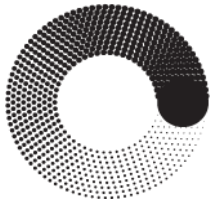
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(МОСКОВСКИЙ ПОЛИТЕХ)



МОСКОВСКИЙ
ПОЛИТЕХ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Мобильная разработка»

на тему «Разработка desktop-приложения «Геометрический тренажер»
на языке C# с использованием Windows Forms»

Выполнила:
Кучерова Мария Андреевна
Группа:
221–361

Москва, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, насыщенном информацией и цифровыми технологиями, развитие когнитивных способностей человека приобретает особую значимость. Способность к логическому анализу, пространственное воображение и комбинаторное мышление являются ключевыми компетенциями не только для инженеров, программистов и ученых, но и для специалистов в самых разных областях. Одним из наиболее эффективных и увлекательных способов тренировки этих навыков являются логические игры и головоломки.

Актуальность темы

Актуальность данной работы обусловлена растущим спросом на образовательные и развивающие программные продукты. Компьютерные игры давно перестали быть исключительно развлекательным контентом, превратившись в мощный инструмент обучения и тренировки. Геометрические головоломки, подобные классической игре «Танграм», эффективно развивают пространственное мышление, учат видеть целое через его части, анализировать и синтезировать сложные формы. Однако многие существующие цифровые аналоги таких игр имеют ряд ограничений: они либо предлагают статический набор заданий, либо лишены инструментов для творчества и создания собственных головоломок.

Разработка desktop-приложения «Геометрический тренажер» направлена на решение этой проблемы. Проект предлагает не только игровой режим для решения готовых задач, но и мощный редактор, позволяющий пользователям самим создавать и модифицировать фигуры. Ключевой особенностью проекта является инновационный подход к представлению и обработке фигур, основанный на побитовых операциях. Это не только делает приложение интересным с точки зрения программирования, но и обеспечивает высокую производительность и компактность хранения данных, что является

важным аспектом при разработке эффективного программного обеспечения. Таким образом, проект актуален как с точки зрения создания полезного развивающего инструмента, так и с точки зрения исследования и применения нетривиальных алгоритмических подходов в прикладном программировании.

Цель работы

Целью настоящего курсового проекта является разработка программного продукта – desktop-приложения «Геометрический тренажер» на языке программирования C# с использованием технологической платформы Windows Forms. Приложение должно предоставлять пользователю функционал для интерактивного создания, редактирования, сохранения геометрических фигур, а также игровой режим для решения логических задач на основе операций над этими фигурами.

Задачи работы

Для достижения поставленной цели необходимо было решить следующий перечень задач:

1) провести анализ предметной области – изучить принципы геометрических головоломок и логических операций над фигурами;

2) выполнить обзор существующих программных аналогов для выявления их преимуществ, недостатков и определения уникальных особенностей разрабатываемого продукта;

3) осуществить выбор и технико-экономическое обоснование средств реализации проекта, включая язык программирования, платформу и среду разработки;

4) сформулировать детальные функциональные и технические требования к приложению;

5) спроектировать архитектуру программного продукта, определив его основные модули и взаимосвязи между ними;

6) разработать эффективную модель данных для представления геометрических фигур, позволяющую легко выполнять над ними логические и трансформационные операции;

7) спроектировать и реализовать ключевые алгоритмы: логические операции (объединение, вычитание, пересечение), геометрические трансформации (поворот, зеркальное отражение), алгоритмы отрисовки и интерактивного взаимодействия;

8) спроектировать и реализовать интуитивно понятный пользовательский интерфейс, включающий несколько окон: главное меню, редактор фигур, игровой режим;

9) реализовать механизм сохранения и загрузки пользовательских фигур в файловую систему.

Объект и предмет исследования

Объект исследования: процесс разработки desktop-приложения для образовательных и развивающих целей на платформе .NET.

Предмет исследования: методы, модели и алгоритмы для представления, визуализации и выполнения логико-геометрических операций над двумерными составными фигурами на основе применения побитовых операций и целочисленного кодирования.

Практическая значимость

Практическая значимость разработанного приложения заключается в его потенциальном применении в нескольких областях:

- в образовании: приложение может использоваться на уроках геометрии и информатики для наглядной демонстрации логических операций, комбинаторики и основ алгоритмизации. Оно способствует развитию пространственного воображения и логического мышления у школьников и студентов;

- в качестве развивающего инструмента: «Геометрический тренажер» может служить инструментом для когнитивной тренировки для людей всех возрастов, помогая поддерживать и улучшать такие навыки, как внимание, память и способность к решению нестандартных задач;
- как платформа для творчества: наличие редактора фигур позволяет пользователям создавать собственные уникальные головоломки и делиться ими, что превращает приложение из простого тренажера в творческую площадку;
- как прототип для дальнейших разработок: реализованные в проекте архитектура и алгоритмы могут послужить прочной основой для создания более сложных коммерческих или open-source проектов, включая мобильные игры, онлайн-платформы для соревнований и более сложные симуляторы.

1. АНАЛИТИЧЕСКИЙ ОБЗОР И ПОСТАНОВКА ЗАДАЧИ

На данном этапе разработки необходимо провести комплексный анализ предметной области, изучить существующие на рынке решения, сделать обоснованный выбор технологического стека и четко сформулировать требования к будущему приложению. Этот раздел закладывает теоретический и концептуальный фундамент для всего проекта.

