видеоигры достаточно веселые и, к тому же, многие из них имеют достаточно низкий порог вхождения.

Прибыль из видеоигр можно получить различными способами, например, некоторые компании предпочитают сделать игру бесплатной, но добавить в нее микротранзакции и рекламу, что позволяет получить доход даже из тех пользователей, которые не рассчитывали платить за видеоигру. Из вышеперечисленных фактов становится ясно, что разработка игр является весьма актуальным и прибыльным занятием на данный момент.

# 1 СРЕДСТВА РАЗРАБОТКИ ИГРОВЫХ ПРИЛОЖЕНИЙ

## 1.1 Графическая библиотека *DirectX*

DirectX – это набор компонентов для операционной системы Windows, который позволяет различным компьютерным приложениям, в том числе компьютерным играм, взаимодействовать с видео и аудио оборудованием напрямую. Игры, поддерживающие DirectX, используют встроенные в оборудование функции акселерации мультимедиа более эффективно, благодаря чему производительность выполнения мультимедийных задач многократно повышается.

DirectX был создан специально для операционной системы Windows компанией Microsoft. В настоящее время данная технология используется в том числе и на игровой консоли Microsoft Xbox, но недоступна для других платформ. На рисунке 1.1 изображена конвейерная архитектура DirectX:

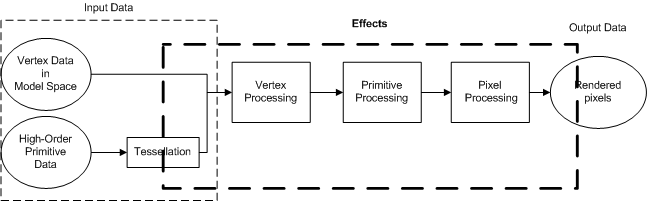


Рисунок 1.1 – Конвейерная архитектура DirectX

DirectX базируется на технологии COM (The Microsoft Component Object Model). COM – это объектно-ориентированная модель программирования [1, с. 24], которая используется различными технологиями, в том числе DirectX. COM-объект в сущности является инкапсулированным компонентом функциональности, который используется приложениями для выполнения одной и более различных задач. Компоненты COM упаковываются в двоичные файлы, которые называют COM server. На практике работа с COM похожа на вызов методов для объекта C++.

Во время работы с приложениями, созданными с использованием DirectX, применяются указатели на интерфейсы объектов. Работа с объектами осуществляется благодаря вызову различных методов его интерфейса. Создание объектов происходил посредством вызова методов интерфейса контекстного окна [2, c. 287].

Начиная с 2002 года, технология DirectX улучшила и расширила поддержку шейдеров. Шейдер – компьютерная программа, которая предназначается для исполнения на видеокартах (GPU). Для составления шейдеров используются специализированные языки программирования. Код, составленный на специальном языке, впоследствии компилируются в инструкции для CPU. Шейдеры применяются при работе с графикой и видео для определения изображения или геометрических объектов, чтобы впоследствии модифицировать изображение различными эффектами, такими как сдвиг, отражение, преломление, затемнение, поглощение и рассеивание света [3, c. 316].

## 1.2 Графическая библиотека *OpenGL*

OpenGL – это спецификация, которая определяет независимый от языка программирования интерфейс для создания приложений, использующих трехмерную или двумерную компьютерную графику. Данная спецификация разрабатывается некоммерческой организацией Khronos Group совместно с её сообществом. Большинство производителей GPU (Graphics Processing Unit), в той или иной мере, влияли на реализацию OpenGL. Данный стандарт доступен на большинстве современных платформ, что позволяет ему конкурировать с ранее описанной технологией DirectX. OpenGL является основополагающим компонентом при взаимодействии с GPU в операционных системах Mac и Linux. Данный продукт можно считать промежуточной частью между пользовательским уровнем и аппаратной частью, что позволяет предоставить пользователям единый интерфейс на различных платформах, используя при этом возможности аппаратной поддержки [4, c. 40]. На рисунке 1.2 представлена графическая схема конвейерной архитектуры OpenGL:

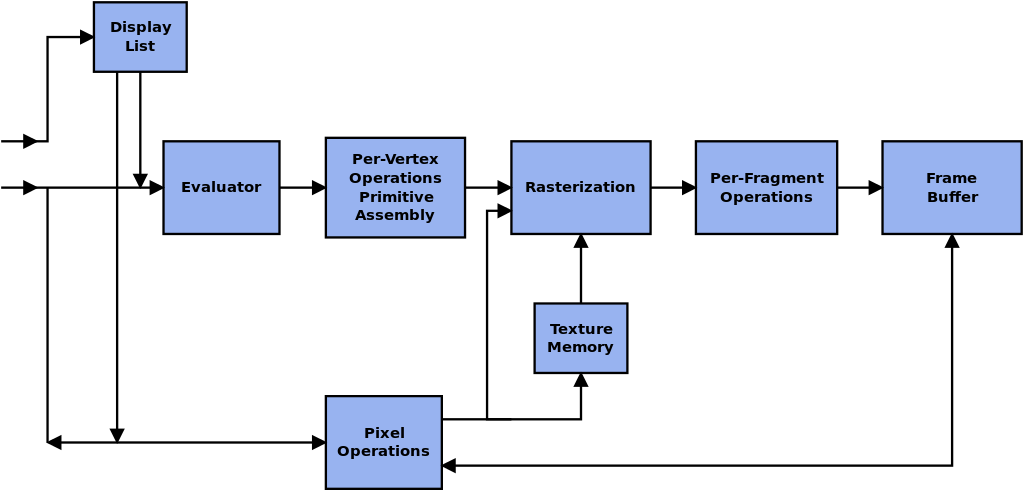


Рисунок 1.2 – Конвейерная архитектура OpenGL.

Реализация OpenGL для работы использует стандартные функции языка программирования С. Для данных функций существует стандартизированный ABI (Application Binary Interface), что, в свою очередь, означает, что имеется возможность использовать OpenGL из любого языка программирования, который имеет поддержку вызовов функций из native библиотек [5, c. 21].

В основе большинства возможностей OpenGL лежит машина состояний (конечный автомат), а результат вызовов функций OpenGL в свою очередь зависит от её внутреннего состояния, которое способно изменить контекст графического приложения. Для того, чтобы получить возможность манипулировать конкретным объектом в OpenGL для начала требуется выбрать данный объект в качестве текущего, и только после этого его можно будет модифицировать.

Реализации OpenGL позволяют пользователям добавлять различные расширения к основной спецификации. Приложение может получить доступ к списку поддерживаемых OpenGL расширений прямо во время выполнения приложения, а также проверить на доступность те расширения, которые приложение желает использовать. Большая часть функционала OpenGL – это различные расширения [6, c. 277]. Такой подход позволяет приложениям пользователей использовать функционал, который производитель только недавно реализовал в своем драйвере и задокументировал, не дожидаясь включения данных возможностей в официальную спецификацию. Наиболее удачные расширения со временем могут стать частью основной спецификации OpenGL. Таким образом, каждая новая версия OpenGL – это набор новых интегрированных расширений для более старой версии. Данный подход позволяет программисту не придавать значения версии OpenGL, которую он использует, так как для программы главное лишь то, какие доступны расширения.

## 1.3 Преимущества графических библиотек *OpenGL* и *DirectX*

С целью выбрать для поставленной задачи наилучшую графическую библиотеку рассматриваются все достоинства и недостатки графических библиотек OpenGL и DirectX.

К преимуществам графической библиотеки OpenGL можно отнести следующие факторы:

* библиотека способна предоставить пользователю доступ к функциям аппаратного ускорения для 3D-графики;
* библиотека является кроссплатформенной, что существенно расширяет область ее применения;
* библиотека предоставляет возможность для использования современного функционала с помощью специальных расширений, что позволяет не ждать включения их в основную спецификацию;
* функционал DirectX 10 и 11 версии доступен в OpenGL для любой платформы;
* основная спецификация постоянно обновляется, что позволяет ей оставаться в современной и актуальной форме.

Несмотря на большое количество преимуществ OpenGL, данная графическая библиотека также имеет и ряд недостатков:

* драйвер OpenGL требует самостоятельной установки с официального сайта производителя, в связи с тем, что он не идёт в комплекте с драйверами операционной системы Windows;
* не предоставляет пользователю ничего, кроме средств работы, так как данная библиотека не является библиотекой высокого уровня.

Стоит также рассмотреть достоинства графической библиотеки DirectX:

* библиотека также способна предоставлять доступ к функциям аппаратного ускорения 3D-графики;
* современное и унифицированное API (Application Programming Interface), которое постоянно развивается компанией Microsoft;
* поставляется в комплекте с драйверами для операционной системы Windows.

DirectX в свою очередь имеет и ряд недостатков:

* может использоваться только на операционных системах Windows и на консолях Xbox;
* для того, чтобы использовать все возможности современных версий требуется полностью перейти на неё, вследствие чего может возникнуть надобность обновить код уже созданного приложения;
* также не является библиотекой высокого уровня, так как не предоставляют пользователю ничего, кроме средств для работы с графикой.

Взвесив все преимущества и недостатки обеих графических библиотек, можно прийти к выводу, что наилучшим вариантом для отображения графических объектов в разрабатываемом приложении является графическая библиотека OpenGL, так как в результате сравнения было выяснено, что она имеет больше достоинств, чем DirectX. После того, как было выбрано средство для разработки игрового графического приложения, следует рассмотреть игровой жанр, которому будет соответствовать разработанное приложение.

## 1.4 Игровой жанр «артиллерийская игра»

Артиллерийская игра – это жанр видеоигр, который основывается на тактических сражениях, наиболее важной составляющей которых является расчет параболической траектории полета снаряда, которая, в свою очередь, зависит от множества параметров, таких как: мощности выстрела, силы ветра, углом выстрела и другого.

В артиллерийских играх весь игровой процесс строится на стрельбе из огнестрельного оружия (в частности, артиллерийских установок). Целью игр данного жанра является точное поражение целей, путем построения траектории полета снаряда, на которую может повлиять множество факторов. Цели, которые игрок должен поразить, в подобных играх располагаются на определенном расстоянии и могут быть скрыты разного рода препятствиями. В связи с этими факторами стрельба зачастую производится по дуговидной траектории. В большинстве игр жанра учитывается ветер, а также характеристики вооружения и защиты. Довольно часто разработчики комбинируют игровой процесс данного жанра с игровым процессом, характерным для других игровых жанров, таких как «платформер» или «стратегия».

Современные артиллерийские игры стремятся к созданию наиболее красочной графики, поддержке сложных физических моделей, которые будут учитывать множество различных факторов, а также большой упор делается на создание качественной онлайн составляющей игры. С развитием интернета у игроков из разных стран появилась возможность принимать участие в различных турнирах и чемпионатах по компьютерным играм.

Столь популярный жанр не смог обойти стороной и флэш-игры. Большинство таких артиллерийских игр являются аркадами, то есть, они достаточно просты и похожи на игры из игровых автоматов, но с повышенным качеством графики и более проработанной физикой объектов. К несомненным плюсам флэш-игр можно отнести возможность играть в них бесплатно. Также к преимуществам таких игр можно отнести небольшой размер и простоту. Многие флэш-игры также поддерживают игру вдвоем или игру по сети, что является существенным фактором для тех игроков, которым не нравится играть в игры в одиночку.

## 1.5 Игры жанра «артиллерийская игра»

Первые игры жанра «артиллерийская игра» имели текстовый формат. В этих играх артиллерия моделировалась исходя из введенных пользователем данных. Одной из первых игр жанра «артиллерийская игра» была Artillery, созданная Майклом Форманом на языке BASIC и опубликованная в журнале Creative Computing в 1976 году. После чего эта игра была воссоздана и дополнена М. Е. Лайаном и Брайеном Вестом в 1977 году и стала известна под названием War 3. Впоследствии жанр получил развитие благодаря тому, что начали создаваться игры с более удобным графическим интерфейсом, который позволял лучше оценивать траекторию полета снарядов. В 1980 году появилась ранняя версия игры Artillery для платформы Apple II. Этот вариант был написан на Applesoft BASIC имел все преимущества оригинальной игры, но также позволял игрокам видеть графическое представление танков, поля боя и ландшафта. Также в версии для Apple II была добавлена механика ветра, который влиял на траекторию полета снарядов. На рисунке 1.4 представлен интерфейс игры Artillery для Apple II.



Рисунок 1.4 – Интерфейс игры Artillery для Apple II.

Вскоре после появления графической версии игры Artillery появилась версия данной игры для консолей. Игра получила название «Smithereens!» и была выпущена в 1982 году для Magnavox Odyssey. Данная игра существенно выделялась на фоне других игр подобного жанра, так как в ней присутствовал мультиплеер на двух игроков. Игроки управляли катапультами, которые располагались по разным сторонам защитной стены и метали артиллерийские снаряды. Данная игра не была пошаговой, как большинство игр подобного жанра.

С увеличением популярности PC, игры данного жанра начали появляться и на данной платформе. Одной из первых игр жанра «артиллерийская игра» на платформе PC была Artillery Combat, которая была создана и выпущена Рэдом Делародели в 1988 году. Игра включала в себя танковые бои и случайно генерируемый ландшафт. В 1990 году Кеннетом Морсом была выпущена игра для MS-DOS под названием Tank Wars. Эта игра имела большое количество нововведений, которые ранее не были свойственны жанру «артиллерийская игра». В число нововведений вошла возможность покупать вооружение и защиту за заработанные очки, игра против большого количества игроков и соперников под управлением искусственного интеллекта, который имел различные модели поведения. Игра представляла из себя сражение неподвижных танков на случайно генерируемом двухмерном ландшафте. Танки в игре также располагались случайным образом. В зависимости от выбранной сложности, танки под управлением компьютера имели различную стратегию поведения. Также присутствовала возможность командной игры. Целью игры было уничтожение всех враждебных танков. Для достижения этой цели необходимо было регулировать угол наклона, силу выстрела, а также нужно было учитывать силу ветра, который существенно влиял на траекторию полета артиллерийского снаряда. Сражение в игре продолжалось до тех пор, пока в живых не останется один танк или одна команда танков. После окончания сражения игроки могли потратить заработанные очки на закупку вооружения и защиты. На рисунке 1.5 представлен скриншот игрового процесса Tank Wars.

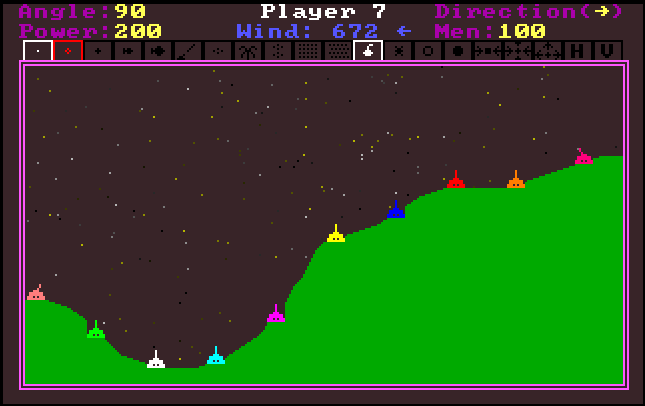


Рисунок 1.5 – Интерфейс игры Tank Wars для MS-DOS.

Через год после выпуска Tank Wars, в 1991 году, появилась первая версия игры Scorched Earth, которая существенно расширила игровой процесс артиллерийских игр, так как привнесла в жанр возможность передвижения, а также возможность использования различных бонусов и дополнительного снаряжения, такого как парашют или щит. Интерфейс в игре можно было настроить под свои потребности.

В 1995 году игровая компания Team17 Software выпустила первую версию культовой игры Worms для Amiga. В играх серии Worms игрок управляет отрядом червей на разрушаемом ландшафте и сражается с отрядами других игроков или искусственного интеллекта. Игры данной серии отличаются своим ярким визуальным стилем с большим количеством мультипликационных анимации. В дальнейшем игры серии Worms претерпели большое количество изменений и нововведений. Например, существуют игры серии, в которых действие происходит в необычном для жанра трехмерном измерении. На рисунке 1.6 изображен игровой процесс современной версии игры Worms W.M.D, выпущенной в 2016 году.



Рисунок 1.6 – Интерфейс игры Worms W.M.D

## 1.6 Разработка компьютерных игр в настоящее время

Из года в год популярность компьютерных игр возрастает, так как все больше и больше людей в мире играют в компьютерные игры. Подобный спрос на видеоигры может являться результатом технологического развития, а также следствием распространения персональных компьютеров, планшетов и мобильных устройств среди рядовых пользователей. По этой причине рынок компьютерных развлечений стал все более расширяющимся, что является веским аргументом в пользу проектов по созданию компьютерных игр. Рынок игр увеличивается в геометрической прогрессии. Наиболее удачные игровые проекты могут продаваться миллионными тиражами.

Зарождение индустрии компьютерных игр началось в 1970-х годах одновременно со стартом продаж первых персональных компьютеров, рассчитанных на массового потребителя. За 50 лет индустрия компьютерных игр развилась настолько сильно, что уже во многих аспектах опережает своих прямых конкурентов: музыкальную индустрию, киноиндустрию и шоу-бизнес.

В наши дни большинство разработчиков рассматривают индустрию компьютерных игр как сектор экономики, в котором они способны зарабатывать на создании развлечений. Но создание игр – это не только способ заработка, но ещё и инструмент культуры.

Ретро игры, придуманные много лет назад, и сейчас пользуются широким спросом. Ярким примером можно считать серию игр The Legend of Zelda, в которую с удовольствием играют пользователи нескольких поколений. Первая игра серии вышла в 1986 году и уже тогда завоевала огромную популярность у пользователей. Новые игры серии выходят и в наше время, хотя за этот период они и претерпели сильные изменения как в графическом, так и в геймплейном аспекте.

На сегодняшний день возрождение старых игр является очень актуальным и прибыльным занятием, так как потенциальная аудитория игры будет включать не только новое, не знакомое с игрой, поколение игроков, но и тех людей, которые в свое время полюбили оригинальную игру и хотят испытать эмоции, схожие с теми, что они получили при первом прохождении оригинала. На рисунке 1.7 представлена динамика доходов игровой индустрии в настоящее время, а также планы развития на ближайшие несколько лет.

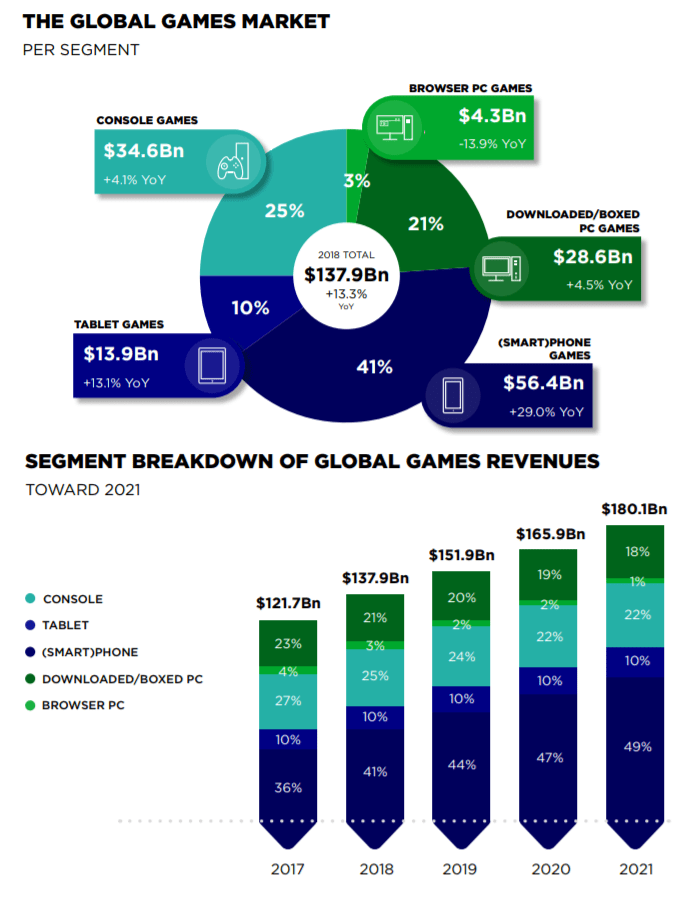


Рисунок 1.7 – Динамика доходов игровой индустрии по категориям в мире

Если брать во внимание данные, представленные на диаграмме, то можно сделать вывод, что разработка и поддержка игровых приложений будет актуальна и прибыльна ещё долгие годы, а производство новой и более мощной вычислительной техники поможет не только удовлетворить потребности конечных пользователей любых возрастных категорий, но и позволит разработчикам удешевить разработку игр, что, в свою очередь, многократно увеличит доход игровых разработчиков.