1.1. Анализ предметной области

Предметная область данного проекта лежит на стыке геометрии, логики и интерактивных развлечений. В основе лежит концепция составных геометрических фигур, которые можно комбинировать и преобразовывать для получения новых, более сложных форм. Историческим и идейным прародителем таких головоломок является древняя китайская игра «Танграм». В ней игроку необходимо сложить из семи стандартных плоских фигур («танов») заданный силуэт. Ключевой принцип – использование всех частей без перекрытия.

Разрабатываемый «Геометрический тренажер» заимствует и расширяет эту концепцию, переводя ее в цифровой формат и обогащая новыми возможностями. Игровая механика приложения строится на четырех базовых принципах, описанных ниже.

1. Дискретное представление пространства: игровая область представляет собой сетку, разделенную на элементарные, неделимые компоненты. В данном проекте такой элементарной единицей является равнобедренный прямоугольный треугольник. Все фигуры строятся из этих базовых «атомов». Такое представление идеально подходит для цифровой обработки и хранения.

2. Композиция и декомпозиция фигур: основная деятельность пользователя связана с созданием сложных фигур из простых и, наоборот, анализом сложных фигур. Это реализуется через набор логических операций.

3. Логические операции: в отличие от физических головоломок, цифровой формат позволяет реализовать операции, заимствованные из булевой алгебры и теории множеств, но примененные к геометрии. Основные операции в проекте:

- сложение (объединение): аналог операции UNION в теории множеств или логического OR. Если взять две фигуры A и B, то их сумма A + B будет представлять собой фигуру, включающую все элементарные треугольники, которые принадлежат либо A, либо B, либо им обоим;
- вычитание (разность): аналог операции DIFFERENCE или A AND NOT B. Результатом операции A - B будет фигура, состоящая из тех треугольников, которые принадлежат A, но не принадлежат B;
- пересечение (умножение): аналог операции INTERSECTION или логического AND. Результатом операции A × B будет фигура, содержащая только те треугольники, которые одновременно принадлежат и фигуре A, и фигуре B.

4. Геометрические трансформации: для усложнения задач и расширения возможностей пользователя вводятся стандартные аффинные преобразования на плоскости:

- поворот: изменение ориентации фигуры вокруг центра на 90 градусов по или против часовой стрелки;
- зеркальное отражение: создание симметричной копии фигуры относительно вертикальной оси.

Таким образом, предметная область проекта – это создание цифровой «песочницы» для манипулирования дискретными геометрическими объектами с помощью четко определенных логических и трансформационных правил. Это открывает широкие возможности для создания бесконечного числа головоломок различной сложности.

1.2. Обзор существующих аналогов

Для определения конкурентных преимуществ и уникального торгового предложения разрабатываемого продукта был проведен анализ существующих на рынке аналогов. Были рассмотрены следующие приложения.

1) Polygrams - Tangram Puzzles (Мобильная игра, iOS/Android): популярная мобильная игра, основанная на классическом Танграме. Игроку предлагается огромное количество уровней, где нужно из заданного набора цветных полигонов сложить определенный силуэт.

Достоинства:

- привлекательный, современный пользовательский интерфейс;
- большое количество готовых уровней и регулярные обновления;
- простое и интуитивное управление (drag-and-drop);
- система подсказок и достижений.

Недостатки:

- отсутствие редактора фигур. Пользователь ограничен стандартным набором элементов;
- механика ограничена только сложением фигур. Логические операции вычитания и пересечения отсутствуют;
- наличие рекламы и встроенных покупок, что может отвлекать от игрового процесса.

2) Block Puzzle (Различные вариации, PC/Mobile): жанр игр, где необходимо заполнить заданную область фигурами, состоящими из блоков (как в «Тетрисе»). Фигуры нельзя вращать, только перетаскивать.

Достоинства:

- простая и затягивающая игровая механика;
- не требует длительного обучения.

Недостатки:

- крайне ограниченный функционал: только перетаскивание, нет трансформаций и логических операций;

- нет режима творчества или редактора;
- однообразный игровой процесс.

3) GeoGebra (Образовательное ПО, Desktop/Web): мощный математический инструмент для изучения геометрии, алгебры, статистики и анализа. Позволяет создавать сложные интерактивные чертежи и модели.

Достоинства:

- огромная функциональность для математического моделирования;
- поддержка скриптов, анимации, 3D-графики;
- полная свобода в создании геометрических построений.

Недостатки:

- избыточная сложность для целей простой головоломки. GeoGebra – это профессиональный инструмент, а не игра;
- высокий порог вхождения для рядового пользователя;
- отсутствие структурированного игрового режима с уровнями и очками.

1.3. Отличия и преимущества разрабатываемого проекта

На фоне проанализированных аналогов «Геометрический тренажер» занимает уникальную нишу. Он сочетает в себе простоту и увлекательность игрового процесса (как в Polygrams) с элементами творчества и кастомизации, отсутствующими в большинстве казуальных игр. Ключевые отличия:

- наличие редактора фигур: это главная особенность, позволяющая пользователям не быть пассивными потребителями контента, а активно создавать свои собственные фигуры и головоломки;
- реализация полного набора логических операций: в отличие от аналогов, где есть только сложение, наш проект предлагает вычитание и пересечение, что значительно обогащает игровую логику и позволяет создавать более сложные и интересные задачи;

- уникальная модель данных: использование побитовых операций для манипуляции фигурами является нетривиальным техническим решением, которое обеспечивает высокую производительность и представляет академический интерес;
- сбалансированность: проект нацелен на то, чтобы быть достаточно простым для казуального игрока, но при этом достаточно функциональным для энтузиастов головоломок.

1.4. Выбор и обоснование средств реализации

Выбор технологического стека является одним из самых ответственных этапов, поскольку он определяет не только процесс разработки, но и будущие характеристики продукта: производительность, переносимость, возможности для расширения.

1.4.1. Язык программирования

В качестве основного языка программирования был выбран C# (Си-шарп). Этот выбор обусловлен следующими его преимуществами:

- объектно-ориентированность: C# является полностью объектно-ориентированным языком, что позволяет строить гибкую, модульную и легко масштабируемую архитектуру приложения. Концепции инкапсуляции, наследования и полиморфизма идеально подходят для моделирования таких сущностей, как «Фигура», «Форма», «Игра».
- строгая типизация и безопасность: C# – статически типизированный язык. Это означает, что большинство ошибок, связанных с несоответствием типов данных, выявляются на этапе компиляции, а не во время выполнения программы, что значительно повышает надежность и стабильность приложения.
- мощная стандартная библиотека (BCL): .NET Framework, на котором базируется C#, предоставляет обширную библиотеку классов для работы с коллекциями, файловым вводом-выводом, графикой, сетевыми протоколами и многим другим. Это избавляет от необходимости писать

низкоуровневый код и позволяет сконцентрироваться на бизнес-логике приложения.

- управляемый код и сборка мусора: Программы на C# выполняются в среде CLR (Common Language Runtime), которая берет на себя управление памятью. Автоматическая сборка мусора (Garbage Collector) освобождает разработчика от ручного выделения и освобождения памяти, что снижает риск утечек памяти и других трудноуловимых ошибок.
- отличная интеграция со средой разработки: C# неразрывно связан с интегрированной средой разработки (IDE) Microsoft Visual Studio, которая предоставляет лучшие в своем классе инструменты для написания кода, отладки, профилирования и дизайна интерфейсов.

1.4.2. Платформа

Приложение разработано для платформы .NET Framework. На момент начала разработки это был зрелый, стабильный и широко распространенный фреймворк для создания desktop-приложений под управлением ОС Windows. Он обеспечивает всю необходимую инфраструктуру (CLR, BCL) для выполнения C#-кода. Хотя существует более современный .NET Core (и его преемник .NET 5+), для целей данного проекта, не требующего кроссплатформенности, возможностей .NET Framework более чем достаточно.

1.4.3. Технология GUI

Для создания графического пользовательского интерфейса (GUI) была выбрана технология Windows Forms. Этот выбор был сделан после сравнения с более современной технологией WPF (Windows Presentation Foundation):

- простота и скорость разработки: Windows Forms предлагает визуальный дизайнер с подходом drag-and-drop, что позволяет очень быстро создавать прототипы и готовые интерфейсы. Модель, основанная на событиях (event-driven), интуитивно понятна и проста в освоении;
- достаточная производительность: для 2D-графики и интерфейсов, не требующих сложных анимаций, векторной графики или аппаратного ускорения, производительности GDI+, на которой базируется WinForms,

вполне хватает. Проект не содержит визуальных эффектов, которые бы требовали мощностей WPF;

- низкий порог вхождения: по сравнению с WPF, которая требует изучения языка разметки XAML и концепций вроде привязки данных (Data Binding) и команд, Windows Forms является значительно более простой технологией. Для курсового проекта, где акцент делается на алгоритмической части, а не на дизайне интерфейса, WinForms является оптимальным выбором.

В то время как WPF предоставляет больше гибкости в дизайне и лучше подходит для современных, стилизованных интерфейсов, для «Геометрического тренажера» с его утилитарным и функциональным дизайном, преимущества WinForms в скорости и простоте разработки перевешивают.

1.4.3. Среда разработки

В качестве среды разработки была использована Microsoft Visual Studio. Это де-факто стандарт для разработки на C# и .NET. Ее использование предоставляет следующие ключевые преимущества:

- интеллектуальный редактор кода: IntelliSense, автодополнение, рефакторинг и анализ кода в реальном времени значительно ускоряют написание кода и снижают количество ошибок;
- мощный отладчик: возможность устанавливать точки останова, пошагово выполнять код, просматривать значения переменных, стек вызовов и состояние памяти является незаменимым инструментом для поиска и исправления ошибок;
- визуальный дизайнер: встроенный дизайнер для Windows Forms позволяет визуально конструировать окна и элементы управления, что упрощает разработку GUI;
- интеграция с системой контроля версий: встроенная поддержка Git упрощает управление исходным кодом проекта.

1.5. Постановка задачи

На основе проведенного анализа были сформулированы следующие требования к разрабатываемому приложению «Геометрический тренажер».

1.5.1. Функциональные требования

Главное меню: приложение должно иметь главное меню, предоставляющее пользователю выбор между двумя основными режимами: «Редактор фигур» и «Играть».

Редактор фигур:

- должен быть реализован полнофункциональный редактор для создания и модификации геометрических фигур;
- рабочая область редактора должна представлять собой сетку 4x4, состоящую из 16 элементарных треугольников;
- пользователь должен иметь возможность интерактивно, с помощью клика мыши, включать или выключать каждый из 16 треугольников, формируя таким образом фигуру;
- редактор должен отображать 16-битное представление текущей фигуры в реальном времени;
- пользователь должен иметь возможность задать уникальное текстовое имя для создаваемой фигуры.

Сохранение и загрузка фигур:

- приложение должно поддерживать сохранение созданных пользователем фигур в локальный файл;
- должен быть реализован механизм загрузки ранее сохраненных фигур для их последующего редактирования или использования в игре;
- формат хранения должен быть простым и текстовым.

В редакторе должны быть реализованы функции для трансформации текущей фигуры:

- поворот на 90 градусов влево (против часовой стрелки);
- поворот на 90 градусов вправо (по часовой стрелке);
- зеркальное отражение по вертикали.