В процессе исследования игровых приложений и средств для их разработки было решено, что жанр «артиллерийская игра» станет основным для разрабатываемого приложения, а в качестве средства обработки графики была выбрана графическая библиотека OpenGL, графика которой будет отображаться в окне Windows Presentation Foundation (WPF) приложения. Подобное сочетание позволит упростить создание пользовательского интерфейса, так как можно использовать элементы управления, встроенные в WPF, такие как кнопки, полосы прогресса, поля для ввода и так далее.

# 2 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

## 2.1 Данные для проектирования игрового приложения

Игровое приложение «Перестрелка» должно иметь различные текстуры для отображения на экране пользователя, чтобы повысить положительные впечатление от игрового процесса. Для создания изображений для танков, ракет, горного хребта и заднего фона используется программа Adobe Photoshop CC 2020, в которой с нуля нарисованы игровые текстуры. Текстура танка создается в черно-белых цветах с расчетом на то, что последующий цвет будет накладываться посредством библиотеки OpenGL. Далее созданная текстура танка разделяется на две независимые составляющие – ходовую часть и башню. Разделение на составные части требуется для того, чтобы в игровом приложении была возможность анимировать каждый из компонентов по отдельности, так как во время игры самоходная артиллерийская установка может менять угол наклона своего орудия. При создании изображений для горного хребта и заднего фона применялась технология слоев, которая позволила сначала создать общую композицию с сущностями, отображающимися на экране, а впоследствии разделить эту композицию на отдельные растровые изображения. Созданные изображения сохраняются в формате bmp и впоследствии добавляются в проект Visual Studio.

Для обеспечения возможности работы с функциями OpenGL в приложении WPF загружаются необходимые графические библиотеки. Для отображения графики в игровом приложении используется библиотека OpenTK, которая является реализацией OpenGL для языка C#, и библиотека GlWpfControl, позволяющая отображать графику OpenGL в специальном компоненте WPF. Библиотека OpenTK содержит реализацию всех основных функций OpenGL, а также содержит ряд классов и перечислений, которые обеспечивают более понятный интерфейс взаимодействия с функциями OpenGL. Библиотека GlWpfControl содержит компонент для WPF-приложения, который способен отображать в себе графику, созданную средствами OpenGL.

Для реализации паттерна проектирования «декоратор», позволяющего стрелять ракетами с разными характеристиками, помимо основной красной текстуры ракеты создаются голубая и фиолетовая текстуры, используемые для скорострельных и мощных снарядов соответственно. Впоследствии паттерн «декоратор» позволяет добавить новые типы ракет, в связи с чем может потребоваться создание новых текстур для ракет. Данная проблема решается средствами OpenGL, которые позволяют накладывать другой цвет на уже загруженные текстуры.

## 2.2 Структура программного обеспечения

В ходе создания архитектуры игрового приложения использовался процесс декомпозиции. Декомпозиция представляет из себя универсальный научный метод решения сложных задач посредством разбиения задачи на несколько более простых подзадач [7]. Каждая подзадача проектируется и разрабатывается, после чего все подзадачи объединяются в общее решение поставленной задачи. Элементы, полученные в результате декомпозиции, связаны с функциями отдельных подзадач единой сложной задачи. Уникальность каждого элемента декомпозиции позволяет сформировать отличительные черты составляющих объектов.

Таким образом, перед началом разработки программного обеспечения при помощи процесса декомпозиции создается обобщенный алгоритм работы информационного приложения, который описывает последовательность срабатывания его функций. Первым делом происходит инициализация приложения WPF, которое запускается сразу после входа в программу. Следом за инициализацией компонентов WPF происходит создание окна OpenGL и инициализация всех настроек данного окна. После создания и инициализации окна OpenGL требуется инициализировать сцену для отрисовки, что включает в себя загрузку с жесткого диска текстур, которые будут использоваться в игровом приложении. Следом за загрузкой текстур, на экране пользователя отображается главное меню игры, из которого пользователь может попасть в меню покупки боеприпасов. Совершив закупку боезапаса каждым из игроков, требуется создать на инициализированной сцене игровые объекты и передать им загруженные текстуры. Как только игровые объекты будут инициализированы, следует заполнить инвентарь игроков ракетами, которые будут использоваться в последующем сражении. После полной инициализации всех игровых объектов, игровое приложение может начать отрисовывать созданную сцену в окне OpenGL. Во время отрисовки сцены, игровым приложением нажатые клавиши для последующего обновления состояний объектов. Каждый объект обновляется в зависимости от происходящего на сцене и в зависимости от действий игроков. Далее проверяются столкновения объектов друг с другом. При столкновении, объекты повторно обновляют свое состояние на сцене. Следующим шагом проверяются условия победы для игроков. Если ни один игрок не победил в игре, то сцена перерисовывается и снова начинается считывание клавиш. Когда один из игроков победил, на экране отображается сообщение с предложением начать игру заново. Если игроки откажутся от еще одной игровой партии, то произойдет выгрузка всех текстур и объектов из памяти и выход из приложения. Обобщенный алгоритм работы приложения представлена на рисунке 2.1.

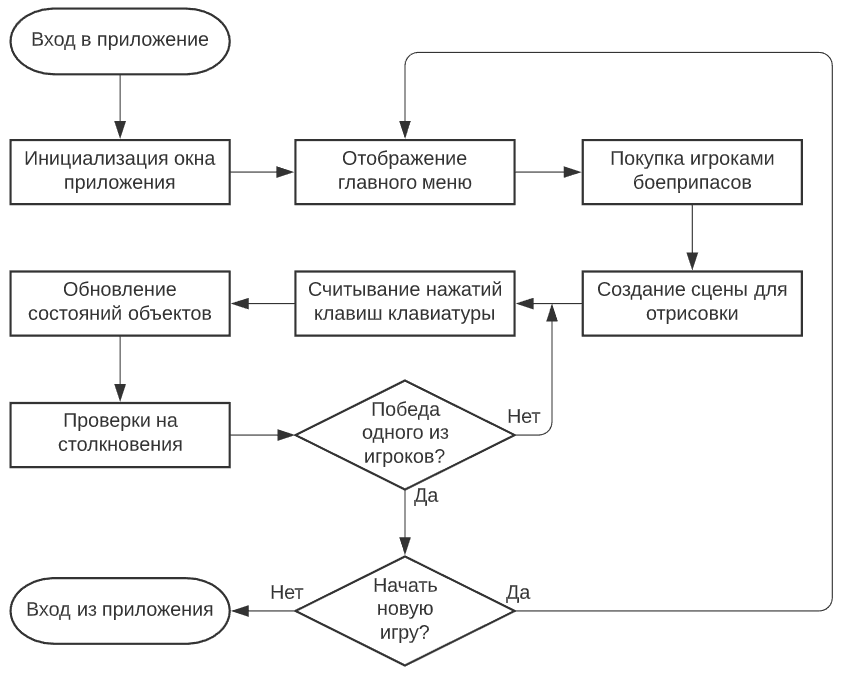


Рисунок 2.1 – Обобщенный алгоритм работы приложения

Использование процесса декомпозиции позволяет добиться решения задачи с минимальными затратами и максимальной гибкостью. Таким образом, данный подход дал возможность протестировать каждую подсистему во время разработки, что позволило выявить и устранить ряд потенциальных ошибок на начальных этапах написания программного продукта.

Для возможности в будущем легко расширить функционал программы используются паттерны (шаблоны) проектирования. Существует большое количество различных паттернов проектирования, которые решают разные задачи проектирования. В данном курсовом проекте используется структурный паттерн проектирования «декоратор», который определяет общую структуру приложения [8, c. 173].

В игровом приложении «Перестрелка» декорируемым объектом является ракета. Паттерн «декоратор» реализуется при помощи создания базового абстрактного класса Rocket, который предоставляет интерфейс ракеты и описывает ее основные характеристики: бронебойность и скорострельность. Экземпляр подкласса ракеты был помещен в абстрактный класс RocketDecorator, делегирующий запросы внутреннему компоненту-классу. Внутрь класса можно вложить любые экземпляры подклассов ракеты, что позволяет динамически добавлять функциональность этим экземплярам. Также были реализованы классы HalfDamageRocket, DoubleDamageRocket, DoubleCooldownRocket и HalfCooldownRocket, наследующиеся от класса-декоратора, которые позволяют изменять бронебойность и скорострельность ракеты соответственно. В игровом приложении данные классы комбинируются, чтобы получить более сильные ракеты с медленной перезарядкой и более быстрые ракеты с низкой бронебойностью. На рисунке 2.2 представлена схема использованного в игровом приложении структурного паттерна проектирования «декоратор».

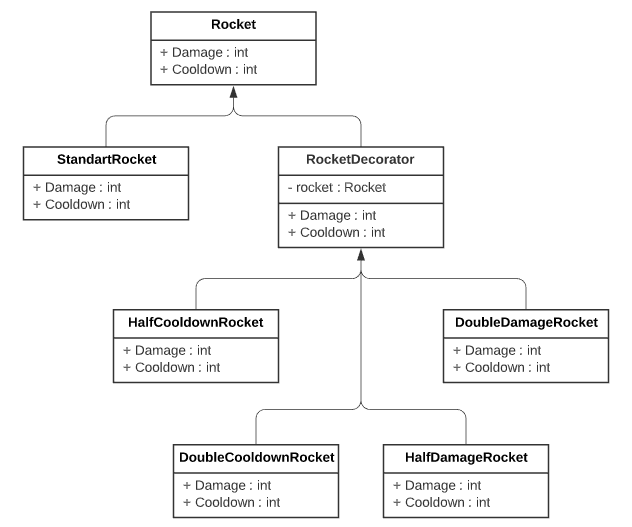


Рисунок 2.2 – Схема использованного паттерна «декоратор»

Применение паттерна «декоратор» повышает гибкость разрабатываемого программного обеспечения и позволяет динамически подключать новую функциональность [9], что предоставляет возможность добавлять новые виды боеприпасов без написания с нуля большей части кода. Данный подход сильно упрощает разработку программного продукта, так как позволяет повторно использовать уже написанные классы для решения новых задач. Также данный паттерн открывает возможности комбинировать эффекты на ракетах и создавать новые сочетания, используя уже существующие классы.

## 2.3 Схема данных игрового приложения

В процессе проектирования архитектуры игрового приложения «Перестрелка» принимается решение разбить основное приложение на 3 части – игровой движок, игровую логику и пользовательский интерфейс. Классы каждой части были помещены в отдельные проекты.

Проект с игровым движком содержит в себе базовые классы для работы игрового приложения. Наиболее важным классом данного приложения является класс GameObject, который состоит из компонентов, представленных интерфейсом IComponent и исполняемых скриптов, которые представлены абстрактным классом Script. Под компонентами в данном случае понимаются классы, реализующие интерфейс IComponent, которые хранят в себе какие-либо данные и содержат методы для обработки этих данных. Таким образом, компоненты способны хранить информацию об текстурах, которые будут отрисовываться на сцене и о положении объектов на игровом поле. Также компоненты способны хранить информацию о столкновениях объектов на сцене, которая впоследствии может обрабатываться при помощи исполняемых скриптов. Под исполняемыми скриптами следует понимать классы, наследующиеся от абстрактного класса Script, которые предназначены для исполнения каких-либо действий в каждом кадре, который отрисовывается в игровом приложении. Данные классы могут брать данные из компонентов игрового объекта и обрабатывать их в соответствии с нуждами системы. Так, например, исполняемые скрипты способны передвигать игровые объекты по сцене, создавать новые игровые объекты, уничтожать уже существующие объекты, изменять текстуру объектам, проверять объекты на столкновение и многое другое.

В проекте с игровым движком по умолчанию реализуется несколько классов-компонентов, которые необходимы большинству игровых объектов, а именно классы Texture2D, Animation2D, который наследуется от класса текстуры, Transform и Collider. Классы Texture2D и Animation2D хранят в себе данные для последующей отрисовки объекта на сцене посредством класса Renderer. Класс Transform хранит всю информацию о положении объекта на сцене, такую как позицию на сцене, угол поворота объекта. Класс Collider содержит информацию о вершинах фигуры, с которой может случиться столкновение.

Также в проекте с игровым движком содержится класс Scene, который хранит набор объектов, отрисовываемых на сцене. Каждый кадр состояние объектов на сцене обновляется посредством циклического вызова метода обновления скриптов для всех объектов на сцене. Объекты, которые должны быть добавлены или удалены из набора помещаются в отдельный список, который обрабатывает сцена после обновления состояний объектов. Данный подход позволяет избежать ошибок при переборе коллекций. После обновления всех объектов на сцене, обновленные объекты отрисовываются на сцене при помощи класса Renderer. Класс Renderer способен брать текстуры у каждого объекта на сцене и отрисовать их посредством вызовов функций библиотеки OpenGL. Текстуры объектов представлены классами Texture2D и Animation2D, которые хранят информацию о размерах текстуры и о самой текстуре, загруженной из файла. Animation2D хранит информацию о нескольких текстурах в виде массива, а также данный класс способен определять, какую именно текстуру из массива нужно отобразить в данный момент времени.

Также в классе Scene содержится ссылка на окно WPF приложения, в котором будет отрисовываться игровая сцена. Данная ссылка нужна для того, чтобы сцена могла подстроиться под размеры окна, а также использовать какие-либо элементы интерфейса, разработанные средствами WPF, такие как кнопки, полосы прогресса, элементы с текстом и так далее. На рисунке 2.3 представлена диаграмма классов движка игрового приложения.

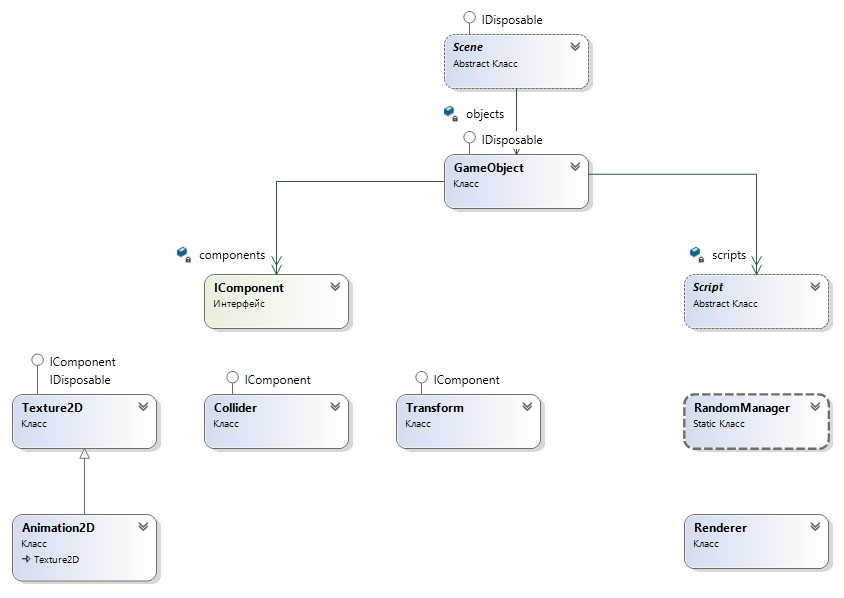


Рисунок 2.3 – Иерархия классов движка игрового приложения

После разработки движка игрового приложения разрабатывается структура проекта с игровой логикой приложения. Данный проект использует объекты из игрового движка для своей работы, но при этом расширяет их и дополняет новыми сущностями.

Главным классом в проекте с игровой логикой является класс BattleScene, который наследуется от абстрактного класса Scene из игрового движка, но при этом дополняет его реализацией инициализации основных игровых объектов на сцене. Класс BattleScene создает и инициализирует скриптами и компонентами все базовые игровые объекты, такие как танки и горы. Для корректной работы, танк должен обладать набором скриптов и компонентов, которые будут позволять ему изменять свое местоположение, изменять угол наклона орудия и совершать стрельбу по баллистической траектории. Также танку нужны компоненты, которые будут позволять контролировать оставшееся здоровье танка и управлять его боезапасом. Для управления здоровьем предусматривается класс Health, который позволяет добавлять и отнимать здоровье у игрового объекта. Для реализации инвентаря создается класс Inventory, который осуществляет возможность добавления и получения боезапаса по потребности игрока. Боезапас, доступный для игроков, описывается классами, производными от класса Rocket. Данные классы реализуют структурный паттерн проектирования «декоратор», что позволяет гибко настраивать характеристики каждой ракеты. Описание принципа работы шаблона проектирования «декоратор» описано ранее в пункте 2.3. Для того, чтобы описать корректное поведение ракет, реализуются скрипты, позволяющие добавить объекту физику, которые основаны на реальной физической модели. Также создаются скрипты, позволяющие реализовать столкновения ракеты с другими объектами с последующим взрывом и нанесением урона. Для повышения оптимизации игры также создается скрипт, который уничтожает ракету, если она вылетает за пределы игрового поля, что позволяет избавиться от лишних игровых объектов на поле, которые бы нагружали сцену и снижали общую производительность. Для отслеживания состояния игры в каждый момент времени создается специальный объект, который в каждом кадре проверяет, не уничтожен ли танк одного из игроков.

Для повышения гибкости описанной структуры добавляется класс BattleSceneSettings, который хранит конфигурацию боевой сцены, что позволяет настраивать боевую сцену во время выполнения программного обеспечения. Данный класс содержит всю информацию о танках каждого из игроков и о том, сколько у них боеприпасов каждого из типов, созданных посредством использования шаблона проектирования «декоратор». Описанная структура классов игровой логики приложения представлена на рисунке 2.4.

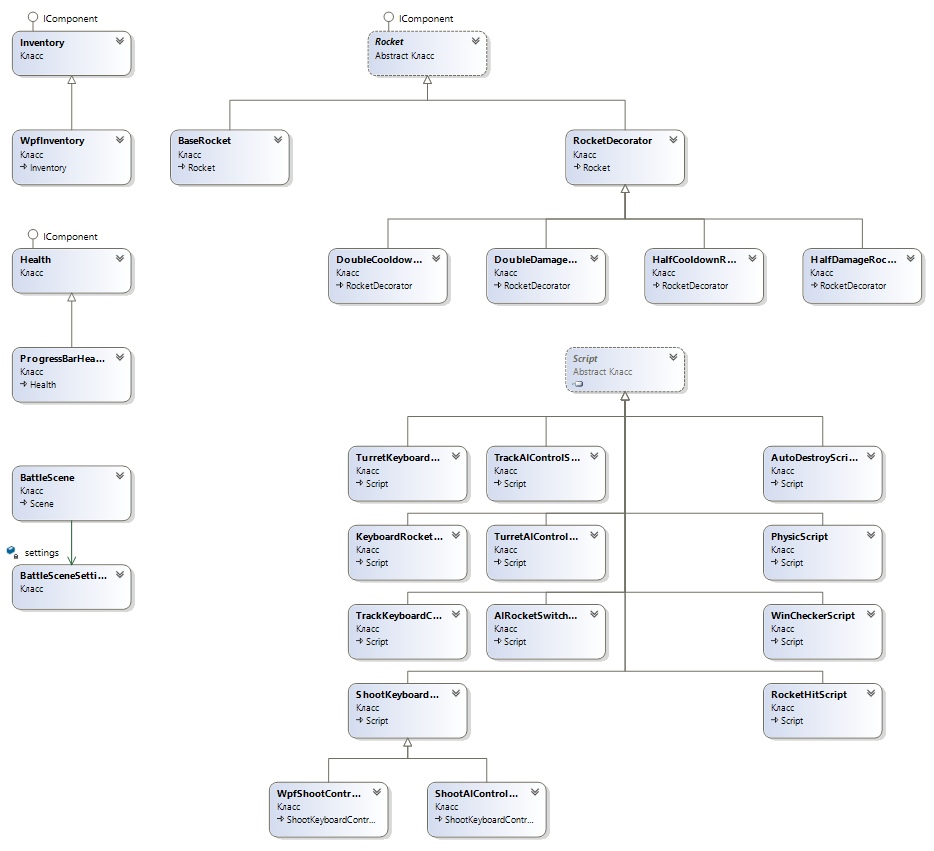


Рисунок 2.4 – Иерархия классов игровой логики приложения

В процессе проектирование архитектуры компьютерной игры «Перестрелка» создается схема данных, которая применяется для объектно-ориентированного проектирования и анализа классов приложения. Данную диаграмму можно использовать для конструирования, визуализации и документирования программных систем для последующего выявления и устранения лишних зависимостей. Сама схема данных включает в себя несколько видов строительных блоков, а именно диаграмму классов, сущности и связи между этими сущностями. Сущности являются основными элементами модели, а связи соединяют данные сущности между собой, что позволяет найти и избавиться от лишних зависимостей среди разрабатываемых классов. Диаграммы в свою очередь группируют наборы подобных сущностей в единую взаимосвязанную систему, которая показывает принцип работы разрабатываемого приложения. Таким образом, схема данных показывает набор классов и интерфейсов, и связи между ними. На разработанной схеме данных видно, что создаваемые классы связаны между собой достаточно просто, что говорит о понятности разработанной структуры приложения. На рисунке 2.5 изображена разработанная схема данных.