Игровой режим:

- должен быть реализован игровой режим, представляющий собой головоломку;
- перед началом игры пользователь должен иметь возможность выбрать режим: играть со стандартными (сохраненными в файле) фигурами или со случайно сгенерированными;
- в каждом раунде игры пользователю представляется задача: две исходные фигуры (A и B) и логическая операция над ними (+, -, ×);
- пользователю предлагается 6 вариантов ответа, из которых только один является правильным результатом выполнения операции;
- пользователь должен выбрать правильный вариант ответа;
- приложение должно вести подсчет очков: начислять очки за правильный ответ и снимать за неправильный.

Уровни сложности:

- в игровом режиме должна быть возможность выбора уровня сложности (легко, средне, сложно);
- уровень сложности должен влиять на тип предлагаемых логических операций (например, на легком уровне только сложение, на сложном – все три).

1.5.2. Технические требования:

Технические требования включают в себя следующие пункты:

- приложение должно быть разработано на языке C# с использованием платформы .NET Framework;
- графический интерфейс должен быть реализован с помощью технологии Windows Forms;
- архитектура приложения должна быть модульной, с логическим разделением кода на слои (ядро, редактор, игра, интерфейс);
- представление фигуры должно быть реализовано с помощью 16-битного беззнакового целого числа (ushort);

- все логические и трансформационные операции над фигурами должны быть реализованы с использованием побитовых операций для максимальной производительности.

2. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

На этапе проектирования закладываются основы будущей программной системы. Определяется ее общая структура, разрабатываются модели данных, продумываются ключевые алгоритмы и проектируется пользовательский интерфейс. Качественное проектирование является залогом успешной реализации, поддержки и дальнейшего развития продукта.

2.1. Архитектура приложения

Для обеспечения гибкости, масштабируемости и простоты поддержки была выбрана многоуровневая (или модульная) архитектура. Весь код приложения логически разделен на несколько независимых, но взаимодействующих между собой компонентов, реализованных в виде отдельных пространств имен (namespace). Такой подход позволяет изолировать различные аспекты функциональности: базовую логику, логику редактора, игровую логику и элементы пользовательского интерфейса.

Структура приложения включает следующие основные модули:

1) GeometryGame.Core: этот модуль является фундаментом всего приложения. Он не зависит ни от каких других модулей проекта и содержит самые базовые и переиспользуемые компоненты. В него входят:

- определение модели данных (Shape);
- реализация всех базовых операций над фигурами (ShapeOperations), включая логические операции и трансформации;
- вспомогательные классы, например, для математических вычислений (TriangleHelper).

Этот модуль может быть вынесен в отдельную библиотеку классов (DLL) и использован в других проектах.

2) GeometryGame.Editor (модуль редактора): этот модуль отвечает за всю функциональность, связанную с созданием и редактированием фигур. Он

зависит от модуля Core, так как использует классы Shape и ShapeOperations. Основные компоненты:

- форма ShapeEditorForm – основной интерфейс редактора;
- форма ShapeSelectorForm – интерфейс для выбора, удаления и создания новых фигур.

3) GeometryGame.Game (модуль игры): этот модуль инкапсулирует всю игровую логику. Он также зависит от модуля Core для получения данных о фигурах и выполнения операций над ними. Основные компоненты:

- форма GameModeSelectionForm – окно выбора режима игры;
- форма ShapeGameForm – основной игровой экран, где генерируются задания и обрабатываются ответы пользователя.

4) GeometryGame.UI (модуль пользовательского интерфейса): это самый верхний уровень архитектуры. Он отвечает за запуск приложения и отображение главного меню. Он зависит от модулей Editor и Game для того, чтобы открывать соответствующие окна по действиям пользователя. Основные компоненты:

- класс Program – точка входа в приложение;
- форма MainMenu – главное меню приложения.

Такая архитектура, представленная на рисунке 1, обладает рядом преимуществ:

- низкая связанность (Low Coupling): Модули минимально зависят друг от друга. Например, ядро Core ничего не знает о том, как его используют – в игре или в редакторе;
- высокая связность (High Cohesion): Код внутри каждого модуля решает одну, четко определенную задачу;
- переиспользуемость: Модуль Core можно легко использовать в другом приложении;
- упрощение разработки и тестирования: Каждый модуль можно разрабатывать и тестировать независимо от других.

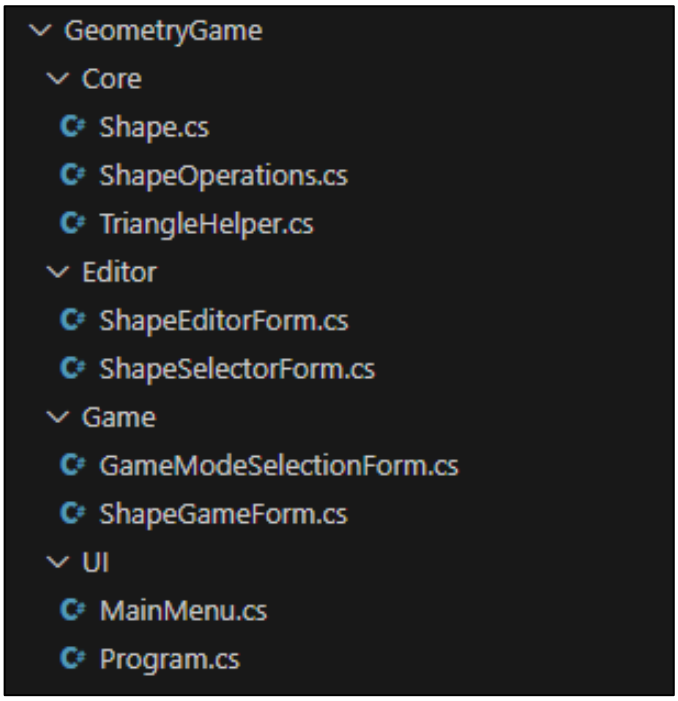


Рисунок 1 – Архитектура приложения

2.2. Проектирование модели данных

Ключевым решением, определившим всю дальнейшую реализацию, является выбор способа представления геометрической фигуры. Вместо традиционных подходов, использующих массивы векторов или списки полигонов, была разработана компактная и эффективная модель на основе битовой маски.

Рабочая область для создания фигур представляет собой большой квадрат, разделенный на сетку 4x4. Однако элементарной единицей является не маленький квадрат, а равнобедренный прямоугольный треугольник. Это достигается путем деления каждого из 4 больших квадратов (размером 2x2) на 4 таких треугольника, сходящихся в центре. Таким образом, вся рабочая область состоит из 4 * 4 = 16 элементарных треугольников.

Это количество идеально соответствует разрядности 16-битного беззнакового целого числа – ushort в C#. Каждый бит в этом числе ставится в соответствие одному из 16 треугольников на поле. Если бит установлен в 1, соответствующий треугольник считается частью фигуры (закрашен). Если бит равен 0, треугольник пуст.

Эта модель данных проиллюстрирована на рисунке 2.

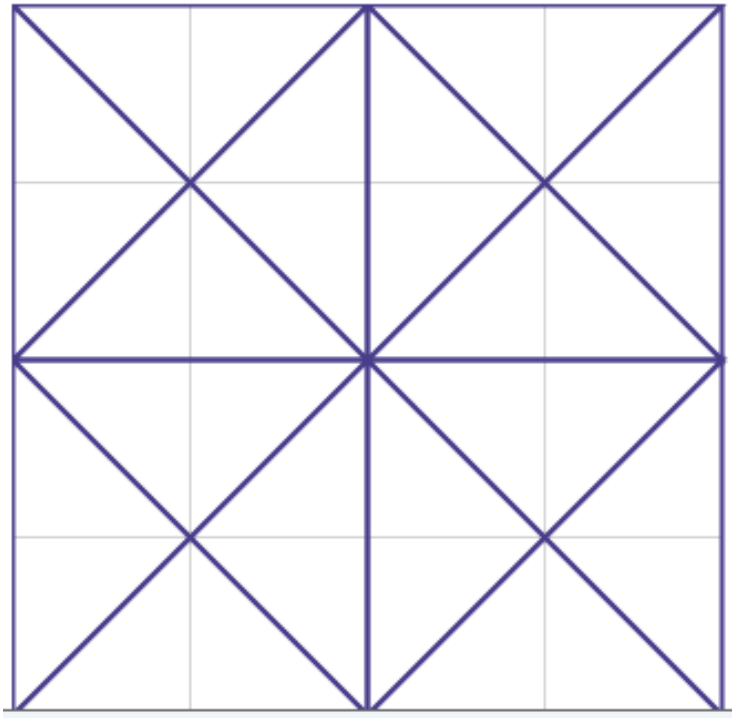


Рисунок 2 – Модель представления геометрической фигуры на основе 16-битного числа

Преимущества такого подхода:

- компактность: любая, даже самая сложная фигура, описывается всего двумя байтами. Это чрезвычайно эффективно для хранения и передачи данных;
- производительность: логические операции над фигурами (сложение, вычитание, пересечение) сводятся к элементарным побитовым операциям (OR, AND, NOT), которые выполняются процессором за один такт. Это на порядки быстрее, чем итерация по массивам полигонов;
- простота: несмотря на кажущуюся сложность, работа с битовыми масками упрощает многие алгоритмы, особенно связанные с логическими операциями.

Для инкапсуляции этих данных был спроектирован простой класс Shape, содержащий два поля.

Листинг 1 — Класс Shape:

```
public class Shape
{
    // Имя фигуры, задаваемое пользователем
    public string Name { get; set; }

    // 16-битные данные, кодирующие геометрию фигуры
    public ushort Data { get; set; }
}
```

Этот класс является основной единицей данных, передаваемой между различными модулями приложения.

2.3. Проектирование ключевых алгоритмов

Эффективность и функциональность приложения напрямую зависят от качества реализованных алгоритмов. В данном проекте можно выделить несколько групп ключевых алгоритмов.

Благодаря выбранной модели данных, реализация логических операций становится тривиальной и элегантной. Все операции вынесены в статический класс ShapeOperations.

Сложение (Add): соответствует побитовой операции «ИЛИ» (|, OR). Результирующий бит равен 1, если хотя бы один из соответствующих битов в операндах равен 1. Это в точности соответствует геометрическому объединению.

Вычитание (Subtract): соответствует операции «Исключающее И» или «И-НЕ» (& ~, AND NOT). Сначала второй операнд инвертируется (~b), что означает "все, что не входит в фигуру B". Затем результат пересекается с фигурой A (a & ...). Таким образом, в результате остаются только те биты (треугольники), которые были в A, но отсутствовали в B.

Пересечение (Multiply): соответствует побитовой операции «И» (&, AND). Результирующий бит равен 1, только если оба соответствующих бита в операндах равны 1. Геометрически это означает, что в итоговую фигуру войдут только общие для A и B треугольники.

Примеры работы этих операций показаны на рисунках 3, 4 и 5.