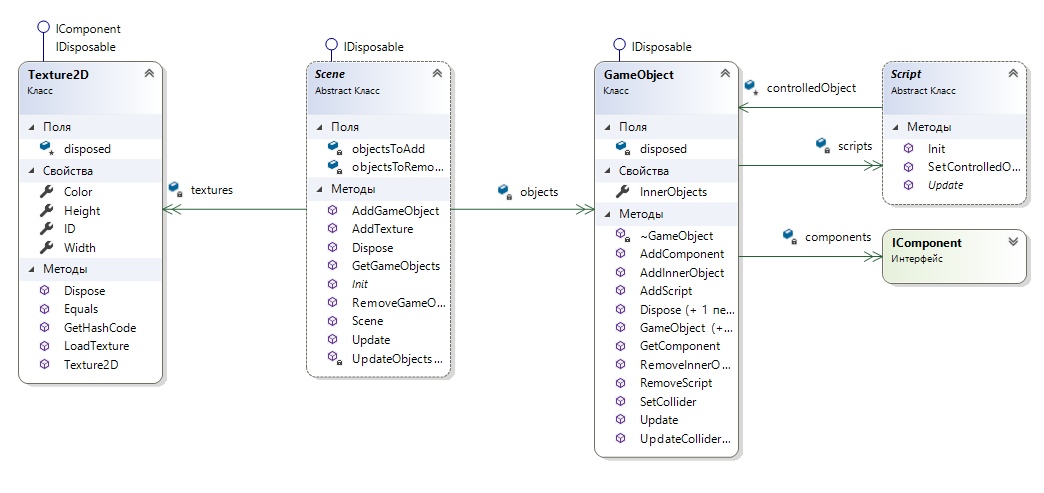


Рисунок 2.5 – Схема данных разрабатываемого приложения.

Благодаря данной схеме, приложение приобрело более логичную структуру, где объекты взаимодействуют только с теми сущностями, которые необходимы им для правильной работоспособности созданных классов. Разработка схемы данных игрового программного обеспечения позволило проанализировать структуру разрабатываемых классов и устранить лишние зависимости между объектами, которые образовались при проектировании игрового приложения.

В результате проделанной работы создается достаточно гибкая структура классов, удовлетворяющая требованиям разрабатываемого игрового приложения, которая позволяет легко расширить функционал посредством добавления новых классов. Данная структура строится таким образом, чтобы максимизировать возможность повторного использования кода, что сильно упрощает процесс разработки программного обеспечения. Впоследствии данная структура приложения воссоздается благодаря средствам объектно-ориентированного языка программирования C# в совокупности с графической библиотекой OpenGL, графические элементы которой отображаются в окне WPF приложения.

# 3 ВЕРИФИКАЦИЯ И АПРОБАЦИЯ ПРИЛОЖЕНИЯ

## 3.1 Принцип работы игрового приложения

В ходе разработки данного игрового приложения необходимо создать окно WPF приложения. Так как .Net Framework напрямую не поддерживает работу с технологией OpenGL, требуется использовать внешнюю библиотеку, реализующую основные функции OpenGL, такую как OpenTK или SharpGL. Данные библиотеки предоставляют классы для работы с функциями OpenGL в .Net-приложениях. Для разработки приложения выбирается библиотека OpenTK, так как она наиболее удачно реализует интерфейс OpenGL, дополняя его более строгой типизацией, что позволяет минимизировать ошибки программиста при разработке. Также данная библиотека достаточно проста для подключения к проекту, что позволяет сосредоточиться на реализации основного алгоритма игрового приложения.

Для отображения графики OpenGL в окне WPF требуется добавить в проект дополнительную библиотеку, реализующую компонент для отрисовки графики [10, с. 76]. В качестве библиотеки, содержащий данный компонент выбирается библиотека OpenTK.GLWpfControl, так как она наиболее просто интегрируется с библиотекой OpenTK. Для подключения компонента к окну WPF добавляется тег <glWpfControl:GLWpfControl/>, позволяющий отрисовывать внутри себя графику OpenGL (Приложение А, код программы MainWindow.xaml). Для работы с данным компонентом следует создать 2 основных события: OpenTKControl\_Ready и OpenTKControl\_Render. Первое событие срабатывает при инициализации компонента и позволяет настроить окно для дальнейшей отрисовки графики. Второе событие срабатывает при отрисовке каждого кадра.

В конструкторе класса MainWindow (Приложение А, код программы MainWindow.xaml.cs) создается игровая сцена и инициализируется начальными параметрами. Данная сцена передается на отрисовку классу Renderer (Приложение А, код программы Renderer.cs), который в каждом кадре перебирает объекты на сцене и отображает их в компоненте OpenGL. Такой подход позволяет отделить игровые объекты от логики их отрисовки, что потенциально позволяет легко подменить метод отрисовки и отрисовываемую сцену.

Все сцены игрового приложения наследуются от абстрактного класса Scene, который реализует базовый функционал каждой сцены, оставляя своим потомкам для переопределения метод Init() (Приложение А, код программы Scene.cs). Данный метод должны реализовывать все потомки класса Scene и в нем требуется создать и сконфигурировать начальное состояние всех объектов на сцене. В данном игровом приложении создается подкласс абстрактной сцены BattleScene (Приложение А, код программы BattleScene.cs), который инициализирует танки, их запас здоровья и боезапас. Также данный класс инициализирует танки способом управления: с клавиатуры или посредством искусственного интеллекта. При запуске игрового приложения инициализируется сцена с танками под управлением искусственного интеллекта и с бесконечным боезапасом и здоровьем, что позволяет создать эффектное главное меню игры, на котором происходит бесконечное сражение двух танков. На рисунке 3.1 изображено главное меню созданного игрового приложения, на фоне которого происходит бесконечное сражение двух танков под управлением искусственного интеллекта.

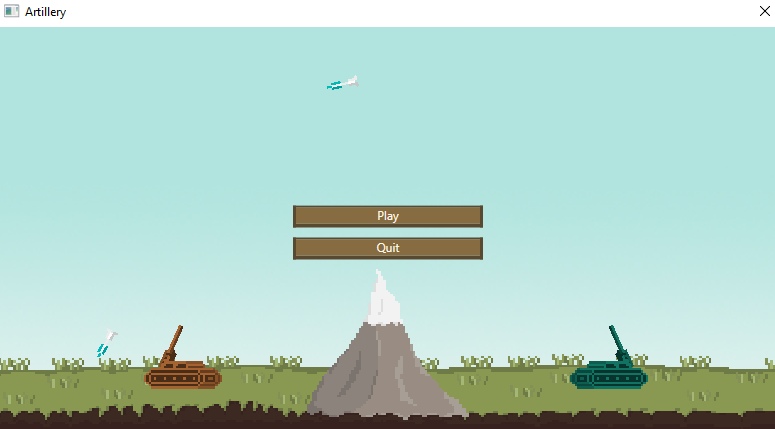


Рисунок 3.1 – Главное меню игры.

Искусственный интеллект в игре реализуется при помощи классов AIRocketSwitcherScript (Приложение А, код программы AIRocketSwitcherScript.cs), ShootAIControlScript (Приложение А, код программы ShootAIControlScript.cs), TrackAIControlScript (Приложение А, код программы TrackAIControlScript.cs) и TurretAIControlScript (Приложение А, код программы TurretAIControlScript.cs), которые добавляются игровому объекту танка и в каждом кадре изменяют его состояние таким образом, чтобы он передвигался по сцене и стрелял по врагу. Расчет траектории стрельбы осуществляется при помощи формул, которые были выведены из физических законов. Таким образом пушка под управлением искусственного интеллекта всегда нацелена на танк врага, но из-за постоянного движения танков они способны уворачиваться от выстрелов друг друга.

В случае, если игрок решит начать игру, на экране будет отображено меню с возможностью покупки ракет разного типа и разной стоимостью. Все ракеты отличаются между собой характеристиками бронебойности и скорострельности, что реализуется посредством использования паттерна проектирования «Декоратор». На рисунке 3.2 представлено меню игры с возможностью покупки боеприпасов.

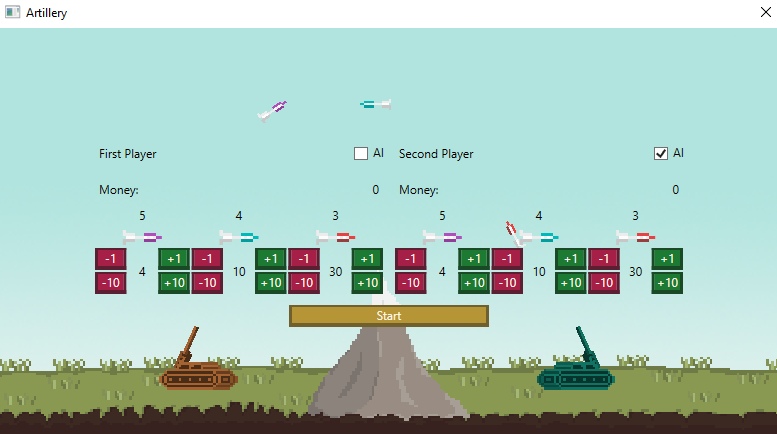


Рисунок 3.2 – Меню покупки боеприпасов.

После того, как пользователь выберет боеприпасы, которые он хочет приобрести, начинается инициализация новой боевой сцены, на которой создаются танки и устанавливается им инвентарь с выбранным пользователем боезапасом и начальное здоровье танков.

Базовый игровой объект на сцене представлен классом GameObject (Приложение А, код программы GameObject.cs). Данный класс описывает функционал игрового объекта на сцене и имеет поля List<Script> scripts и Dictionary<string, IComponent> components, позволяющие добавлять функциональность созданным игровым объектам.

Поле scripts хранит список подклассов абстрактного класса Script. Каждый класс в данном списке имеет метод void Update(TimeSpan delta), который обновляет состояние объекта, к которому он прикреплен. Данный метод циклически вызывается у всех скриптов каждый кадр посредством вызова у игрового объекта метода void Update(TimeSpan delta).

Поле components является словарем, который содержит список сущностей, которые хранят различные данные об игровом объекте. Эти данные используются скриптами, позволяя хранить информацию отдельно от логики, обрабатывающей ее. Данный подход позволяет гибко комбинировать наборы скриптов и компонентов, добиваясь различных вариаций одних и тех же объектов. Все компоненты реализуют маркерный интерфейс IComponent (Приложение А, код программы IComponent.cs), что позволяет поместить разные объекты в одну коллекцию. Наиболее важными и часто используемыми компонентами является компонент Texture2D (Приложение А, код программы Texture2D.cs), Transform (Приложение А, код программы Transform.cs) и Collider (Приложение А, код программы Collider.cs).

Класс Texture2D описывает текстуру и хранит базовую информацию о ней: идентификатор, ширину и высоту. Также данный класс имеет метод Texture2D LoadTexture(string path), позволяющий загрузить текстуру из файла. При загрузке текстуры осуществляется конфигурация ее параметров посредством вызова методов OpenGL. Данный компонент используется классом Renderer (Приложение А, код программы Renderer.cs) для отрисовки объекта на сцене. Класс Renderer имеет метод void Render(), который позволяет отрисовать список объектов на сцене. Данный метод циклически перебирает объекты на сцене, берет их компоненты Texture2D и Transform и отображает полученную текстуру в полученных координатах на сцене. Метод отрисовки также способен поворачивать отрисовываемое изображение вокруг определенной точки, что позволяет легко реализовать анимацию полета ракеты и анимацию поворота башни танка. Вращение изображения реализовано при помощи использования матриц поворота.

Класс Transform (Приложение А, код программы Transform.cs) предоставляет информацию о положении объекта на сцене, о размере объекта, о его повороте, а также о его родительском объекте. Информация о родительском объекте нужна для того, чтобы реализовать возможность вкладывать одни игровые объекты внутрь других, создавая сложные системы из вложенных объектов, которые бы имели возможность двигаться относительно друг друга, а также взаимодействовать сообща. При попытке получить позицию объекта на сцене сначала проверяется, есть ли у объекта родитель. Если родитель существует, то будет возвращена позиция объекта, смещенная на позицию родительского объекта. Такая логика позволяет двигать всю систему объектов на сцене посредством перемещения лишь одного родительского объекта.

Класс Collider реализует проверку объектов на столкновение друг с другом (Приложение А, код программы Collider.cs). Данный класс имеет метод bool CheckCollision(Collider collider), позволяющий проверить 2 объекта на пересечение друг с другом. Данный метод основан на теореме, согласно которой два выпуклых объекта не пересекаются тогда и только тогда, когда существует такая прямая, что их проекции на эту прямую не пересекаются. Для создания объекта Collider можно вызвать его конструктор, передав в него массив вершин фигуры, столкновения с которой будут проверяться. Для упрощения передвижения данных вершин по сцене был реализован метод в классе GameObject, который позволяет привязать координаты вершин фигуры к положению текстуры объекта. В разработанном игровом приложении система столкновений позволяет определять, столкнулась ли ракета с препятствием или танком. В случае столкновения ракета взрывается, а если ракета столкнулась с танком, то танку наносится урон. На рисунке 3.3 изображена анимация взрыва ракеты при столкновении с горой.



Рисунок 3.3 – Столкновение ракеты с горой.

Для реализации анимации в игровом приложении был создан класс Animation2D (Приложение А, код программы Animation2D.cs). Данный класс наследуется от класса Texture2D, расширяя его функционал посредством добавления таймера обновления анимации и массива изображений. Класс анимации, как и класс текстуры, имеет метод для загрузки изображений из файла. Данный метод загружает изображения из файла и делит их на кадры анимации, идентификаторы которых помещаются в массив int[] animationId. Таймер анимации обновляется в каждом кадре благодаря прибавлению к текущему количеству миллисекунд с начала анимации число, равное количеству миллисекунд, прошедшее с прошлого кадра. Впоследствии данный таймер используется для получения индекса изображения в массиве. Когда класс Renderer запрашивает идентификатор текстуры для отрисовки, анимация возвращает идентификатор текстуры, хранящейся в массиве текстур по индексу, характерному для данного момента времени. Такая логика позволила реализовать плавные анимации, независящие от частоты кадров приложения.

## 3.2 Результаты тестирования игрового приложения

В ходе разработки игрового приложения требовалось тестировать работоспособность созданных классов. С этой целью создается проект модульных тестов GameTests, в котором тестируются разработанные классы.

Проект GameTests содержит в себе классы HealthTests (Приложение А, код программ HealthTests.cs), InventoryTests (Приложение А, код программ InventoryTests.cs), PhysicTests (Приложение А, код программ PhysicTests.cs), RocketTests (Приложение А, код программ RocketTests.cs) и CollisionTests (Приложение А, код программ CollisionTests.cs) для модульного тестирования различных аспектов разработанного приложения.

Класс HealthTests реализует тесты здоровья танка. Метод HealthDamageTest проверяет, корректно ли начисляется урон, а метод HealthIsAliveTest проверяет, правильно ли определяется момент смерти танка.

Тесты инвентаря расположены в классе InventoryTests. Методы данного класса проверяют корректность добавления ракет в инвентарь и корректность изъятия ракет из инвентаря. Также данный класс проверяет возможность добавлять разные виды ракет в инвентарь.

Тестирование игровой физики реализовано посредством создания класса PhysicTests. Данный класс имеет методы для проверки свободного падения тела, проверки падения тела с начальной скоростью и проверки равноускоренного движения тел.

Класс RocketTests имеет методы для тестирования различных конфигураций ракет при помощи декорирования базовой ракеты. Данные методы комбинируют различные декорации ракеты, получая снаряды с разными значениями бронебойности и скорострельности. Также методы данного класса проверяют комбинации декораторов на правильность подсчетов урона от ракет и на валидность значения скорострельности при декорировании.

В классе CollisionTests сосредоточены модульные тесты системы столкновений, применяемой в игре. Созданные методы циклически тестируют на столкновение разнообразные геометрические фигуры. Данные тесты случайным образом перемещают фигуры по игровому пространству и проверяют, правильно ли они сталкиваются друг с другом.

На рисунке 3.4 представлены результаты модульного тестирования разработанных классов игрового приложения.

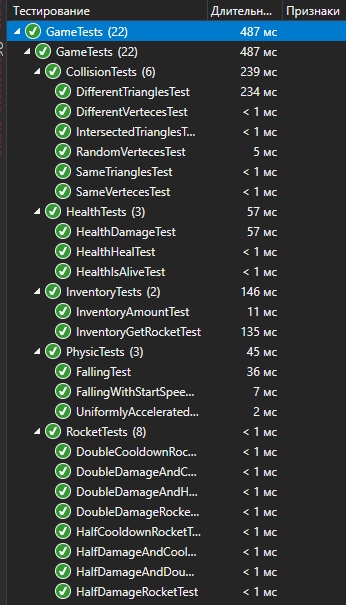


Рисунок 3.4 – Результаты тестирования приложения.

## 3.3 Результаты верификации игрового приложения

После запуска игры открывается окно главного меню с возможностью перейти на экран покупки боеприпасов артиллерийских установок. На экране покупки игроки выбирают себе боеприпасы для последующего использования в бою. После настройки сцены и конфигурации инвентаря танка начинается, игровой процесс. Игрокам необходимо управлять своими самоходными артиллерийскими установками с помощью клавиатуры, осуществлять выстрелы по сложным артиллерийским траекториям и уворачиваться от вражеских снарядов. Снаряды, выпускаемые игроками, летят по параболическим траекториям, которые имитируют физику падения реальных тел. При попадании снаряда у артиллерийской установки уменьшается запас здоровья. При уменьшении здоровья до нуля артиллерийская установка уничтожается и игрок, управляющий данной артиллерийской установкой проигрывает матч. Вначале оба игрока имеют полный запас здоровья и инвентаря. На рисунке 3.5 изображено начальное состояние игрового поля.

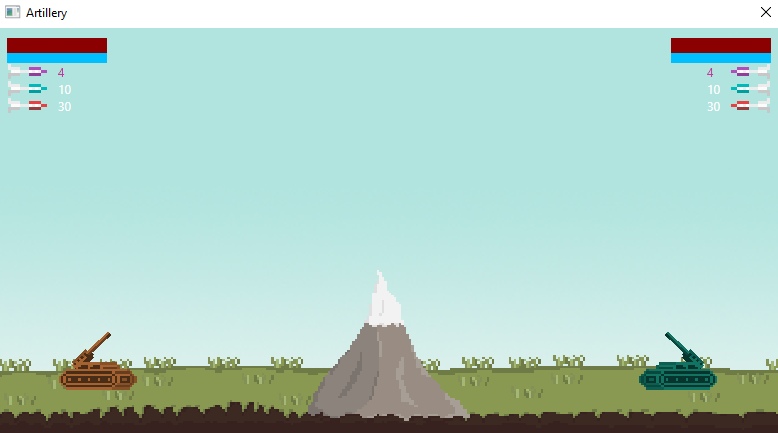


Рисунок 3.5 – Игровое поле.

В таблице 3.1 представлены клавиши для управления самоходными артиллерийскими установками.

Таблица 3.1 – Управление самоходными артиллерийскими установками

|  |  |  |
| --- | --- | --- |
| Действие | Игрок 1 | Игрок 2 |
| Передвижение влево/вправо | A/D | Left/Right |
| Поворот башни вверх/вниз | W/S | Up/Down |
| Смена ракеты | Q/E | -/+ |
| Стрельба | Space | Enter |

Цель каждого игрока – уничтожить танк противника. Для уничтожения врага у игроков есть запас ракет, которыми они могут наносить повреждения вражескому танку. При выстреле по врагу из общего боезапаса вычитается ракета, которой был произведен выстрел. Танки имеют в своем распоряжении три вида ракет с различной бронебойностью и скорострельностью, между которыми они могут переключаться в любой момент боя. Попадание ракетой по танку наносит ему повреждение, равное бронебойности снаряда. Оставшееся здоровье танка отображается в углу экрана. Здоровье танка до и после попадания изображено на рисунке 3.6.



Рисунок 3.6 – Здоровье танка до и после получения повреждения.

Как только один из игроков наносит другому критическое повреждение, он становится победителем, а артиллерийская установка его оппонента уничтожается. После победы появляется меню, позволяющее перезапустить игру с теми же параметрами и настройками инвентаря или вернуться в главное меню, откуда можно изменить параметры игры и заново закупить боеприпасы для самоходных артиллерийских установок. Также из главного меню можно включить искусственный интеллект для одной или обеих артиллерийских установок, который будет передвигаться по полю и производить выстрелы как настоящий игрок. Искусственный интеллект позволяет обойтись без одного из игроков в случае желания поиграть в игру в одиночестве. Сообщение об окончании игры представлено на рисунке 3.7.

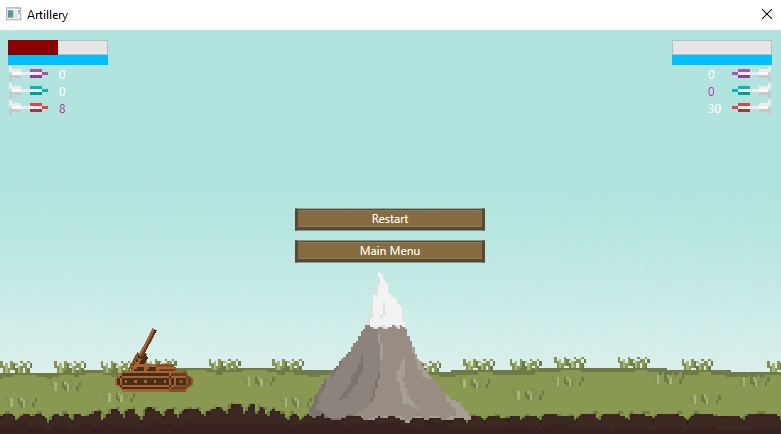


Рисунок 3.7 – Сообщение об окончании игры.

После окончания игры, победитель может продолжить перемещаться по полю и производить выстрелы оставшимися ракетами. Если игрок захочет выйти из игры, он может нажать на кнопку для перехода в главное меню, откуда у него появится возможность нажать соответствующую кнопку для выхода из игрового приложения.

# ЗАКЛЮЧЕНИЕ

Разработано игровое приложение «Перестрелка». Данное приложение является представителем жанра «Артиллерийская игра», что определяет основные требования к игровому процессу: управление траекторией полета снаряда и динамический геймплей. Созданное игровое приложение охватывает все возрастные категории пользователей, является простым в освоении и способствует развитию реакции у игроков.

Достоинством разработанной игры является разнообразие боеприпасов, которые вносят тактическую составляющую в игровой процесс. Также необычной особенностью является наличие возможности покупать боеприпасы перед игрой для использования в сражении с другим игроком.

По итогам анализа графических средств разработки, для реализации игрового приложения используется библиотека OpenTK, которая предоставляет доступ к графическим инструментам, содержащимся в OpenGL. Графика приложения отображается в компоненте окна WPF, который предоставляется при подключении библиотеки GLWpfControl. Данный подход позволил сосредоточиться на реализации игровой логики, так как окно WPF имеет ряд элементов, упрощающих взаимодействие пользователя с игровым приложением.

При выполнении работы структура разработанного игрового приложения определяется таким образом, чтобы впоследствии ее можно было легко расширить новым функционалом. Данный эффект достигается благодаря применению шаблонов проектирования, в частности структурного паттерна «Декоратор».

Курсовая работа выполнена самостоятельно, проверена в системе «Антиплагиат». Процент оригинальности составляет 99%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Рихтер, Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Дж. Рихтер. – СПб.:Питер, 2020. – 896 с.
2. Миллер, Т. Managed DirectX 9 с управляемым кодом. Программирование игр и графика / Т. Миллер. – М.:КомБук, 2005. – 401 с.
3. Флемов, М. Е. DirectX и C++. Искусство программирования /

М. Е. Флемов. – СПб.:БХВ-Петербург, 2017. – 384 с.

1. Селлерс, Г. Супербиблия OpenGL / Г. Селлерс. – Addison-Wesley Professional, 2015. – 880 с.
2. Вольф, Д. OpenGL 4. Язык Шейдеров. Книга рецептов / Д. Вольф. – М.:ДМК Пресс, 2015. – 368 с.
3. Боресков, А. В. Программирование компьютерной графики. Современный OpenGL / А. В. Боресков. – М.:ДМК Пресс, 2019. – 372 с.
4. Декомпозиция целей и задач [Электронный ресурс]. – 2020. – Режим доступа: https://time-blog.ru/dekompoziciya-celej-i-zadach. – Дата доступа: 20.04.2020
5. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма [и др.]. – СПб.:Питер, 2020. – 368 с.
6. Декоратор [Электронный ресурс]. – 2015. – Режим доступа: https://metanit.com/sharp/patterns/4.1.php. – Дата доступа: 07.03.2020.
7. Мак-Дональд, М. WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов / М. Мак-Дональд. – Вильямс, 2019. – 1024 с.

# ПРИЛОЖЕНИЕ А

(обязательное)

**Листинг программы «Перестрелка»**

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

using System;

using System.Windows;

using OpenTK.Graphics.OpenGL;

using OpenTK.Wpf;

using GameEngineLibrary;

using GameLibrary;

using GameLibrary.Scenes;

using System.Windows.Controls;

namespace GameUserInterface

{

/// <summary>

/// Логика взаимодействия для главного окна игры.

/// </summary>

public partial class MainWindow : Window

{

private BattleSceneSettings settings;

/// <summary>

/// Сцена, которая будет отрисовываться на экране.

/// </summary>

private Scene scene;

/// <summary>

/// Отрисовщик сцены.

/// </summary>

private Renderer renderer;

/// <summary>

/// Создание окна приложения.

/// </summary>

public MainWindow()

{

InitializeComponent();

InitStartScreen();

scene = new BattleScene(this, settings);

MainMenu.Visibility = Visibility.Visible;

FirstPanzerInfo.Visibility = Visibility.Hidden;

SecondPanzerInfo.Visibility = Visibility.Hidden;

RocketShop.Visibility = Visibility.Hidden;

WinMenu.Visibility = Visibility.Hidden;

var WpfControlSettings = new GLWpfControlSettings();

WpfControlSettings.MajorVersion = 3;

WpfControlSettings.MinorVersion = 6;

OpenTKControl.Start(WpfControlSettings);

}

private void OpenTKControl\_Render(TimeSpan delta)

{

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

GL.MatrixMode(MatrixMode.Projection);

GL.LoadIdentity();

GL.Ortho(-Width, Width, Height, -Height, 0d, 1d);

scene.Update(delta);

renderer.Render();

}

private void OpenTKControl\_Ready()

{

GL.Enable(EnableCap.Texture2D);

GL.Enable(EnableCap.Blend);

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

scene.Init();

renderer = new Renderer(scene);

}

private void Window\_Closing(object sender, System.ComponentModel.CancelEventArgs e)

{

scene.Dispose();

GL.Disable(EnableCap.Blend);

GL.Disable(EnableCap.Texture2D);

}

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

{

MainMenu.Visibility = Visibility.Hidden;

RocketShop.Visibility = Visibility.Visible;

}

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

{

Close();

}

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

{

RocketShop.Visibility = Visibility.Hidden;

FirstPanzerInfo.Visibility = Visibility.Visible;

SecondPanzerInfo.Visibility = Visibility.Visible;

scene.Dispose();

settings = new BattleSceneSettings();

int firstPanzerPowerfulRockets = int.Parse(FirstPanzerPowerfulRockets.Content.ToString());

int firstPanzerFastRockets = int.Parse(FirstPanzerFastRockets.Content.ToString());

int firstPanzerRockets = int.Parse(FirstPanzerRockets.Content.ToString());

int secondPanzerPowerfulRockets = int.Parse(SecondPanzerPowerfulRockets.Content.ToString());

int secondPanzerFastRockets = int.Parse(SecondPanzerFastRockets.Content.ToString());

int secondPanzerRockets = int.Parse(SecondPanzerRockets.Content.ToString());

settings.SetFirstPanzerAmounts(firstPanzerPowerfulRockets, firstPanzerFastRockets, firstPanzerRockets);

settings.SetSecondPanzerAmounts(secondPanzerPowerfulRockets, secondPanzerFastRockets, secondPanzerRockets);

settings.FirstPanzerHealth = 100;

settings.SecondPanzerHealth = 100;

settings.FirstPanzerControlType = (bool)IsFirstAI.IsChecked ?

BattleSceneSettings.PanzerControlType.AI :

BattleSceneSettings.PanzerControlType.Keyboard;

settings.SecondPanzerControlType = (bool)IsSecondAI.IsChecked ?

BattleSceneSettings.PanzerControlType.AI :

BattleSceneSettings.PanzerControlType.Keyboard;

scene = new BattleScene(this, settings);

scene.Init();

renderer = new Renderer(scene);

}

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

{

Button button = sender as Button;

Grid grid = button.Parent as Grid;

Label outputLabel = grid.Children[0] as Label;

Label moneyLable = (grid.Parent as Grid).Children[0] as Label;

int amount = int.Parse(button.Content.ToString());

int value = int.Parse(outputLabel.Content.ToString());

int money = int.Parse(moneyLable.Content.ToString());

int cost = int.Parse((grid.Children[1] as Label).Content.ToString()) \* amount;

if (value + amount >= 0 && money - cost >= 0)

{

value += amount;

money -= cost;

outputLabel.Content = value.ToString();

moneyLable.Content = money.ToString();

}

}

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

{

WinMenu.Visibility = Visibility.Hidden;

scene.Dispose();

scene = new BattleScene(this, settings);

scene.Init();

renderer = new Renderer(scene);

}

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

{

WinMenu.Visibility = Visibility.Hidden;

MainMenu.Visibility = Visibility.Visible;

FirstPanzerInfo.Visibility = Visibility.Hidden;

SecondPanzerInfo.Visibility = Visibility.Hidden;

scene.Dispose();

InitStartScreen();

scene = new BattleScene(this, settings);

scene.Init();

renderer = new Renderer(scene);

}

private void InitStartScreen()

{

settings = new BattleSceneSettings();

settings.SetFirstPanzerAmounts(int.MaxValue, int.MaxValue, int.MaxValue);

settings.SetSecondPanzerAmounts(int.MaxValue, int.MaxValue, int.MaxValue);

settings.FirstPanzerHealth = int.MaxValue;

settings.SecondPanzerHealth = int.MaxValue;

settings.FirstPanzerControlType = BattleSceneSettings.PanzerControlType.AI;

settings.SecondPanzerControlType = BattleSceneSettings.PanzerControlType.AI;

}

}

}

**Код программы для MainWindow.xaml:**

<Window Name="PanzerGameWindow" x:Class="GameUserInterface.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:GameUserInterface"

xmlns:glWpfControl="clr-namespace:OpenTK.Wpf;assembly=GLWpfControl"

mc:Ignorable="d"

Closing="Window\_Closing"

Title="Artillery" Height="450" Width="800" ResizeMode="NoResize">

<Window.Resources>

<Style TargetType="Button" x:Key="ButtonStyle">

<Style.Setters>

<Setter Property="Control.Foreground" Value="#fff7eb"/>

<Setter Property="Control.Background" Value="#876c42"/>

<Setter Property="Control.BorderBrush" Value="#594931"/>

<Setter Property="Control.BorderThickness" Value="3 2 3 4"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Border Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style.Setters>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Control.Foreground" Value="#f2ebe1"/>

<Setter Property="Control.Background" Value="#755e39"/>

<Setter Property="Control.BorderBrush" Value="#4a3d29"/>

</Trigger>

</Style.Triggers>

</Style>

<Style TargetType="Button" x:Key="StartButtonStyle">

<Style.Setters>

<Setter Property="Control.Foreground" Value="#fff7eb"/>

<Setter Property="Control.Background" Value="#b59535"/>

<Setter Property="Control.BorderBrush" Value="#70602e"/>

<Setter Property="Control.BorderThickness" Value="3 2 3 4"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Border Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style.Setters>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Control.Foreground" Value="#f2ebe1"/>

<Setter Property="Control.Background" Value="#9c802d"/>

<Setter Property="Control.BorderBrush" Value="#66572a"/>

</Trigger>

</Style.Triggers>

</Style>

<Style TargetType="Button" x:Key="BuyButtonStyle">

<Style.Setters>

<Setter Property="Control.Foreground" Value="#fff7eb"/>

<Setter Property="Control.Background" Value="#1d7a32"/>

<Setter Property="Control.BorderBrush" Value="#1a4a25"/>

<Setter Property="Control.BorderThickness" Value="3 2 3 4"/>

<Setter Property="Control.Margin" Value="1"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Border Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style.Setters>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Control.Foreground" Value="#f2ebe1"/>

<Setter Property="Control.Background" Value="#1b6b2d"/>

<Setter Property="Control.BorderBrush" Value="#163b1e"/>

</Trigger>

</Style.Triggers>

</Style>

<Style TargetType="Button" x:Key="SellButtonStyle">

<Style.Setters>

<Setter Property="Control.Foreground" Value="#fff7eb"/>

<Setter Property="Control.Background" Value="#a61f46"/>

<Setter Property="Control.BorderBrush" Value="#5c1f31"/>

<Setter Property="Control.BorderThickness" Value="3 2 3 4"/>

<Setter Property="Control.Margin" Value="1"/>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Border Background="{TemplateBinding Background}"

BorderBrush="{TemplateBinding BorderBrush}"

BorderThickness="{TemplateBinding BorderThickness}">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style.Setters>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Control.Foreground" Value="#f2ebe1"/>

<Setter Property="Control.Background" Value="#821736"/>

<Setter Property="Control.BorderBrush" Value="#4a1928"/>

</Trigger>

</Style.Triggers>

</Style>

</Window.Resources>

<Grid>

<glWpfControl:GLWpfControl

x:Name="OpenTKControl"

Render="OpenTKControl\_Render"

Ready="OpenTKControl\_Ready"/>

<StackPanel x:Name="FirstPanzerInfo"

Visibility="Hidden"

HorizontalAlignment="Left"

VerticalAlignment="Top"

Margin="10">

<ProgressBar

x:Name="FirstPanzerHealth"

Minimum="0"

Maximum="100"

Value="100"

Foreground="DarkRed"

Width="100"

Height="15"/>

<ProgressBar

x:Name="FirstPanzerCooldown"

Minimum="0"

Maximum="100"

Value="100"

Foreground="DeepSkyBlue"

Width="100"

Height="10"/>

<StackPanel

Orientation="Horizontal"

HorizontalAlignment="Left">

<StackPanel>

<Image Source="Resources\PowerfulRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft"

Margin="1"

Height="15px"/>

<Image Source="Resources\FastRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft"

Margin="1"

Height="15px"/>

<Image Source="Resources\Rocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft"

Margin="1"

Height="15px"/>

</StackPanel>

<StackPanel

x:Name="FirstPanzerInventory">

<Label Content="0" Height="15px" Padding="0" Margin="10 1 10 1"/>

<Label Content="0" Height="15px" Padding="0" Margin="10 1 10 1"/>

<Label Content="0" Height="15px" Padding="0" Margin="10 1 10 1"/>

</StackPanel>

</StackPanel>

</StackPanel>

<StackPanel x:Name="SecondPanzerInfo"

Visibility="Hidden"

HorizontalAlignment="Right"

VerticalAlignment="Top"

Margin="10">

<ProgressBar

x:Name="SecondPanzerHealth"

Minimum="0"

Maximum="100"

Value="100"

Foreground="DarkRed"

Width="100"

Height="15"/>

<ProgressBar

x:Name="SecondPanzerCooldown"

Minimum="0"

Maximum="100"

Value="100"

Foreground="DeepSkyBlue"

Width="100"

Height="10"/>

<StackPanel

Orientation="Horizontal"

HorizontalAlignment="Right">

<StackPanel

x:Name="SecondPanzerInventory">

<Label Content="0" Height="15px" Padding="0" Margin="10 1 10 1"/>

<Label Content="0" Height="15px" Padding="0" Margin="10 1 10 1"/>

<Label Content="0" Height="15px" Padding="0" Margin="10 1 10 1"/>

</StackPanel>

<StackPanel>

<Image Source="Resources\PowerfulRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

Margin="1"

Height="15px"/>

<Image Source="Resources\FastRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

Margin="1"

Height="15px"/>

<Image Source="Resources\Rocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

Margin="1"

Height="15px"/>

</StackPanel>

</StackPanel>

</StackPanel>

<StackPanel x:Name="MainMenu"

Visibility="Visible"

HorizontalAlignment="Center"

VerticalAlignment="Center"

Width="200px">

<Button

x:Name="PlayGameBtn"

Content="Play"

Margin="5"

Click="PlayGameBtn\_Click"

Style="{StaticResource ButtonStyle}"/>

<Button

x:Name="QuitGameBtn"

Content="Quit"

Margin="5"

Click="QuitGameBtn\_Click"

Style="{StaticResource ButtonStyle}"/>

</StackPanel>

<StackPanel x:Name="WinMenu"

Visibility="Hidden"

HorizontalAlignment="Center"

VerticalAlignment="Center"

Width="200px">

<Button

x:Name="RestartGameBtn"

Content="Restart"

Margin="5"

Click="RestartGameBtn\_Click"

Style="{StaticResource ButtonStyle}"/>

<Button

x:Name="MainMenuBtn"

Content="Main Menu"

Margin="5"

Click="MainMenuBtn\_Click"

Style="{StaticResource ButtonStyle}"/>

</StackPanel>

<Grid x:Name="RocketShop"

Visibility="Hidden"

HorizontalAlignment="Center"

VerticalAlignment="Center"

Width="600">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

<Grid Grid.Column="0"

Grid.Row="0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<CheckBox Content="AI"

x:Name="IsFirstAI"

Margin="5"

IsChecked="False"

HorizontalAlignment="Right"

VerticalAlignment="Center"

Grid.Column="1"/>

<Label Content="First Player"

Margin="5"

VerticalAlignment="Center"

Grid.Column="0"/>

</Grid>

<Grid Grid.Column="1"

Grid.Row="0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<CheckBox Content="AI"

x:Name="IsSecondAI"

Margin="5"

IsChecked="True"

HorizontalAlignment="Right"

VerticalAlignment="Center"

Grid.Column="1"/>

<Label Content="Second Player"

Margin="5"

VerticalAlignment="Center"

Grid.Column="0"/>

</Grid>

<Grid Grid.Column="0"

Grid.Row="1"

Margin="5">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="48px"/>

</Grid.RowDefinitions>

<Label Content="0" Grid.Row="0" Grid.Column="2" HorizontalAlignment="Right"/>

<Label Content="Money: " Grid.ColumnSpan="2" Grid.Row="0" Grid.Column="0"/>

<Image Source="Resources\PowerfulRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft" Margin="2"

Height="15px" Grid.Row="2" Grid.Column="0"/>

<Image Source="Resources\FastRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft" Margin="2"

Height="15px" Grid.Row="2" Grid.Column="1"/>

<Image Source="Resources\Rocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft" Margin="2"

Height="15px" Grid.Row="2" Grid.Column="2"/>

<Label Content="5"

HorizontalAlignment="Center"

Grid.Column="0"

Grid.Row="1"/>

<Label Content="4"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="1"/>

<Label Content="3"

HorizontalAlignment="Center"

Grid.Column="2"

Grid.Row="1"/>

<Grid

Grid.Row="3"

Grid.Column="0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="30px"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="24px"/>

<RowDefinition Height="24px"/>

</Grid.RowDefinitions>

<Label Name="FirstPanzerPowerfulRockets"

Content="4"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="2"/>

<Label Content="5" Visibility="Hidden"/>

<Button Content="-10" Grid.Column="0" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="-1" Grid.Column="0" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="+1" Grid.Column="2" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

<Button Content="+10" Grid.Column="2" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

</Grid>

<Grid

Grid.Row="3"

Grid.Column="1">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="30px"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="24px"/>

<RowDefinition Height="24px"/>

</Grid.RowDefinitions>

<Label Name="FirstPanzerFastRockets"

Content="10"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="2"/>

<Label Content="4" Visibility="Hidden"/>

<Button Content="-10" Grid.Column="0" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="-1" Grid.Column="0" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="+1" Grid.Column="2" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

<Button Content="+10" Grid.Column="2" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

</Grid>

<Grid

Grid.Row="3"

Grid.Column="2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="30px"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="24px"/>