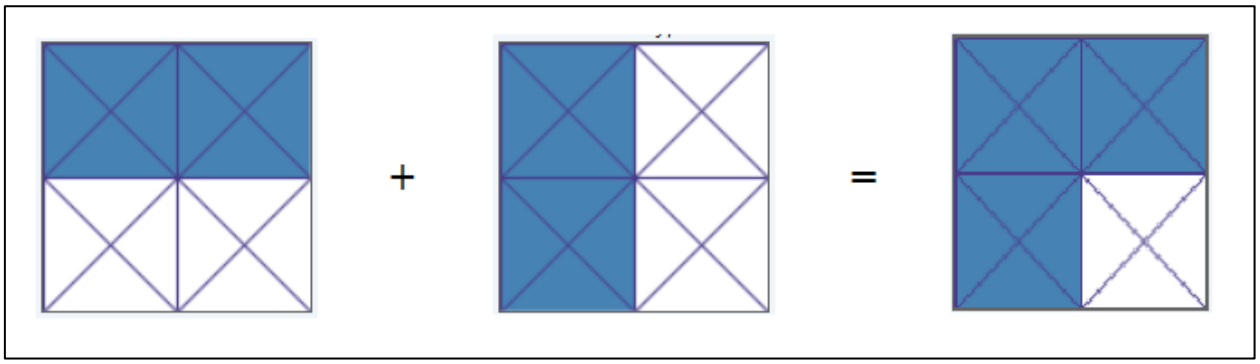


Рисунок 3 – Пример операции сложения (OR)

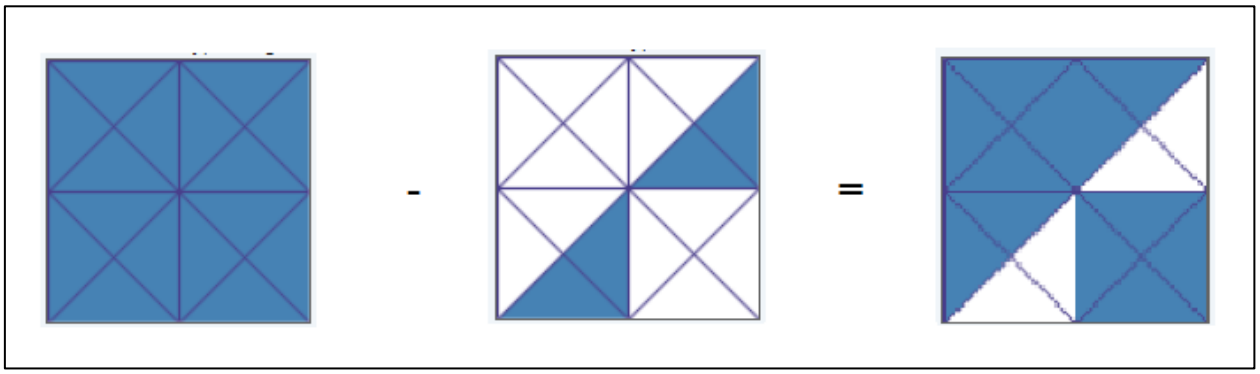


Рисунок 4 – Пример операции вычитания (AND NOT)

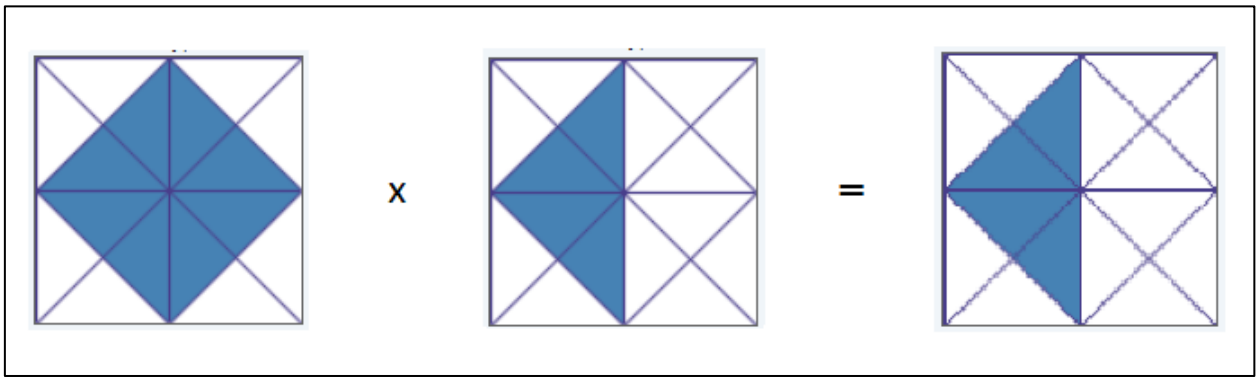


Рисунок 5 – Пример операции пересечения (AND)

В отличие от логических операций, алгоритмы трансформации требуют более сложной логики перестановки битов внутри 16-битного числа. Они также реализованы в классе ShapeOperations.

Алгоритм поворота вправо (RotateRight): задача состоит в том, чтобы перестроить биты так, чтобы фигура повернулась на 90 градусов по часовой стрелке. Алгоритм состоит из двух частей: поворот больших квадратов (квадрантов) и поворот треугольников внутри каждого квадрата.

1. Итерация: алгоритм проходит в цикле по всем 4 квадрантам (от 0 до 3).

2. Определение координат квадранта: для каждого квадранта вычисляются его "координаты" в сетке 2x2: row = bigSquare / 2, col = bigSquare % 2.