<RowDefinition Height="24px"/>

</Grid.RowDefinitions>

<Label Name="FirstPanzerRockets"

Content="30"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="2"/>

<Label Content="3" Visibility="Hidden"/>

<Button Content="-10" Grid.Column="0" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="-1" Grid.Column="0" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="+1" Grid.Column="2" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

<Button Content="+10" Grid.Column="2" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

</Grid>

</Grid>

<Grid Grid.Column="1"

Grid.Row="1"

Margin="5">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="\*"/>

<RowDefinition Height="48px"/>

</Grid.RowDefinitions>

<Label Content="0" Grid.Row="0" Grid.Column="2" HorizontalAlignment="Right"/>

<Label Content="Money: " Grid.ColumnSpan="2" Grid.Row="0" Grid.Column="0"/>

<Image Source="Resources\PowerfulRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft" Margin="2"

Height="15px" Grid.Row="2" Grid.Column="0"/>

<Image Source="Resources\FastRocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft" Margin="2"

Height="15px" Grid.Row="2" Grid.Column="1"/>

<Image Source="Resources\Rocket.bmp"

RenderOptions.BitmapScalingMode="NearestNeighbor"

FlowDirection="RightToLeft" Margin="2"

Height="15px" Grid.Row="2" Grid.Column="2"/>

<Label Content="5"

HorizontalAlignment="Center"

Grid.Column="0"

Grid.Row="1"/>

<Label Content="4"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="1"/>

<Label Content="3"

HorizontalAlignment="Center"

Grid.Column="2"

Grid.Row="1"/>

<Grid

Grid.Row="3"

Grid.Column="0">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="30px"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="24px"/>

<RowDefinition Height="24px"/>

</Grid.RowDefinitions>

<Label Name="SecondPanzerPowerfulRockets"

Content="4"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="2"/>

<Label Content="5" Visibility="Hidden"/>

<Button Content="-10" Grid.Column="0" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="-1" Grid.Column="0" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="+1" Grid.Column="2" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

<Button Content="+10" Grid.Column="2" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

</Grid>

<Grid

Grid.Row="3"

Grid.Column="1">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="30px"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="24px"/>

<RowDefinition Height="24px"/>

</Grid.RowDefinitions>

<Label Name="SecondPanzerFastRockets"

Content="10"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="2"/>

<Label Content="4" Visibility="Hidden"/>

<Button Content="-10" Grid.Column="0" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="-1" Grid.Column="0" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="+1" Grid.Column="2" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

<Button Content="+10" Grid.Column="2" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

</Grid>

<Grid

Grid.Row="3"

Grid.Column="2">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="\*"/>

<ColumnDefinition Width="30px"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="24px"/>

<RowDefinition Height="24px"/>

</Grid.RowDefinitions>

<Label Name="SecondPanzerRockets"

Content="30"

VerticalAlignment="Center"

HorizontalAlignment="Center"

Grid.Column="1"

Grid.Row="0"

Grid.RowSpan="2"/>

<Label Content="3" Visibility="Hidden"/>

<Button Content="-10" Grid.Column="0" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="-1" Grid.Column="0" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource SellButtonStyle}"/>

<Button Content="+1" Grid.Column="2" Grid.Row="0" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

<Button Content="+10" Grid.Column="2" Grid.Row="1" Click="BuyBtn\_Click" Style="{StaticResource BuyButtonStyle}"/>

</Grid>

</Grid>

<Button x:Name="StartGameBtn"

Grid.ColumnSpan="2"

Content="Start"

Grid.Column="0"

Grid.Row="2"

Margin="5"

Width="200"

Click="StartGameBtn\_Click"

Style="{StaticResource StartButtonStyle}"/>

</Grid>

</Grid>

</Window>

**Код программы для Animation2D.cs:**

using OpenTK.Graphics.OpenGL;

using System;

using System.Drawing;

using System.Drawing.Imaging;

namespace GameEngineLibrary

{

/// <summary>

/// Класс, отвечающий за анимированные 2d изображения.

/// </summary>

public class Animation2D : Texture2D

{

/// <summary>

/// Массив идентификаторов текстур анимации.

/// </summary>

private int[] animationId;

/// <summary>

/// Индекс текущей текстуры.

/// </summary>

private int index;

/// <summary>

/// Время, прошедшее с начала анимации.

/// </summary>

private int currentTime;

/// <summary>

/// Уникальный идентификатор текстуры,

/// которая должна отображаться в текущий

/// момент анимации.

/// </summary>

public override int ID

{

get

{

return animationId[index];

}

}

/// <summary>

/// Время анимации.

/// </summary>

public int AnimationTime { get; set; }

/// <summary>

/// Создание анимации.

/// </summary>

/// <param name="animationId">Массив идентификаторов изображений в анимации.</param>

/// <param name="width">Размер одного изображения.</param>

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

public Animation2D(int[] animationId, int width, int height)

: base(animationId[0], width, height)

{

index = 0;

AnimationTime = 1;

currentTime = 0;

this.animationId = animationId;

}

/// <summary>

/// Конструктор копирования анимации.

/// Его стоит использовать для передачи анимаций объектам,

/// чтобы не захватывать лишние неуправляемые ресурсы.

/// </summary>

/// <param name="animation"></param>

public Animation2D(Animation2D animation)

: this(animation.animationId, animation.Width, animation.Height)

{

disposed = true;

AnimationTime = animation.AnimationTime;

}

/// <summary>

/// Обновить состояние анимации.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public void Update(TimeSpan delta)

{

currentTime += delta.Milliseconds;

if (currentTime >= AnimationTime)

{

currentTime = 0;

}

double deltaTime = (double)currentTime / AnimationTime;

index = (int)(animationId.Length \* deltaTime);

}

/// <summary>

/// Загрузить анимацию из файла.

/// </summary>

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

/// <param name="columns">Количество колонок со спрайтами.</param>

/// <param name="rows">Количество рядов со спрайтами.</param>

/// <returns>Новая анимация.</returns>

public static Animation2D LoadAnimation(string path, int columns = 1, int rows = 1)

{

int length = columns \* rows;

int[] idArr = new int[length];

Bitmap bitmap = new Bitmap(path);

int spriteWidth = bitmap.Width / columns;

int spriteHeight = bitmap.Height / rows;

Rectangle rectangle = new Rectangle(0, 0, spriteWidth, spriteHeight);

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

{

rectangle.X = spriteWidth \* (i % columns);

rectangle.Y = spriteHeight \*(i / columns);

int id = GL.GenTexture();

GL.BindTexture(TextureTarget.Texture2D, id);

Bitmap sprite = bitmap.Clone(rectangle,

System.Drawing.Imaging.PixelFormat.Format32bppArgb);

BitmapData data = sprite.LockBits(

new Rectangle(0, 0, sprite.Width, sprite.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);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureWrapS,

(int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureWrapT,

(int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureMinFilter,

(int)TextureMinFilter.Nearest);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureMagFilter,

(int)TextureMagFilter.Nearest);

sprite.UnlockBits(data);

idArr[i] = id;

}

return new Animation2D(idArr, spriteWidth, spriteHeight);

}

/// <summary>

/// Уничтожение анимации.

/// </summary>

public override void Dispose()

{

if (disposed)

return;

foreach (int id in animationId)

{

GL.DeleteTexture(id);

}

disposed = true;

}

}

}

**Код программы для Collider.cs:**

using OpenTK;

using OpenTK.Graphics.OpenGL;

namespace GameEngineLibrary

{

/// <summary>

/// Класс, описывающий выпуклую фигуру,

/// которая способна совершать проверки на

/// пересечение с другими выпуклыми фигурами.

/// </summary>

public class Collider : IComponent

{

/// <summary>

/// Массив вершин выпуклой фигры.

/// </summary>

private Vector2[] verteces;

/// <summary>

/// Создание коллайдера.

/// </summary>

/// <param name="verteces">Вершины выпуклой фигуры, расположенные против часовой стрелки.</param>

public Collider(params Vector2[] verteces)

{

this.verteces = verteces;

}

/// <summary>

/// Проверка пересечения между двумя коллайдерами.

/// </summary>

/// <param name="collider">Коллайдер, с которым будет осуществляться проверка.</param>

/// <returns>True, если есть пересечение, иначе false.</returns>

public bool CheckCollision(Collider collider)

{

int count = verteces.Length + collider.verteces.Length;

Vector2[] allVertices = new Vector2[count];

verteces.CopyTo(allVertices, 0);

collider.verteces.CopyTo(allVertices, verteces.Length);

Vector2 normal;

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

{

normal = GetNormal(allVertices, i);

Vector2 firstProjection = GetProjection(normal);

Vector2 secondProjection = collider.GetProjection(normal);

if (firstProjection.X < secondProjection.Y ||

secondProjection.X < firstProjection.Y)

{

return false;

}

}

return true;

}

/// <summary>

/// Получение нормального вектора к заданной стороне фигуры.

/// </summary>

/// <param name="verteces">Вершины фигуры.</param>

/// <param name="num">Номер стороны.</param>

/// <returns></returns>

private Vector2 GetNormal(Vector2[] verteces, int num)

{

int next = num + 1;

next = next == verteces.Length ? 0 : next;

Vector2 firstPoint = verteces[num];

Vector2 secondPoint = verteces[next];

Vector2 edge = new Vector2(

secondPoint.X - firstPoint.X,

secondPoint.Y - firstPoint.Y);

return new Vector2(-edge.Y, edge.X);

}

/// <summary>

/// Получение вектора проекции фигуры на плоскость.

/// </summary>

/// <param name="vector">Вектор плоскости.</param>

/// <returns>Вектор проекции.</returns>

private Vector2 GetProjection(Vector2 vector)

{

Vector2 result = new Vector2();

bool isNull = true;

foreach (Vector2 current in verteces)

{

float projection = vector.X \* current.X +

vector.Y \* current.Y;

if (isNull)

{

result = new Vector2(projection, projection);

isNull = false;

}

if (projection > result.X)

{

result.X = projection;

}

if (projection < result.Y)

{

result.Y = projection;

}

}

return result;

}

}

}

**Код программы для GameObject.cs:**

using System;

using OpenTK;

using System.Collections.Generic;

namespace GameEngineLibrary

{

/// <summary>

/// Класс, который описывает объект на сцене.

/// </summary>

public class GameObject : IDisposable

{

private bool disposed = false;

private Dictionary<string, IComponent> components;

/// <summary>

/// Список добавленных скриптов.

/// </summary>

private List<Script> scripts;

/// <summary>

/// Список внутренних объектов.

/// </summary>

public List<GameObject> InnerObjects { get; private set; }

/// <summary>

/// Создание нового игрового объекта.

/// </summary>

/// <param name="texture">Текстура объекта.</param>

/// <param name="position">Положение объекта на сцене.</param>

/// <param name="rotationPoint">Точка поворота объекта.</param>

/// <param name="scale">Масштаб объекта.</param>

/// <param name="rotation">Угол поворота объекта.</param>

public GameObject(Texture2D texture, Vector2 position,

Vector2 rotationPoint, Vector2 scale, double rotation)

{

InnerObjects = new List<GameObject>();

scripts = new List<Script>();

components = new Dictionary<string, IComponent>();

Transform transform = new Transform();

transform.Position = position;

transform.RotationPoint = rotationPoint;

transform.Scale = scale;

transform.Rotation = rotation;

components["transform"] = transform;

if (texture != null)

{

components["texture"] = texture;

}

}

/// <summary>

/// Создание нового игрового объекта.

/// </summary>

public GameObject() : this(null, Vector2.Zero, Vector2.Zero, Vector2.One, 0)

{

}

/// <summary>

/// Создание нового игрового объекта с текстурой.

/// </summary>

/// <param name="texture">Текстура объекта.</param>

public GameObject(Texture2D texture) : this()

{

components["texture"] = texture;

}

/// <summary>

/// Добавление объекта внутрь другого объекта.

/// Объекты внутри других объектов должны отрисовываться

/// относительно позиции родительского объекта.

/// </summary>

/// <param name="gameObject">Объект для вставки.</param>

public void AddInnerObject(GameObject gameObject)

{

InnerObjects.Add(gameObject);

Transform transform = gameObject.GetComponent("transform") as Transform;

if (transform != null)

{

transform.Parent = this;

}

}

/// <summary>

/// Удаление объекта из другого объекта.

/// Объекты внутри других объектов должны отрисовываться

/// относительно позиции родительского объекта.

/// </summary>

/// <param name="gameObject">Объект для удаления.</param>

public void RemoveInnerObject(GameObject gameObject)

{

InnerObjects.Remove(gameObject);

Transform transform = gameObject.GetComponent("transform") as Transform;

if (transform != null)

{

transform.Parent = null;

}

}

/// <summary>

/// Добавление скрипта для объекта.

/// Скрипт определяет основное поведение объекта.

/// </summary>

/// <param name="script">Скрипт для добавления.</param>

public void AddScript(Script script)

{

scripts.Add(script);

script.SetControlledObject(this);

script.Init();

}

/// <summary>

/// Удаление скрипта объекта.

/// Скрипт определяет основное поведение объекта.

/// </summary>

/// <param name="script">Скрипт для удаления.</param>

public void RemoveScript(Script script)

{

scripts.Remove(script);

script.SetControlledObject(null);

}

/// <summary>

/// Установить коллайдер объекту.

/// </summary>

/// <param name="collider">Новый коллайдер.</param>

public void SetCollider(Collider collider)

{

components["collider"] = collider;

}

/// <summary>

/// Привязать коллайдер к текстуре.

/// </summary>

public void UpdateColliderToTexture()

{

Transform transform = GetComponent("transform") as Transform;

Texture2D texture = GetComponent("texture") as Texture2D;

float x = transform.Position.X;

float y = transform.Position.Y;

float width = texture.Width \* transform.Scale.X;

float height = texture.Height \* transform.Scale.Y;

float rotationX = transform.RotationPoint.X \* transform.Scale.X;

float rotationY = transform.RotationPoint.Y \* transform.Scale.Y;

double angle = transform.Rotation \* Math.Sign(transform.Scale.X);

Vector2[] vertices = new Vector2[]

{

new Vector2(width, 0),

new Vector2(width, height),

new Vector2(0, height),

new Vector2(0, 0),

};

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

{

vertices[i].X -= rotationX;

vertices[i].Y -= rotationY;

vertices[i] = new Vector2(

(float)(Math.Cos(angle) \* vertices[i].X -

Math.Sin(angle) \* vertices[i].Y),

(float)(Math.Sin(angle) \* vertices[i].X +

Math.Cos(angle) \* vertices[i].Y));

vertices[i].X += rotationX;

vertices[i].Y += rotationY;

vertices[i].X += x;

vertices[i].Y += y;

}

SetCollider(new Collider(vertices));

}

/// <summary>

/// Обновить состояние объекта.

/// </summary>

/// <param name="delta">Время, прошедшее между кадрами.</param>

public void Update(TimeSpan delta)

{

foreach (Script script in scripts)

{

script.Update(delta);

}

foreach (GameObject gameObject in InnerObjects)

{

gameObject.Update(delta);

}

Animation2D animation = GetComponent("texture") as Animation2D;

if (animation != null)

{

animation.Update(delta);

}

}

/// <summary>

/// Получение компонентов объекта по названию.

/// </summary>

/// <param name="key">Название компонента.</param>

/// <returns></returns>

public IComponent GetComponent(string key)

{

IComponent component;

components.TryGetValue(key, out component);

return component;

}

/// <summary>

/// Добавление компонента объекту.

/// </summary>

/// <param name="key">Название компонента.</param>

/// <param name="component">Компонент для добавления.</param>

public void AddComponent(string key, IComponent component)

{

components[key] = component;

}

/// <summary>

/// Уничтожение игрового объекта.

/// </summary>

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

/// <summary>

/// Уничтожение игрового объекта.

/// </summary>

/// <param name="disposing">Если уже уничтожен.</param>

public void Dispose(bool disposing)

{

if (!disposed)

{

if (disposing)

{

if (GetComponent("texture") is Texture2D texture)

{

texture.Dispose();

}

}

disposed = true;

}

}

/// <summary>

/// Деструктор игрового объекта.

/// </summary>

~GameObject()

{

Dispose(false);

}

}

}

**Код программы для IComponent.cs:**

namespace GameEngineLibrary

{

/// <summary>

/// Маркерный интерфейс, который помечает все компоненты в GameObject.

/// </summary>

public interface IComponent

{

}

}

**Код программы для RandomManager.cs:**

using System;

namespace GameEngineLibrary

{

/// <summary>

/// Класс, отвечающий за случайные числа в игре.

/// </summary>

public static class RandomManager

{

private static Random random = new Random();

/// <summary>

/// Возвращает неотрицательное случайное целое число.

/// </summary>

/// <returns>Неотрицательное случайное целое число</returns>

public static int Next()

{

return random.Next();

}

/// <summary>

/// Возвращает неотрицательное случайное целое число,

/// которое меньше определенного максимального значения.

/// </summary>

/// <param name="maxValue">Максимальное значение.</param>

/// <returns>Неотрицательное случайное целое число</returns>

public static int Next(int maxValue)

{

return random.Next(maxValue);

}

/// <summary>

/// Возвращает неотрицательное случайное целое число

/// в указанном диапазоне.

/// </summary>

/// <param name="minValue">Минимальное значение.</param>

/// <param name="maxValue">Максимальное значение.</param>

/// <returns>Неотрицательное случайное целое число</returns>

public static int Next(int minValue, int maxValue)

{

return random.Next(minValue, maxValue);

}

/// <summary>

/// Возвращает случайное число с плавающей точкой,

/// которое больше или равно 0,0 и меньше 1,0.

/// </summary>

/// <returns></returns>

public static double NextDouble()

{

return random.NextDouble();

}

}

}

**Код программы для Renderer.cs:**

using System;

using System.Collections.Generic;

using OpenTK;

using OpenTK.Graphics.OpenGL;

namespace GameEngineLibrary

{

/// <summary>

/// Класс, отвечающий за отрисовку объектов на сцене.

/// </summary>

public class Renderer

{

/// <summary>

/// Сцена для отрисовки.

/// </summary>

private Scene scene;

/// <summary>

/// Создание нового объекта для отрисовки объектов на сцене.

/// </summary>

/// <param name="scene">Сцена, которую будет рендерить объект.</param>

public Renderer(Scene scene)

{

this.scene = scene;

}

/// <summary>

/// Отрисовать объекты на сцене.

/// </summary>

public void Render()

{

Render(scene.GetGameObjects());

}

/// <summary>

/// Отрисовать объекты из переданного массива.

/// </summary>

/// <param name="objectsToRender">Массив объектов для отрисовки.</param>

private void Render(List<GameObject> objectsToRender)

{

foreach (GameObject gameObject in objectsToRender)

{

Render(gameObject.InnerObjects);

RenderObject(gameObject);

}

}

/// <summary>

/// Отрисовать объект на сцене.

/// </summary>

/// <param name="gameObject">Объект для отрисовки.</param>

private void RenderObject(GameObject gameObject)

{

Texture2D texture = gameObject.GetComponent("texture") as Texture2D;

Transform transform = gameObject.GetComponent("transform") as Transform;

if (texture == null || transform == null)

{

return;

}

Vector2 rotationPoint = transform.RotationPoint;

Vector2 position = transform.Position;

double rotation = transform.Rotation;

Vector2[] vertices = new Vector2[4]

{

new Vector2(0, 0),

new Vector2(1, 0),

new Vector2(1, 1),

new Vector2(0, 1)

};

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

GL.Begin(PrimitiveType.Quads);

GL.Color3(texture.Color);

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

{

GL.TexCoord2(vertices[i]);

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

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

vertices[i].X -= rotationPoint.X;

vertices[i].Y -= rotationPoint.Y;

vertices[i] = new Vector2(

(float)(Math.Cos(rotation) \* vertices[i].X -

Math.Sin(rotation) \* vertices[i].Y),

(float)(Math.Sin(rotation) \* vertices[i].X +

Math.Cos(rotation) \* vertices[i].Y));

vertices[i].X += rotationPoint.X;

vertices[i].Y += rotationPoint.Y;

vertices[i] \*= transform.Scale;

vertices[i] += position;

GL.Vertex2(vertices[i]);

}

GL.End();

}

/// <summary>

/// Устанавливает сцену для отрисовки.

/// </summary>

/// <param name="scene">Новая сцена для отрисовки.</param>

public void SetSceneToRender(Scene scene)

{

this.scene = scene;

}

}

}

**Код программы для Scene.cs:**

using System;

using System.Collections.Generic;

using System.Windows;

namespace GameEngineLibrary

{

/// <summary>

/// Интерфейс, описывающий пользовательскую сцену.