3. Вычисление новых координат квадранта: при повороте по часовой стрелке точка (row, col) переходит в (col, 1 - row). Новые координаты newRow и newCol вычисляются по этой формуле.

4. Вычисление нового индекса квадранта: на основе новых координат вычисляется новый линейный индекс квадранта: newBigSquare = newRow * 2 + newCol.

5. Поворот треугольников внутри квадранта: при повороте всего квадранта треугольники внутри него также смещаются на одну позицию по часовой стрелке. Верхний становится правым, правый – нижним, и т.д. Это достигается формулой newTri = (tri + 1) % 4.

6. Перестановка битов: для каждого треугольника tri в исходном квадранте bigSquare проверяется, установлен ли соответствующий бит в исходных данных (data). Если да, то в результирующем числе result устанавливается бит, соответствующий новому положению этого треугольника: в новом квадранте newBigSquare и на новой позиции newTri.

Алгоритм поворота влево (RotateLeft): логика аналогична повороту вправо, но используются другие формулы преобразования координат:

- новые координаты квадранта: точка (row, col) переходит в (1 - col, row);
- новая позиция треугольника: newTri = (tri + 3) % 4, что эквивалентно смещению на одну позицию против часовой стрелки.

Алгоритм зеркального отражения (Mirror): этот алгоритм отражает фигуру относительно центральной вертикальной оси.

1. Итерация: Цикл по 4 квадрантам.

2. Отражение квадрантов: левые квадранты меняются местами с правыми. Это достигается изменением координаты столбца: newCol = 1 - col. Номер строки row остается неизменным.

3. Отражение треугольников внутри квадранта: верхний и нижний треугольники остаются на своих относительных позициях (0 и 2). А левый и правый меняются местами (1 и 3). Это реализуется условной конструкцией.

4. Перестановка битов: аналогично алгоритмам поворота, биты переносятся из старых позиций в новые.

Иллюстрации работы этих алгоритмов представлены на рисунках 6, 7 и 8.