/// </summary>

public abstract class Scene : IDisposable

{

/// <summary>

/// Окно, в котором отрисовывается сцена.

/// </summary>

public Window GameWindow { get; }

/// <summary>

/// Список созданных текстур.

/// </summary>

private List<Texture2D> textures;

/// <summary>

/// Список объектов на сцене.

/// </summary>

private List<GameObject> objects;

/// <summary>

/// Временный массив с объектами, который нужен

/// для возможности добавления объектов на сцену

/// во время выполнения скриптов.

/// </summary>

private List<GameObject> objectsToAdd;

/// <summary>

/// Временный массив с объектами, который нужен

/// для возможности удаления объектов со сцены

/// во время выполнения скриптов.

/// </summary>

private List<GameObject> objectsToRemove;

/// <summary>

/// Создание сцены.

/// </summary>

/// <param name="window">Окно, в котором отрисовывается сцена.</param>

public Scene(Window window)

{

GameWindow = window;

textures = new List<Texture2D>();

objects = new List<GameObject>();

objectsToRemove = new List<GameObject>();

objectsToAdd = new List<GameObject>();

}

/// <summary>

/// Метод, который вызывается 1 раз для инициализации сцены.

/// </summary>

public abstract void Init();

/// <summary>

/// Метод, который вызывается перед отрисовкой кадра.

/// В этом методе параметры сцены должны обновляться.

/// </summary>

/// <param name="delta">Время, прошедшее между кадрами.</param>

public void Update(TimeSpan delta)

{

foreach (GameObject gameObject in objects)

{

gameObject.Update(delta);

}

UpdateObjectsArray();

}

/// <summary>

/// Метод, позволяющий обновлять количество объектов для обработки динамически.

/// </summary>

private void UpdateObjectsArray()

{

if (objectsToRemove.Count != 0)

{

foreach (GameObject gameObject in objectsToRemove)

{

objects.Remove(gameObject);

}

objectsToRemove.Clear();

}

if (objectsToAdd.Count != 0)

{

objects.AddRange(objectsToAdd);

objectsToAdd.Clear();

}

}

/// <summary>

/// Получение списка с объектами на сцене.

/// </summary>

/// <returns>Список объектов на сцене.</returns>

public List<GameObject> GetGameObjects()

{

return objects;

}

/// <summary>

/// Метод добавления объекта на сцену.

/// </summary>

/// <param name="gameObject">Объект для добавления.</param>

public void AddGameObject(GameObject gameObject)

{

objectsToAdd.Add(gameObject);

}

/// <summary>

/// Метод удаления объекта со сцены.

/// </summary>

/// <param name="gameObject">Объект для удаления.</param>

public void RemoveGameObject(GameObject gameObject)

{

objectsToRemove.Add(gameObject);

}

/// <summary>

/// Метод добавления текстур на сцену.

/// </summary>

/// <param name="texture">Текстура для добавления.</param>

public void AddTexture(Texture2D texture)

{

textures.Add(texture);

}

/// <summary>

/// Уничтожение сцены.

/// </summary>

public void Dispose()

{

foreach (Texture2D texture in textures)

{

texture.Dispose();

}

foreach (GameObject gameObject in objects)

{

gameObject.Dispose();

}

}

}

}

**Код программы для Script.cs:**

using System;

namespace GameEngineLibrary

{

/// <summary>

/// Интерфейс, описывающий скрипт, который определяет поведение игрового объекта.

/// </summary>

public abstract class Script

{

/// <summary>

/// Объект, которым управляет скрипт.

/// </summary>

protected GameObject controlledObject;

/// <summary>

/// Метод, инициализирующий данные скрипта.

/// </summary>

public virtual void Init()

{

}

/// <summary>

/// Метод, который содержит основную логику программы.

/// Данный метод вызывается в каждом кадре игры.

/// </summary>

/// <param name="delta">Время, прошедшее между кадрами.</param>

public abstract void Update(TimeSpan delta);

/// <summary>

/// Устанавливает объект, который будет контролировать скрипт.

/// </summary>

/// <param name="controlledObject">Объект, контролируемый скриптом.</param>

public void SetControlledObject(GameObject controlledObject)

{

this.controlledObject = controlledObject;

}

}

}

**Код программы для Texture2D.cs :**

using OpenTK.Graphics.OpenGL;

using System;

using System.Drawing;

using System.Drawing.Imaging;

namespace GameEngineLibrary

{

/// <summary>

/// Класс, который описывает двухмерную текстуру.

/// </summary>

public class Texture2D : IComponent, IDisposable

{

/// <summary>

/// Переменная, отвечающая за состояние объекта.

/// </summary>

protected bool disposed = false;

/// <summary>

/// Идентификатор текстуры.

/// </summary>

public virtual int ID { get; private set; }

/// <summary>

/// Ширина текстуры.

/// </summary>

public int Width { get; private set; }

/// <summary>

/// Высота текстуры.

/// </summary>

public int Height { get; private set; }

/// <summary>

/// Цвет, который будет наложен на текстуру.

/// </summary>

public Color Color { get; set; }

/// <summary>

/// Создание новой текстуры.

/// </summary>

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

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

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

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

{

ID = id;

Width = width;

Height = height;

Color = Color.Transparent;

}

/// <summary>

/// Загрузка текстуры из файла.

/// </summary>

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

/// <returns>Загруженная текстура.</returns>

public static Texture2D LoadTexture(string path)

{

int id = GL.GenTexture();

GL.BindTexture(TextureTarget.Texture2D, id);

Bitmap bitmap = new Bitmap(path);

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.TextureWrapS,

(int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureWrapT,

(int)TextureWrapMode.Clamp);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureMinFilter,

(int)TextureMinFilter.Nearest);

GL.TexParameter(TextureTarget.Texture2D,

TextureParameterName.TextureMagFilter,

(int)TextureMagFilter.Nearest);

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

}

/// <summary>

/// Сравнение двух объектов.

/// </summary>

/// <param name="obj">Объект для сравнения.</param>

/// <returns>True, если объекты равны и false в противном случае.</returns>

public override bool Equals(object obj)

{

return obj is Texture2D texture &&

ID == texture.ID &&

Width == texture.Width &&

Height == texture.Height;

}

/// <summary>

/// Получение hash-кода объекта.

/// </summary>

/// <returns>Целое число.</returns>

public override int GetHashCode()

{

var hashCode = 1463928665;

hashCode = hashCode \* -1521134295 + ID.GetHashCode();

hashCode = hashCode \* -1521134295 + Width.GetHashCode();

hashCode = hashCode \* -1521134295 + Height.GetHashCode();

return hashCode;

}

/// <summary>

/// Уничтожение текстуры.

/// </summary>

public virtual void Dispose()

{

if (disposed)

return;

GL.DeleteTexture(ID);

disposed = true;

}

}

}

**Код программы для Transform.cs :**

using OpenTK;

namespace GameEngineLibrary

{

/// <summary>

/// Объект, отвечающий за расположение объекта на экране.

/// </summary>

public class Transform : IComponent

{

/// <summary>

/// Объект-родитель для объекта, которому принадлежит данный экземпляр Transform.

/// </summary>

public GameObject Parent { get; set; }

/// <summary>

/// Точка, вокруг которой будет поворачиваться объект.

/// </summary>

public Vector2 RotationPoint { get; set; }

/// <summary>

/// Локальные координаты объекта относительно родительского объекта.

/// </summary>

private Vector2 localPosition;

/// <summary>

/// Позиция объекта.

/// </summary>

public Vector2 Position

{

get

{

if (Parent == null)

{

return localPosition;

}

Transform transform = Parent.GetComponent("transform") as Transform;

if (transform != null)

{

return transform.Position + localPosition;

}

return localPosition;

}

set

{

localPosition = value;

}

}

/// <summary>

/// Масштабирование объект.

/// </summary>

public Vector2 Scale { get; set; }

/// <summary>

/// Угол поворота объекта.

/// </summary>

public double Rotation { get; set; }

}

}

**Код программы для ProgressBarHealth.cs:**

using System.Windows.Controls;

namespace GameLibrary.Components.HealthDecorators

{

/// <summary>

/// Декоратор, который привязывает ProgressBar к здоровью

/// </summary>

public class ProgressBarHealth : Health

{

/// <summary>

/// Декорируемый объект.

/// </summary>

private Health health;

/// <summary>

/// Привязанный ProgressBar.

/// </summary>

private ProgressBar bar;

/// <summary>

/// Создание декоратора, привязывающего здоровье к ProgressBar.

/// </summary>

/// <param name="health">Декорируемый объект.</param>

/// <param name="bar">ProgressBar для привязки.</param>

public ProgressBarHealth(Health health, ProgressBar bar)

{

this.health = health;

this.bar = bar;

bar.Value = health.Value;

}

/// <summary>

/// Нанесение повреждения объекту.

/// </summary>

/// <param name="damage">Количество повреждений, нанесенных объекту.</param>

public override void Damage(int damage)

{

health.Damage(damage);

bar.Value = health.Value;

}

/// <summary>

/// Увеличение здоровья объекта на заданное число.

/// </summary>

/// <param name="health">Количество исцеленного здоровья.</param>

public override void Heal(int health)

{

this.health.Heal(health);

bar.Value = this.health.Value;

}

/// <summary>

/// Проверка, жив ли объект.

/// </summary>

/// <returns>True, если объект ещё жив.</returns>

public override bool IsAlive()

{

return health.IsAlive();

}

}

}

**Код программы для DoubleCooldownRocket.cs:**

namespace GameLibrary.Components.RocketDecorators

{

/// <summary>

/// Декоратор компонента Rocket,

/// который увеличивает время перезарядки в 2 раза.

/// </summary>

public class DoubleCooldownRocket : RocketDecorator

{

/// <summary>

/// Перезарядка, увеличенный в 2 раза.

/// </summary>

public override int Cooldown

{

get

{

return rocket.Cooldown \* 2;

}

}

/// <summary>

/// Создание нового декоратора, который увеичивает время перезарядки в 2 раза.

/// </summary>

/// <param name="rocket">Декорируемый экземпляр.</param>

public DoubleCooldownRocket(Rocket rocket) : base(rocket)

{

}

}

}

**Код программы для DoubleDamageRocket.cs:**

namespace GameLibrary.Components.RocketDecorators

{

/// <summary>

/// Декоратор компонента Rocket,

/// который увеличивает урон от ракеты в 2 раза.

/// </summary>

public class DoubleDamageRocket : RocketDecorator

{

/// <summary>

/// Урон, увеличенный в 2 раза.

/// </summary>

public override int Damage

{

get

{

return rocket.Damage \* 2;

}

}

/// <summary>

/// Создание нового декоратора, который увеичивает урон в 2 раза.

/// </summary>

/// <param name="rocket">Декорируемый экземпляр.</param>

public DoubleDamageRocket(Rocket rocket) : base(rocket)

{

}

}

}

**Код программы для HalfCooldownRocket.cs:**

namespace GameLibrary.Components.RocketDecorators

{

/// <summary>

/// Декоратор компонента Rocket,

/// который уменьшает время перезарядки в 2 раза.

/// </summary>

public class HalfCooldownRocket : RocketDecorator

{

/// <summary>

/// Перезарядка, уменьшенная в 2 раза.

/// </summary>

public override int Cooldown

{

get

{

return rocket.Cooldown / 2;

}

}

/// <summary>

/// Создание нового декоратора, который уменьшает время перезарядки в 2 раза.

/// </summary>

/// <param name="rocket">Декорируемый экземпляр.</param>

public HalfCooldownRocket(Rocket rocket) : base(rocket)

{

}

}

}

**Код программы для HalfDamageRocket.cs:**

namespace GameLibrary.Components.RocketDecorators

{

/// <summary>

/// Декоратор компонента Rocket,

/// который уменьшает урон от ракеты в 2 раза.

/// </summary>

public class HalfDamageRocket : RocketDecorator

{

/// <summary>

/// Урон, уменьшенный в 2 раза.

/// </summary>

public override int Damage

{

get

{

return rocket.Damage / 2;

}

}

/// <summary>

/// Создание нового декоратора, который уменьшает урон в 2 раза.

/// </summary>

/// <param name="rocket">Декорируемый экземпляр.</param>

public HalfDamageRocket(Rocket rocket) : base(rocket)

{

}

}

}

**Код программы для BaseRocket.cs:**

namespace GameLibrary.Components

{

/// <summary>

/// Компонент, описывающий ракету со стандартными параметрами.

/// Ракета наносит 20 урона и имеет скорострельность 500 мс.

/// </summary>

public class BaseRocket : Rocket

{

private int damage;

private int cooldown;

/// <summary>

/// Урон ракеты.

/// </summary>

public override int Damage => damage;

/// <summary>

/// Время до следующего выстрела ракетой.

/// </summary>

public override int Cooldown => cooldown;

/// <summary>

/// Создание новой ракеты.

/// </summary>

public BaseRocket()

{

damage = 20;

cooldown = 500;

}

}

}

**Код программы для Health.cs:**

using GameEngineLibrary;

namespace GameLibrary.Components

{

/// <summary>

/// Компонент, который отвечает за здоровье у объекта.

/// </summary>

public class Health : IComponent

{

/// <summary>

/// Количество здоровья у объекта.

/// </summary>

public int Value { get; private set; }

/// <summary>

/// Создание компонента здоровья.

/// </summary>

public Health()

{

Value = 100;

}

/// <summary>

/// Создание компонента здоровья.

/// </summary>

/// <param name="health">Начальное количество здоровья.</param>

public Health(int health)

{

Value = health;

}

/// <summary>

/// Нанесение повреждения объекту.

/// </summary>

/// <param name="damage">Количество повреждений, нанесенных объекту.</param>

public virtual void Damage(int damage)

{

Value -= damage;

}

/// <summary>

/// Увеличение здоровья объекта на заданное число.

/// </summary>

/// <param name="health">Количество исцеленного здоровья.</param>

public virtual void Heal(int health)

{

this.Value += health;

}

/// <summary>

/// Проверка, жив ли объект.

/// </summary>

/// <returns>True, если объект ещё жив.</returns>

public virtual bool IsAlive()

{

return Value > 0;

}

}

}

**Код программы для Inventory.cs:**

using GameEngineLibrary;

using GameLibrary.Scripts;

using OpenTK;

namespace GameLibrary.Components

{

/// <summary>

/// Инвентарь, отвечающий за выдачу новых ракет,

/// а также за слежку за количеством оставшихся ракет.

/// </summary>

public class Inventory : IComponent

{

/// <summary>

/// Сборщики ракет, которые предназначены для создания новых ракет.

/// </summary>

private RocketBuilder[] rockets;

/// <summary>

/// Общее количество ракет в инвентаре.

/// </summary>

protected int totalAmount;

/// <summary>

/// Количества рокет в инвентаре.

/// </summary>

protected int[] amounts;

/// <summary>

/// Текущий индекс ракеты для создания.

/// </summary>

protected int current;

/// <summary>

/// Создание нового инвентаря.

/// </summary>

/// <param name="rockets">Сборщики ракет.</param>

public Inventory(params RocketBuilder[] rockets)

{

this.rockets = rockets;

amounts = new int[rockets.Length];

current = 0;

totalAmount = 0;

for (int i = 0; i < amounts.Length; i++)

{

amounts[i] = 0;

}

}

/// <summary>

/// Выбрать следующую ракету.

/// </summary>

public virtual void SelectNext()

{

current = (current < rockets.Length - 1) ? current + 1 : 0;

}

/// <summary>

/// Выбрать предыдущую ракету.

/// </summary>

public virtual void SelectPrevious()

{

current = (current > 0 ? current : rockets.Length) - 1;

}

/// <summary>

/// Установить количество выбранных ракет.

/// </summary>

/// <param name="amount">Новое количество выбранных ракет.</param>

public virtual void SetAmount(int amount)

{

totalAmount -= amounts[current];

amounts[current] = amount;

totalAmount += amounts[current];

}

/// <summary>

/// Получение количества текущих ракет.

/// </summary>

/// <returns>Количество текущих ракет.</returns>

public virtual int GetAmount()

{

return amounts[current];

}

/// <summary>

/// Получить выбранную ракету.

/// </summary>

/// <returns>Ракета из инвентаря.</returns>

public virtual GameObject GetRocket()

{

if (amounts[current] > 0)

{

amounts[current]--;

totalAmount--;

return rockets[current].Build();

}

return null;

}

/// <summary>

/// Получение общего количества ракет в инвентаре.

/// </summary>

/// <returns>Общее количество ракет в инвентаре.</returns>

public virtual int GetTotalAmount()

{

return totalAmount;

}

/// <summary>

/// Сборщик ракет.

/// </summary>

public class RocketBuilder

{

/// <summary>

/// Сцена, в которой будут создаваться ракеты.

/// </summary>

private Scene scene;

/// <summary>

/// Компонент создаваемой ракеты.

/// </summary>

private Rocket rocket;

/// <summary>

/// Текстура создаваемой ракеты.

/// </summary>

private Texture2D rocketTex;

/// <summary>

/// Анимация взрыва создаваемой ракеты.

/// </summary>

private Animation2D explosionAnim;

/// <summary>

/// Создание объекта, отвечающего за сборку ракет по заданным правилам.

/// </summary>

/// <param name="scene">Сцена, в которой будут создаваться ракеты.</param>

/// <param name="rocketTex">Текстура создаваемой ракеты.</param>

/// <param name="explosionAnim">Анимация взрыва создаваемой ракеты.</param>

/// <param name="rocket">Компонент создаваемой ракеты.</param>

public RocketBuilder(Scene scene, Texture2D rocketTex, Animation2D explosionAnim, Rocket rocket)

{

this.scene = scene;

this.rocketTex = rocketTex;

this.explosionAnim = explosionAnim;

this.rocket = rocket;

}

/// <summary>

/// Создание ракеты.

/// </summary>

/// <returns>Созданная ракета.</returns>

public GameObject Build()

{

GameObject rocket = new GameObject(rocketTex, Vector2.Zero,

new Vector2(rocketTex.Width / 2, rocketTex.Height / 2),

Vector2.One, 0);

rocket.AddScript(new RocketHitScript(scene, explosionAnim));

rocket.AddComponent("rocket", this.rocket);

return rocket;

}

}

}

}

**Код программы для Rocket.cs:**

using GameEngineLibrary;

namespace GameLibrary.Components

{

/// <summary>

/// Абстрактный класс, описывающий ракеты в игре.

/// </summary>

public abstract class Rocket : IComponent

{

/// <summary>

/// Урон ракеты.

/// </summary>

public abstract int Damage { get; }

/// <summary>

/// Время до следующего выстрела ракетой.

/// </summary>

public abstract int Cooldown { get; }

}

}

**Код программы для RocketDecorator.cs:**

namespace GameLibrary.Components

{

/// <summary>

/// Абстрактный класс, описывающий декораторы ракеты.

/// </summary>

public class RocketDecorator : Rocket

{

/// <summary>

/// Декорируемая ракета.

/// </summary>

protected Rocket rocket;

/// <summary>

/// Урон ракеты.

/// </summary>

public override int Damage => rocket.Damage;

/// <summary>

/// Время до следующего выстрела ракетой.

/// </summary>

public override int Cooldown => rocket.Cooldown;

/// <summary>

/// Создание декоратора ракеты.

/// </summary>

/// <param name="rocket">Декорируемая ракеты.</param>

public RocketDecorator(Rocket rocket)

{

this.rocket = rocket;

}

}

}

**Код программы для WpfInventory.cs:**

using System.Windows.Controls;

using System.Windows.Media;

using GameEngineLibrary;

namespace GameLibrary.Components

{

/// <summary>

/// Инвентарь, который способен отображать информацию о

/// содержимом в специальном поле приложения WPF.

/// </summary>

public class WpfInventory : Inventory

{

/// <summary>

/// Поле, в котором будет отобрааться информация о содержимом инвентаря.

/// </summary>

private StackPanel inventory;

private Brush unselected;

private Brush selected;

/// <summary>

/// Создание нового WPF инвентаря.

/// </summary>

/// <param name="inventory">Поле, в котором будет отображаться содержимое.</param>

/// <param name="rockets">Ракеты в инвентаре.</param>

public WpfInventory(StackPanel inventory, params RocketBuilder[] rockets)

: base(rockets)

{

this.inventory = inventory;

unselected = new SolidColorBrush(Color.FromRgb(255, 255, 255));

selected = new SolidColorBrush(Color.FromRgb(191, 55, 160));

}

/// <summary>

/// Выбор следующего предмета в инвентаре.

/// </summary>

public override void SelectNext()

{

((Label)inventory.Children[current]).Foreground = unselected;

base.SelectNext();

((Label)inventory.Children[current]).Foreground = selected;

}

/// <summary>

/// Выбор предыдущего предмета в инвентаре.

/// </summary>

public override void SelectPrevious()

{

((Label)inventory.Children[current]).Foreground = unselected;

base.SelectPrevious();

((Label)inventory.Children[current]).Foreground = selected;

}

/// <summary>

/// Установка количества предмета в инвентаре.

/// </summary>

public override void SetAmount(int amount)

{

base.SetAmount(amount);

((Label)inventory.Children[current]).Content = amount;

}

/// <summary>

/// Получение предмета из инвентаря.

/// </summary>

public override GameObject GetRocket()

{

GameObject rocket = base.GetRocket();

((Label)inventory.Children[current]).Content = amounts[current];

return rocket;

}

}

}

**Код программы для BattleScene.cs:**

using System.Drawing;

using System.Windows;

using System.Windows.Controls;

using GameEngineLibrary;

using GameLibrary.Components;

using GameLibrary.Components.HealthDecorators;

using GameLibrary.Components.RocketDecorators;

using GameLibrary.Scenes;

using GameLibrary.Scripts;

using OpenTK;

namespace GameLibrary

{

/// <summary>

/// Сцена танкового сражения в игре.

/// </summary>

public class BattleScene : Scene

{

private const string TRACK\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/Track.bmp";

private const string TURRET\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/Turret.bmp";

private const string BACKGROUND\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/BG.bmp";

private const string ROCKET\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/Rocket.bmp";

private const string POWERFULROCKET\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/PowerfulRocket.bmp";

private const string FASTROCKET\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/FastRocket.bmp";

private const string MOUNTAIN\_TEXTURE\_PATH = @"../../../GameLibrary/Resources/Mountain.bmp";

private const string EXPLOSION\_ANIMATION\_PATH = @"../../../GameLibrary/Resources/Explosion.bmp";

private BattleSceneSettings settings;

private GameObject firstPanzer;

private GameObject secondPanzer;

/// <summary>

/// Создание сцены.

/// </summary>

/// <param name="window">Окно, в котором будет отрисовываться сцена.</param>

/// <param name="settings">Настройки игровой сцены.</param>

public BattleScene(Window window, BattleSceneSettings settings)

: base (window)

{

this.settings = settings;

firstPanzer = new GameObject();

secondPanzer = new GameObject();

}

/// <summary>

/// Инициализация сцены.

/// </summary>

public override void Init()

{

AddGameObject(CreateBackground());

AddGameObject(CreateMountain());

Inventory.RocketBuilder[] rockets = CreateRockets();

Inventory firstPanzerInventory = new WpfInventory(

(StackPanel) GameWindow.FindName("FirstPanzerInventory"), rockets);

Inventory secondPanzerInventory = new WpfInventory(

(StackPanel)GameWindow.FindName("SecondPanzerInventory"), rockets);

settings.FillFirtsPanzerInventory(firstPanzerInventory);

settings.FillSecondPanzerInventory(secondPanzerInventory);

BuildFirstPanzer(firstPanzerInventory);

BuildSecondPanzer(secondPanzerInventory);

GameObject winChecker = new GameObject();

winChecker.AddScript(new WinCheckerScript(firstPanzer, secondPanzer,

GameWindow.FindName("WinMenu") as StackPanel));

AddGameObject(firstPanzer);

AddGameObject(secondPanzer);

AddGameObject(winChecker);

}

private void BuildPanzer(GameObject panzer, Texture2D trackTex,

Texture2D turretTex, Vector2 position, Vector2 scale,

int health, ProgressBar healthBar, Inventory inventory,

Script[] panzerScripts, Script[] turretScripts)

{

Transform transform = panzer.GetComponent("transform") as Transform;

transform.Position = position;

transform.Scale = scale;

panzer.AddComponent("texture", trackTex);

panzer.AddComponent("health",

new ProgressBarHealth(new Health(health), healthBar));

foreach (Script script in panzerScripts)

panzer.AddScript(script);

GameObject turret = new GameObject(turretTex, new Vector2(

5 \* scale.X, -4 \* scale.Y), new Vector2(16, 4), scale, MathHelper.Pi / 4);

panzer.AddInnerObject(turret);

turret.AddComponent("inventory", inventory);

foreach (Script script in turretScripts)

turret.AddScript(script);

}

private void BuildFirstPanzer(Inventory inventory)

{

Texture2D trackTexture = Texture2D.LoadTexture(TRACK\_TEXTURE\_PATH);

Texture2D turretTexture = Texture2D.LoadTexture(TURRET\_TEXTURE\_PATH);

trackTexture.Color = Color.FromArgb(200, 120, 60);

turretTexture.Color = Color.FromArgb(200, 120, 60);

AddTexture(trackTexture);

AddTexture(turretTexture);

Vector2 position = new Vector2((float)-GameWindow.Width \* 3 / 4,

(float)GameWindow.Height - trackTexture.Height \* 14);

Script[] trackScripts = null;

Script[] turretScripts = null;

ProgressBar cooldownBar = (ProgressBar)GameWindow.FindName("FirstPanzerCooldown");

ProgressBar healthBar = (ProgressBar)GameWindow.FindName("FirstPanzerHealth");

switch (settings.FirstPanzerControlType)

{

case BattleSceneSettings.PanzerControlType.AI:

trackScripts = CreateTrackAIScripts();

turretScripts = CreateTurretAIScripts(cooldownBar, secondPanzer);

break;

case BattleSceneSettings.PanzerControlType.Keyboard:

trackScripts = CreateTrackKeyboardScripts(OpenTK.Input.Key.A, OpenTK.Input.Key.D);

turretScripts = CreateTurretKeyboardScripts(cooldownBar,

OpenTK.Input.Key.W, OpenTK.Input.Key.S, OpenTK.Input.Key.Space,

OpenTK.Input.Key.E, OpenTK.Input.Key.Q);

break;

}

BuildPanzer(firstPanzer, trackTexture,

turretTexture, position, new Vector2(-5, 5),

settings.FirstPanzerHealth, healthBar,

inventory, trackScripts, turretScripts);

}

private void BuildSecondPanzer(Inventory inventory)

{

Texture2D trackTexture = Texture2D.LoadTexture(TRACK\_TEXTURE\_PATH);

Texture2D turretTexture = Texture2D.LoadTexture(TURRET\_TEXTURE\_PATH);

trackTexture.Color = Color.FromArgb(20, 140, 120);

turretTexture.Color = Color.FromArgb(20, 140, 120);

AddTexture(trackTexture);

AddTexture(turretTexture);

Vector2 position = new Vector2((float)GameWindow.Width \* 3 / 4,

(float)GameWindow.Height - trackTexture.Height \* 14);

Script[] trackScripts = null;

Script[] turretScripts = null;

ProgressBar cooldownBar = (ProgressBar)GameWindow.FindName("SecondPanzerCooldown");

ProgressBar healthBar = (ProgressBar)GameWindow.FindName("SecondPanzerHealth");

switch (settings.SecondPanzerControlType)

{

case BattleSceneSettings.PanzerControlType.AI:

trackScripts = CreateTrackAIScripts();

turretScripts = CreateTurretAIScripts(cooldownBar, firstPanzer);

break;

case BattleSceneSettings.PanzerControlType.Keyboard:

trackScripts = CreateTrackKeyboardScripts(OpenTK.Input.Key.Left, OpenTK.Input.Key.Right);

turretScripts = CreateTurretKeyboardScripts(cooldownBar,

OpenTK.Input.Key.Up, OpenTK.Input.Key.Down, OpenTK.Input.Key.Enter,

OpenTK.Input.Key.Plus, OpenTK.Input.Key.Minus);

break;

}

BuildPanzer(secondPanzer, trackTexture,

turretTexture, position, new Vector2(5, 5),

settings.SecondPanzerHealth, healthBar,

inventory, trackScripts, turretScripts);

}

private Inventory.RocketBuilder[] CreateRockets()

{

Texture2D rocketTex = Texture2D.LoadTexture(ROCKET\_TEXTURE\_PATH);

AddTexture(rocketTex);

Texture2D powerfulRocketTex = Texture2D.LoadTexture(POWERFULROCKET\_TEXTURE\_PATH);

AddTexture(powerfulRocketTex);

Texture2D fastRocketTex = Texture2D.LoadTexture(FASTROCKET\_TEXTURE\_PATH);

AddTexture(fastRocketTex);

Animation2D explosionAnim = Animation2D.LoadAnimation(EXPLOSION\_ANIMATION\_PATH, 5);

explosionAnim.AnimationTime = 120;

AddTexture(explosionAnim);

return new Inventory.RocketBuilder[]

{

new Inventory.RocketBuilder(this, powerfulRocketTex, explosionAnim,

new DoubleDamageRocket(new DoubleCooldownRocket(new BaseRocket()))),

new Inventory.RocketBuilder(this, fastRocketTex, explosionAnim,

new HalfDamageRocket(new HalfCooldownRocket(new BaseRocket()))),

new Inventory.RocketBuilder(this, rocketTex, explosionAnim,

new BaseRocket())

};

}

private GameObject CreateMountain()

{

Texture2D mountainTex = Texture2D.LoadTexture(MOUNTAIN\_TEXTURE\_PATH);

AddTexture(mountainTex);

GameObject mountain = new GameObject(mountainTex,

new Vector2(-mountainTex.Width \* 5 / 2, 80),

Vector2.Zero, new Vector2(5, 5), 0);

mountain.SetCollider(new Collider(new Vector2[] {

new Vector2(-20, 80),

new Vector2(mountainTex.Width \* 5 / 2, 80 + mountainTex.Height \* 5),

new Vector2(-20, 80 + mountainTex.Height \* 10),

new Vector2(-mountainTex.Width \* 5 / 2, 80 + mountainTex.Height \* 5)

}));

return mountain;

}

private GameObject CreateBackground()

{

Texture2D backgroundTex = Texture2D.LoadTexture(BACKGROUND\_TEXTURE\_PATH);

AddTexture(backgroundTex);

return new GameObject(backgroundTex,

new Vector2((float)-GameWindow.Width, (float)-GameWindow.Height),

Vector2.Zero, new Vector2(5, 5), 0);

}

private Script[] CreateTrackKeyboardScripts(OpenTK.Input.Key left, OpenTK.Input.Key right)

{

TrackKeyboardControlScript panzerControl = new TrackKeyboardControlScript(this, 300f);

panzerControl.SetKeyToMoveLeft(left);

panzerControl.SetKeyToMoveRight(right);

return new Script[] { panzerControl };

}

private Script[] CreateTrackAIScripts()

{

return new Script[] { new TrackAIControlScript(300f, 200) };

}

private Script[] CreateTurretKeyboardScripts(ProgressBar cooldownBar,

OpenTK.Input.Key up, OpenTK.Input.Key down, OpenTK.Input.Key shoot,

OpenTK.Input.Key next, OpenTK.Input.Key previous)

{

TurretKeyboardControlScript turretControl = new TurretKeyboardControlScript(2);

turretControl.SetKeyToTurnUp(up);

turretControl.SetKeyToTurnDown(down);

ShootKeyboardControlScript shootControl = new ShootKeyboardControlScript(this);

shootControl.SetKey(shoot);

KeyboardRocketSwitcherScript rocketSwitcher = new KeyboardRocketSwitcherScript();

rocketSwitcher.SetKeyToSelectNext(next);

rocketSwitcher.SetKeyToSelectPrevious(previous);

WpfShootControlScript wpfShootControl = new WpfShootControlScript(this, cooldownBar, shootControl);

return new Script[] { turretControl, rocketSwitcher, wpfShootControl };

}

private Script[] CreateTurretAIScripts(ProgressBar cooldownBar, GameObject target)

{

TurretAIControlScript turretControl = new TurretAIControlScript(2);

ShootAIControlScript shootControl = new ShootAIControlScript(this, 500);

AIRocketSwitcherScript rocketSwitcher = new AIRocketSwitcherScript(100, 1000);

turretControl.SetTarget(target);

WpfShootControlScript wpfShootControl = new WpfShootControlScript(this, cooldownBar, shootControl);

return new Script[] { turretControl, rocketSwitcher, wpfShootControl };

}

}

}

**Код программы для BattleSceneSettings.cs:**

using GameLibrary.Components;

namespace GameLibrary.Scenes

{

/// <summary>

/// Настройки сцены.

/// </summary>

public class BattleSceneSettings

{

/// <summary>

/// Перечисление типов управления танком.

/// </summary>

public enum PanzerControlType

{

/// <summary>

/// Управление искусственным интеллектом.

/// </summary>

AI,

/// <summary>

/// Управление с клавиатуры.

/// </summary>

Keyboard

}

private const int ROCKET\_TYPES = 3;

private int[] firstPanzerRocketAmounts;

private int[] secondPanzerRocketAmounts;

/// <summary>

/// Здоровье первого танка.

/// </summary>

public int FirstPanzerHealth { get; set; }

/// <summary>

/// Здоровье второго танка.

/// </summary>

public int SecondPanzerHealth { get; set; }

/// <summary>

/// Тип управления первого танка.

/// </summary>

public PanzerControlType FirstPanzerControlType { get; set; }

= PanzerControlType.AI;

/// <summary>

/// Тип управления второго танка.

/// </summary>

public PanzerControlType SecondPanzerControlType { get; set; }

= PanzerControlType.AI;

/// <summary>

/// Создание настроек сцены.

/// </summary>

public BattleSceneSettings()

{

firstPanzerRocketAmounts = new int[ROCKET\_TYPES];

secondPanzerRocketAmounts = new int[ROCKET\_TYPES];

}

/// <summary>

/// Задание инвентаря первому танку.

/// </summary>

/// <param name="amounts">Количество ракет.</param>

public void SetFirstPanzerAmounts(params int[] amounts)

{

SetAmounts(firstPanzerRocketAmounts, amounts);

}

/// <summary>

/// Задание инвентаря второму танку.

/// </summary>

/// <param name="amounts">Количество ракет.</param>

public void SetSecondPanzerAmounts(params int[] amounts)

{

SetAmounts(secondPanzerRocketAmounts, amounts);

}

/// <summary>

/// Заполнение инвентаря первого танка.

/// </summary>

/// <param name="inventory">Инвентарь танка.</param>

public void FillFirtsPanzerInventory(Inventory inventory)

{

FillInventory(inventory, firstPanzerRocketAmounts);

}

/// <summary>

/// Заполнение инвентаря второго танка.

/// </summary>

/// <param name="inventory">Инвентарь танка.</param>

public void FillSecondPanzerInventory(Inventory inventory)

{

FillInventory(inventory, secondPanzerRocketAmounts);

}

/// <summary>

/// Заполнение инвенторя танка.

/// </summary>

/// <param name="inventory">Инвентарь для заполнения.</param>

/// <param name="amounts">Количества ракет.</param>

private void FillInventory(Inventory inventory, int[] amounts)

{

foreach (int amount in amounts)

{

inventory.SetAmount(amount);

inventory.SelectNext();

}

}

/// <summary>

/// Установка количеств ракет у танка.

/// </summary>

/// <param name="panzerAmounts">Количество ракет танка.</param>

/// <param name="amounts">Новое количество ракет танка.</param>

private void SetAmounts(int[] panzerAmounts, int[] amounts)

{

for (int i = 0; i < amounts.Length; i++)

{

panzerAmounts[i] = amounts[i];

}

}

}

}

**Код программы для AIRocketSwitcherScript.cs:**

using GameEngineLibrary;

using GameLibrary.Components;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, отвечающий за переключение ракет танка с искусственным интеллектом.

/// </summary>

public class AIRocketSwitcherScript : Script

{

private bool isCooldown = false;

private int lastPressTime = 0;

private int cooldown;

private int minCooldown;

private int maxCooldown;

private Inventory inventory;

/// <summary>

/// Создание скрипта, управляющего сменой ракет у

/// танка с искусственным интеллектом.

/// </summary>

/// <param name="minCooldown">Минимальная частота смены.</param>

/// <param name="maxCooldown">Максимальная частота смены.</param>

public AIRocketSwitcherScript(int minCooldown, int maxCooldown)

{

this.minCooldown = minCooldown;

this.maxCooldown = maxCooldown;

}

/// <summary>

/// Инициализация скрипта.

/// </summary>

public override void Init()

{

inventory = controlledObject.GetComponent("inventory") as Inventory;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

if (isCooldown)

{

lastPressTime += delta.Milliseconds;

if (lastPressTime > cooldown)

isCooldown = false;

return;

}

if (inventory.GetTotalAmount() == 0)

return;

if (RandomManager.Next(2) == 0)

do

inventory.SelectNext();

while (inventory.GetAmount() == 0);

else

do

inventory.SelectPrevious();

while (inventory.GetAmount() == 0);

cooldown = RandomManager.Next(minCooldown, maxCooldown);

lastPressTime = 0;

isCooldown = true;

}

}

}

**Код программы для ShootKeyboardControlScript.cs:**

using GameEngineLibrary;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, управляющий выстрелами танка с искусственным интеллектом.

/// </summary>

public class ShootAIControlScript : ShootKeyboardControlScript

{

private int range;

/// <summary>

/// Создание скрипта.

/// </summary>

/// <param name="scene">Сцена, в которой будут спавниться ракеты.</param>

/// <param name="range">Время между выстрелами.</param>

public ShootAIControlScript(Scene scene, int range)

: base(scene)

{

this.range = range;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

if (isCooldown)

{

UpdateCooldown(delta);

return;

}

Shoot();

Cooldown += RandomManager.Next(range);

}

}

}

**Код программы для TrackAIControlScript.cs:**

using GameEngineLibrary;

using OpenTK;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, отвечающий за движение танка.

/// </summary>

public class TrackAIControlScript : Script

{

private Transform transform;

private Vector2 speed;

private int range;

private int pointToMove;

private int minPosition;

private int maxPosition;

/// <summary>

/// Создание контроллера для танка.

/// </summary>

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

/// <param name="range">Максимальное расстояние, на которое может проехать танк.</param>

public TrackAIControlScript(float speed, int range)

{

this.speed = new Vector2(speed, 0);

this.range = range;

}

/// <summary>

/// Инициализация объекта.

/// </summary>

public override void Init()

{

transform = controlledObject.GetComponent("transform") as Transform;

minPosition = range;

maxPosition = (int) Math.Abs(transform.Position.X);

if (minPosition > maxPosition)

(minPosition, maxPosition) = (maxPosition, minPosition);

pointToMove = RandomManager.Next(minPosition, maxPosition);

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

int x = (int)Math.Abs(transform.Position.X);

if (x > pointToMove)

{

transform.Position -= speed \* (float)delta.TotalSeconds \*

Math.Sign(transform.Position.X);

x = (int)Math.Abs(transform.Position.X);

if (x < pointToMove)

pointToMove = RandomManager.Next(minPosition, maxPosition);

}

else

{

transform.Position += speed \* (float)delta.TotalSeconds \*

Math.Sign(transform.Position.X);

x = (int)Math.Abs(transform.Position.X);

if (x > pointToMove)

pointToMove = RandomManager.Next(minPosition, maxPosition);

}

controlledObject.UpdateColliderToTexture();

}

}

}

**Код программы для TurretAIControlScript.cs:**

using GameEngineLibrary;

using OpenTK;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, контролирующий управление башней танка.

/// </summary>

public class TurretAIControlScript : Script

{

private const double MAX\_ANGLE = Math.PI \* 3 / 8;

private const double MIN\_ANGLE = Math.PI / 4;

private double speed;

private Transform transform;

private Texture2D texture;

private GameObject target;

/// <summary>

/// Создание контроллера для башни танка.

/// </summary>

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

public TurretAIControlScript(double speed)

{

this.speed = speed;

}

/// <summary>

/// Инициализация скрипта.

/// </summary>

public override void Init()

{

transform = controlledObject.GetComponent("transform") as Transform;

texture = controlledObject.GetComponent("texture") as Texture2D;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

Vector2 aim = new Vector2(0, texture.Height / 2);

aim -= transform.RotationPoint;

aim = new Vector2(

(float)(Math.Cos(transform.Rotation) \* aim.X -

Math.Sin(transform.Rotation) \* aim.Y),

(float)(Math.Sin(transform.Rotation) \* aim.X +

Math.Cos(transform.Rotation) \* aim.Y));

aim += transform.RotationPoint;

aim \*= transform.Scale;

aim += transform.Position;

Transform targetTransform = target.GetComponent("transform") as Transform;

Texture2D targetTexture = target.GetComponent("texture") as Texture2D;

Vector2 enemy = new Vector2(

targetTransform.Position.X + targetTexture.Width \* targetTransform.Scale.X / 2,

targetTransform.Position.Y + targetTexture.Height \* targetTransform.Scale.Y / 2);

float height = enemy.Y - aim.Y;

float gravity = 0.2f;

Vector2 impuls = new Vector2(

(float)(-Math.Sign(transform.Scale.X) \* 15 \*

Math.Cos(transform.Rotation)),

(float)(-15 \* Math.Sin(transform.Rotation)));

float riseTime = -impuls.Y / gravity;

float path = -(impuls.Y \* riseTime -

gravity \* riseTime \* riseTime / 2);

float fallTime = (float)Math.Sqrt(2 \* (path + height) / gravity);

float time = riseTime + fallTime;

float targetX = aim.X + impuls.X \* time / 1.25f;

if (targetX \* targetX < enemy.X \* enemy.X && transform.Rotation > MIN\_ANGLE)

{

transform.Rotation -= speed \* delta.TotalSeconds;

}

if (targetX \* targetX > enemy.X \* enemy.X && transform.Rotation < MAX\_ANGLE)

{

transform.Rotation += speed \* delta.TotalSeconds;

}

}

/// <summary>

/// Установка цели, по которой будет осуществляться стрельба.

/// </summary>

/// <param name="target">Цель стрельбы.</param>

public void SetTarget(GameObject target)

{

this.target = target;

}

}

}

**Код программы для KeyboardRocketSwitcherScript.cs:**

using GameEngineLibrary;

using GameLibrary.Components;

using OpenTK.Input;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт для управления переключением ракет при помощи клавиатуры.

/// </summary>

public class KeyboardRocketSwitcherScript : Script

{

private Key next;

private Key previous;

private bool isCooldown = false;

private int lastPressTime = 0;

private const int COOLDOWN = 100;

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

if (isCooldown)

{

lastPressTime += delta.Milliseconds;

if (lastPressTime > COOLDOWN)

{

isCooldown = false;

}

return;

}

KeyboardState keyboard = Keyboard.GetState();

Inventory inventory = controlledObject.GetComponent("inventory") as Inventory;

if (keyboard[next])

{

inventory.SelectNext();

lastPressTime = 0;

isCooldown = true;

return;

}

if (keyboard[previous])

{

inventory.SelectPrevious();

lastPressTime = 0;

isCooldown = true;

return;

}

}

/// <summary>

/// Установить кнопку для выбора следующей ракеты.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKeyToSelectNext(Key key)

{

next = key;

}

/// <summary>

/// Установить кнопку для выбора предыдущей ракеты.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKeyToSelectPrevious(Key key)

{

previous = key;

}

}

}

**Код программы для ShootKeyboardControlScript.cs:**

using OpenTK.Input;

using GameEngineLibrary;

using OpenTK;

using System;

using GameLibrary.Components;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, отвечающий за стрельбу танка при помощи клавиатуры.

/// </summary>

public class ShootKeyboardControlScript : Script

{

private readonly Scene scene;

private Key shoot;

/// <summary>

/// Поле, указывающее, находится ли на перезарядке танк.

/// </summary>

protected bool isCooldown;

/// <summary>

/// Время перезарядки.

/// </summary>

public int Cooldown { get; set; }

/// <summary>

/// Время, прошедшее с последнего выстрела.

/// </summary>

public int LastShoot { get; set; }

/// <summary>

/// Создание контроллера для выстрелов танка.

/// </summary>

/// <param name="scene">Сцена, в котрой будут осуществляться выстрелы.</param>

public ShootKeyboardControlScript(Scene scene)

{

Cooldown = 0;

this.scene = scene;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

KeyboardState keyboard = Keyboard.GetState();

if (isCooldown)

{

UpdateCooldown(delta);

}

else if (keyboard[shoot])

{

Shoot();

}

}

/// <summary>

/// Установить кнопку выстрела.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKey(Key key)

{

shoot = key;

}

/// <summary>

/// Обновление состояния перезарядки.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

protected void UpdateCooldown(TimeSpan delta)

{

LastShoot += delta.Milliseconds;

if (LastShoot >= Cooldown)

{

isCooldown = false;

}

}

/// <summary>

/// Совершить выстрел ракетой.

/// </summary>

protected void Shoot()

{

Inventory inventory = controlledObject.GetComponent("inventory") as Inventory;

GameObject rocket = inventory.GetRocket();

if (rocket == null)

{

return;

}

Transform transform = controlledObject.GetComponent("transform") as Transform;

Texture2D texture = controlledObject.GetComponent("texture") as Texture2D;

Texture2D rocketTex = rocket.GetComponent("texture") as Texture2D;

float x = rocketTex.Width \* transform.Scale.X;

float y = rocketTex.Height \* transform.Scale.Y;

Vector2 spawnPoint = new Vector2(

-texture.Width \* transform.Scale.X,

-texture.Height \* transform.Scale.Y);

spawnPoint = new Vector2(

(float)(Math.Cos(transform.Rotation \*

Math.Sign(transform.Scale.X)) \* spawnPoint.X -

Math.Sin(transform.Rotation \*

Math.Sign(transform.Scale.X)) \* spawnPoint.Y),

(float)(Math.Sin(transform.Rotation \*

Math.Sign(transform.Scale.X)) \* spawnPoint.X +

Math.Cos(transform.Rotation \*

Math.Sign(transform.Scale.X)) \* spawnPoint.Y));

Vector2 rocketPoint = new Vector2(

(float)(Math.Cos(transform.Rotation \*

-Math.Sign(transform.Scale.X)) \* -x -

Math.Sin(transform.Rotation \*

-Math.Sign(transform.Scale.X)) \* -y),

(float)(Math.Sin(transform.Rotation \*

-Math.Sign(transform.Scale.X)) \* -x +

Math.Cos(transform.Rotation \*

-Math.Sign(transform.Scale.X)) \* -y));

spawnPoint.Y += texture.Height \* transform.Scale.Y / 2;

spawnPoint.X -= rocketPoint.X / 2;

spawnPoint.Y -= rocketPoint.Y / 2;

Transform rocketTransform = rocket.GetComponent("transform") as Transform;

rocketTransform.Position = transform.Position + spawnPoint;

rocketTransform.Rotation = transform.Rotation;

rocketTransform.Scale = transform.Scale;

rocket.AddScript(new PhysicScript(

new Vector2((float)(-Math.Sign(transform.Scale.X) \* 15 \* Math.Cos(transform.Rotation)),

(float)(-15 \* Math.Sin(transform.Rotation))),

new Vector2(0, 0.2f)));

scene.AddGameObject(rocket);

Rocket rocketComponent = rocket.GetComponent("rocket") as Rocket;

Cooldown = rocketComponent.Cooldown;

LastShoot = 0;

isCooldown = true;

}

}

}

**Код программы для TrackKeyboardControlScript.cs:**

using OpenTK.Input;

using GameEngineLibrary;

using OpenTK;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, отвечающий за передвижение танка при помощи клавиатуры.

/// </summary>

public class TrackKeyboardControlScript : Script

{

private Key left;

private Key right;

private Vector2 speed;

Scene scene;

/// <summary>

/// Создание контроллера для танка.

/// </summary>

/// <param name="scene">Сцена, в которой происходит перемещение объекта.</param>

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

public TrackKeyboardControlScript(Scene scene, float speed)

{

this.scene = scene;

this.speed = new Vector2(speed, 0);

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

KeyboardState keyboard = Keyboard.GetState();

Transform transform = controlledObject.GetComponent("transform") as Transform;

Vector2 translate = Vector2.Zero;

if (keyboard[left])

{

translate -= speed \* (float)delta.TotalSeconds;

}

if (keyboard[right])

{

translate += speed \* (float)delta.TotalSeconds;

}

transform.Position += translate;

controlledObject.UpdateColliderToTexture();

Collider thisCollider = controlledObject.GetComponent("collider") as Collider;

foreach (GameObject gameObject in scene.GetGameObjects())

{

Collider collider = gameObject.GetComponent("collider") as Collider;

if (gameObject != controlledObject &&

collider != null &&

collider.CheckCollision(thisCollider))

{

transform.Position -= translate;

controlledObject.UpdateColliderToTexture();

break;

}

}

}

/// <summary>

/// Установить кнопку для движения влево.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKeyToMoveLeft(Key key)

{

left = key;

}

/// <summary>

/// Установить кнопку для движения вправо.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKeyToMoveRight(Key key)

{

right = key;

}

}

}

**Код программы для TurretKeyboardControlScript.cs:**

using GameEngineLibrary;

using OpenTK.Input;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, контролирующий управление башней танка.

/// </summary>

public class TurretKeyboardControlScript : Script

{

private const double MAX\_ANGLE = Math.PI \* 3 / 8;

private const double MIN\_ANGLE = Math.PI / 8;

private Transform transform;

private Key up;

private Key down;

private double speed;

/// <summary>

/// Создание контроллера для башни танка.

/// </summary>

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

public TurretKeyboardControlScript(double speed)

{

this.speed = speed;

}

/// <summary>

/// Инициализация скрипта.

/// </summary>

public override void Init()

{

transform = controlledObject.GetComponent("transform") as Transform;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

KeyboardState keyboard = Keyboard.GetState();

if (keyboard[up] && transform.Rotation < MAX\_ANGLE)

{

transform.Rotation += speed \* delta.TotalSeconds;

}

if (keyboard[down] && transform.Rotation > MIN\_ANGLE)

{

transform.Rotation -= speed \* delta.TotalSeconds;

}

}

/// <summary>

/// Установить кнопку для поворота вверх.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKeyToTurnUp(Key key)

{

up = key;

}

/// <summary>

/// Установить кнопку для поворота вниз.

/// </summary>

/// <param name="key">Кнопка на клавиатуре.</param>

public void SetKeyToTurnDown(Key key)

{

down = key;

}

}

}

**Код программы для AutoDestroyScript.cs:**

using System;

using GameEngineLibrary;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, который отвечает за автоматическое уничтожение объекта через определенный промежуток времени.

/// </summary>

public class AutoDestroyScript : Script

{

/// <summary>

/// Сцена, из которой будет удален объект.

/// </summary>

private Scene scene;

/// <summary>

/// Время с создания объекта в миллисекундах.

/// </summary>

private int currentTime;

/// <summary>

/// Время уничтожения объекта.

/// </summary>

private int destroyTime;

/// <summary>

/// Создание скрипта для автоудаления объекта через определенный промежуток времени.

/// </summary>

/// <param name="scene">Сцена из которой будет удален объект.</param>

/// <param name="milliseconds">Количество миллисекунд, через которое будет удален объект.</param>

public AutoDestroyScript(Scene scene, int milliseconds)

{

this.scene = scene;

destroyTime = milliseconds;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

currentTime += delta.Milliseconds;

if (currentTime >= destroyTime)

{

scene.RemoveGameObject(controlledObject);

}

}

}

}

**Код программы для PhysicScript.cs:**

using GameEngineLibrary;

using OpenTK;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Класс, описывающий физику объекта,

/// на который действуют различные силы.

/// </summary>

public class PhysicScript : Script

{

/// <summary>

/// Импульс объекта.

/// </summary>

private Vector2 impulse;

/// <summary>

/// Силы, действующие на объект.

/// </summary>

private Vector2[] forces;

/// <summary>

/// Создание нового объекта, на которого действуют силы.

/// </summary>

/// <param name="impulse">Начальный импульс объекта.</param>

/// <param name="forces">Силы, действующие на объект.</param>

public PhysicScript(Vector2 impulse, params Vector2[] forces)

{

this.impulse = impulse;

this.forces = (Vector2[])forces.Clone();

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

Transform transform = controlledObject.GetComponent("transform") as Transform;

transform.Position += impulse;

transform.Rotation = Math.Sign(transform.Scale.X) \*

Math.Atan(impulse.Y / impulse.X);

foreach (Vector2 force in forces)

{

impulse += force;

}

}

}

}

**Код программы для RocketHitScript.cs:**

using GameEngineLibrary;

using GameLibrary.Components;

using System;

namespace GameLibrary.Scripts

{

/// <summary>

/// Класс, отвечающий за проверку поведение

/// при попадании ракеты по цели.

/// </summary>

public class RocketHitScript : Script

{

/// <summary>

/// Сцена, на которой обрабатываются попадания.

/// </summary>

private Scene scene;

/// <summary>

/// Анимация взрыва ракеты.

/// </summary>

private Animation2D explosionAnim;

/// <summary>

/// Создание скрипта, отвечающего за обрапотку попаданий ракеты.

/// </summary>

/// <param name="scene">Сцена, в которой будет проверяться столкновения.</param>

/// <param name="explosionAnim">Анимация взрыва.</param>

public RocketHitScript(Scene scene, Animation2D explosionAnim)

{

this.scene = scene;

this.explosionAnim = explosionAnim;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

controlledObject.UpdateColliderToTexture();

Transform transform = controlledObject.GetComponent("transform") as Transform;

if (CheckBounds(transform))

{

scene.RemoveGameObject(controlledObject);

return;

}

GameObject[] objects = scene.GetGameObjects().ToArray();

Collider thisCollider = controlledObject.GetComponent("collider") as Collider;

foreach (GameObject gameObject in objects)

{

Collider collider = gameObject.GetComponent("collider") as Collider;

if (gameObject != controlledObject &&

collider != null &&

collider.CheckCollision(thisCollider))

{

GameObject explosion = new GameObject(new Animation2D(explosionAnim),

new OpenTK.Vector2(transform.Position.X + transform.RotationPoint.X \*

transform.Scale.X - explosionAnim.Width / 2 \* transform.Scale.X,

transform.Position.Y + transform.RotationPoint.Y \*

transform.Scale.Y - explosionAnim.Height / 2 \* transform.Scale.Y),

new OpenTK.Vector2(explosionAnim.Width / 2, explosionAnim.Height / 2),

transform.Scale, 0);

explosion.AddScript(new AutoDestroyScript(scene, explosionAnim.AnimationTime));

scene.AddGameObject(explosion);

scene.RemoveGameObject(controlledObject);

if (gameObject.GetComponent("rocket") is Rocket)

{

scene.RemoveGameObject(gameObject);

return;

}

Rocket rocket = controlledObject.GetComponent("rocket") as Rocket;

Health health = gameObject.GetComponent("health") as Health;

if (health != null)

{

health.Damage(rocket.Damage);

if (!health.IsAlive())

{

scene.RemoveGameObject(gameObject);

}

return;

}

}

}

}

/// <summary>

/// Проверка рокины на выход за границы экрана.

/// </summary>

/// <param name="transform">Проверяемая позиция.</param>

/// <returns>True, если ракета за границами экрана.</returns>

private bool CheckBounds(Transform transform)

{

return transform.Position.X > scene.GameWindow.Width ||

transform.Position.X < -scene.GameWindow.Width ||

transform.Position.Y > scene.GameWindow.Height ||

transform.Position.Y < -scene.GameWindow.Height;

}

}

}

**Код программы для WinCheckerScript.cs:**

using GameEngineLibrary;

using GameLibrary.Components;

using System;

using System.Windows.Controls;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, проверяющий, победил ли кто-либо из игроков.

/// </summary>

public class WinCheckerScript : Script

{

private Health firstHealth;

private Health secondHealth;

private StackPanel winMenu;

private bool isWin = false;

/// <summary>

/// Создание скрипта для проверки побед.

/// </summary>

/// <param name="first">Первый танк.</param>

/// <param name="second">Второй танк.</param>

/// <param name="winMenu">Меню, которое будет отображено в случае победы.</param>

public WinCheckerScript(GameObject first, GameObject second, StackPanel winMenu)

{

firstHealth = first.GetComponent("health") as Health;

secondHealth = second.GetComponent("health") as Health;

this.winMenu = winMenu;

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

if (!isWin && (!firstHealth.IsAlive() || !secondHealth.IsAlive()))

{

winMenu.Visibility = System.Windows.Visibility.Visible;

isWin = true;

}

}

}

}

**Код программы для WpfShootControlScript.cs:**

using System;

using System.Windows.Controls;

using GameEngineLibrary;

namespace GameLibrary.Scripts

{

/// <summary>

/// Скрипт, отображающий состояние перезарядки.

/// </summary>

public class WpfShootControlScript : ShootKeyboardControlScript

{

/// <summary>

/// Индикатор, отображающий состояние перезарядки.

/// </summary>

private ProgressBar cooldown;

private ShootKeyboardControlScript shootControlScript;

/// <summary>

/// Создание нового скрипта, который отображает состояние перезарядки на окно WPF.

/// </summary>

/// <param name="scene">Сцена, в которой происходит стрельба.</param>

/// <param name="cooldown">Индикатор перезарядки.</param>

/// <param name="shootControlScript">Декорируемый экземпдяр.</param>

public WpfShootControlScript(Scene scene, ProgressBar cooldown, ShootKeyboardControlScript shootControlScript) : base(scene)

{

this.cooldown = cooldown;

this.shootControlScript = shootControlScript;

}

/// <summary>

/// Инициализация скрипта.

/// </summary>

public override void Init()

{

shootControlScript.SetControlledObject(controlledObject);

shootControlScript.Init();

}

/// <summary>

/// Обновление состояния скрипта.

/// </summary>

/// <param name="delta">Время, прошедшее с предыдущего кадра.</param>

public override void Update(TimeSpan delta)

{

shootControlScript.Update(delta);

cooldown.Maximum = shootControlScript.Cooldown;

cooldown.Value = shootControlScript.LastShoot;

}

}

}

# ПРИЛОЖЕНИЕ Б

(обязательное)

**Руководство пользователя**

1. Введение.

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

Игровое приложение обладает следующим функционалом:

* игровой уровень в виде поля танкового сражения с препятствием в виде непреодолимого горного хребта;
* передвижение игроков;
* покупка боеприпасов.

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 одного или 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* 2 *Duo* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора;
* операционная система *Windows* 7 и новее;
* 100 Мб оперативной памяти;

1. Подготовка к работе.

Приложение запускается путём открытия файла *MainWindow.exe*. Также на компьютере должны быть установлены драйвера для видеокарты. Если все инструкции соблюдены и приложение не выдаёт никаких сообщений об ошибках, значит программа работает исправно.

1. Описание операций.

При запуске приложения открывается интерфейс, который содержит кнопку запуска игры. Первый игрок передвигается с помощью клавиш *W*, *A*, *S*, *D* и стреляет при нажатии клавиши *Space*. Переключение ракет происходит при нажатии клавиш *Q*/*E*. Второй игрок передвигается с помощью клавиш *Up*, *Left*, *Down*, *Right* и стреляет при нажатии клавиши *Enter*. Переключение патрон происходит при нажатии клавиш -/+.

Первый игрок управляет синим танком, а второй игрок – желтым. Цель каждого игрока заключается в уничтожении танка противника. В процессе игры можно передвигаться по игровому полю по горизонтали. Стрелять можно по параболической траектории, чтобы ракета перелетала через гору. При столкновении с горой ракета уничтожается.

В процессе игры можно использовать боеприпасы трёх типов:

* боеприпасы со средним уроном и средней скорострельностью;
* боеприпасы с высоким уроном и низкой скорострельностью;
* боеприпасы с низким уроном и высокой скорострельностью.

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 В

(обязательное)

**Руководство программиста**

1. Назначения и условия применения программы.

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

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

* центральный процессор *Intel Core* 2 *Duo* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора;
* операционная система *Windows* 7 и новее;
* 100 Мб оперативной памяти;

1. Характеристики программы.

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

1. Обращение к программе.

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

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.

В процессе работы приложения игровая статистика выводится в информационные лейблы каждого игрока. Вначале игры выводится информация о количестве денег и ракет у каждого из игроков.

# ПРИЛОЖЕНИЕ Г

(обязательное)

**Руководство системного программиста**

1. Общие сведения о программе.

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

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

* центральный процессор *Intel Core* 2 *Duo* c тактовой частотой 2.30 МГц или более;
* наличие клавиатуры, мыши и цветного монитора;
* операционная система *Windows* 7 или новее;
* 100 Мб оперативной памяти;

1. Структура программы.

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

1. Настройка программы.

Для редактирования кода программы 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.

1. Дополнительные возможности.

Приложение не имеет дополнительных возможностей.

1. Сообщение системному программисту.

Проект игрового движка можно использовать для разработки других игр. Проект игровой логики можно дополнить, добавив новые виды ракет, а также более сложный искусственный интеллект.

# ПРИЛОЖЕНИЕ Д

(обязательное)

**Схема использования паттерна «Декоратор»**