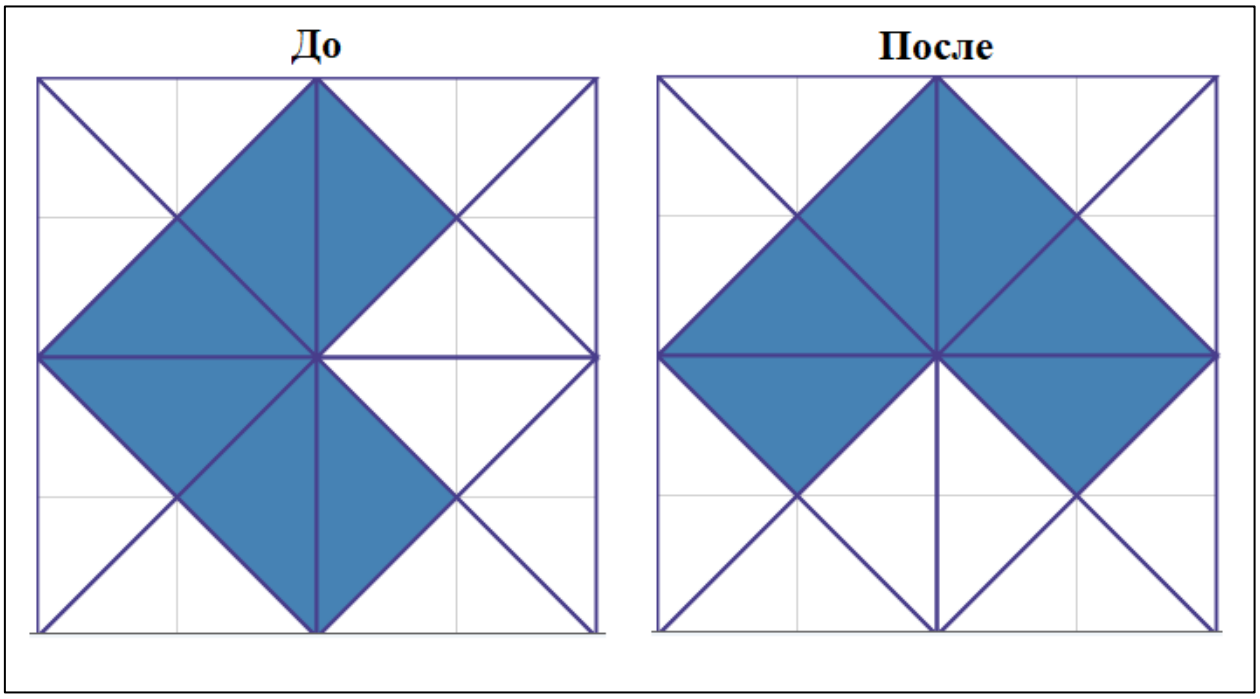


Рисунок 6 – Пример работы алгоритма поворота вправо

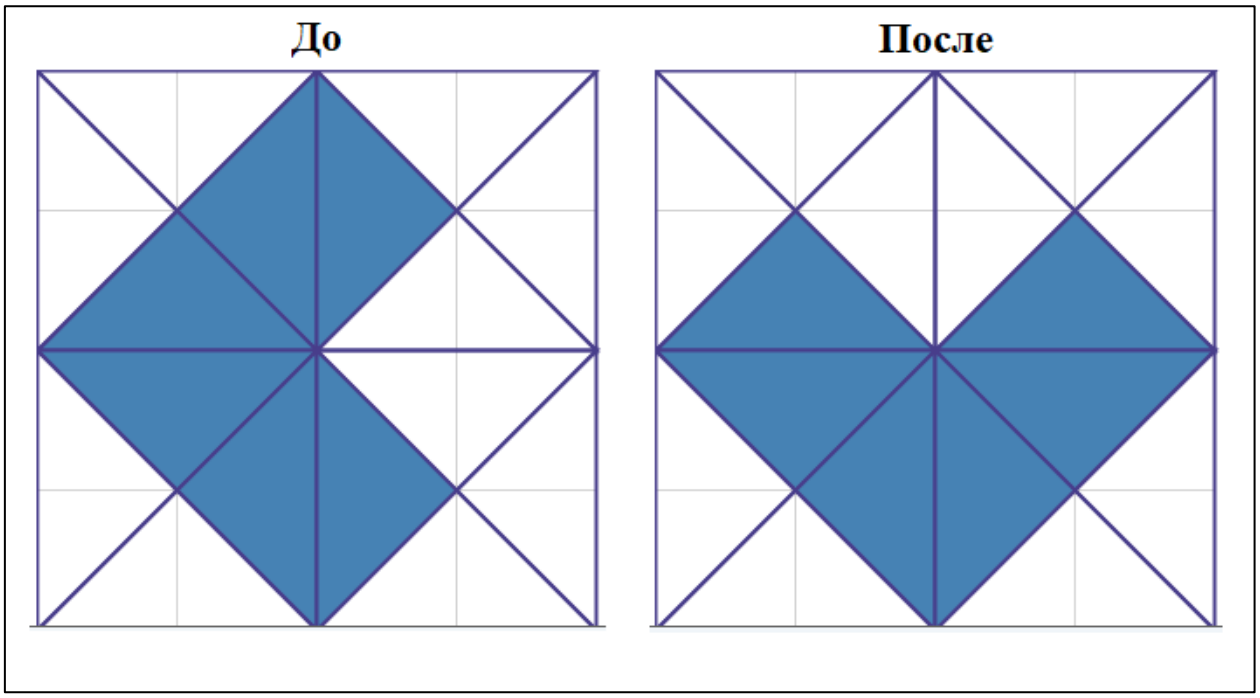


Рисунок 7 – Пример работы алгоритма поворота влево

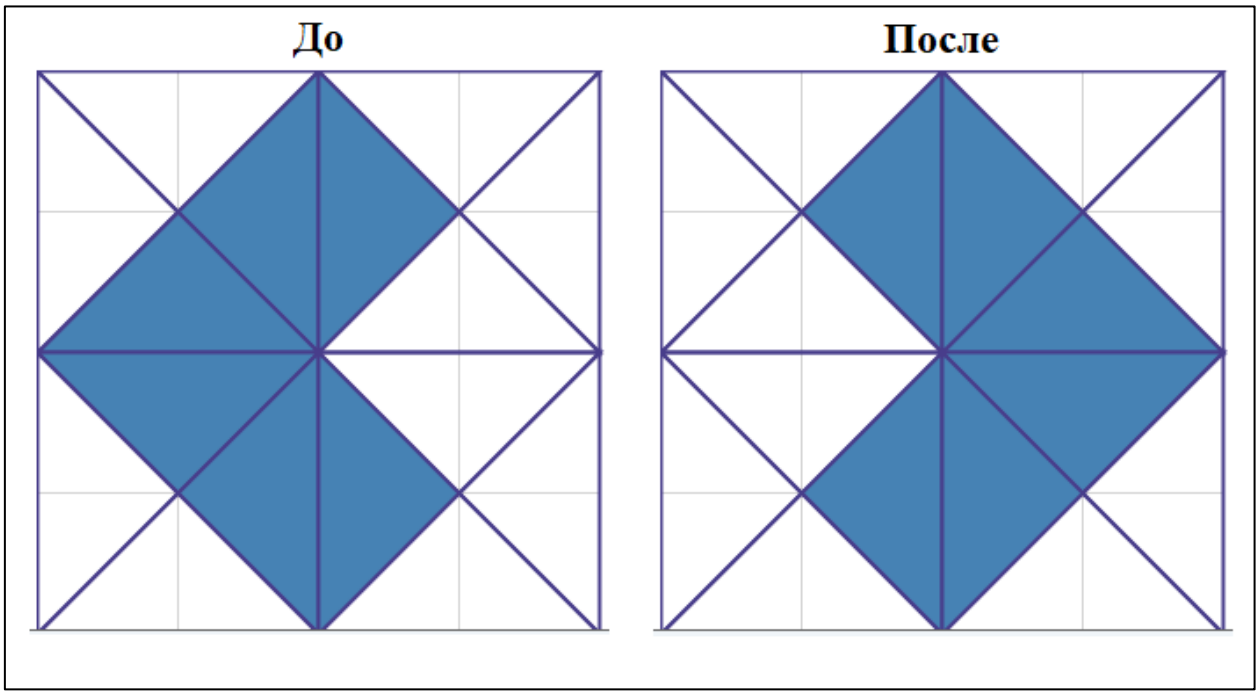


Рисунок 8 – Пример работы алгоритма зеркального отражения

Этот алгоритм отвечает за преобразование абстрактных данных (ushort) в видимое изображение на элементе PictureBox. Он выполняется в обработчике события Paint.

1. Иерархия отрисовки: отрисовка происходит послойно: сначала рисуется фоновая сетка, затем – сама фигура.

2. Отрисовка сетки (DrawGrid): рисуются вертикальные и горизонтальные линии, разделяющие поле на 16 маленьких квадратов, а также диагонали внутри каждого из 4 больших квадрантов, формируя сетку из 16 треугольников.

3. Отрисовка фигуры (DrawBigSquare, DrawTriangle):

- алгоритм итерирует по 4 квадрантам и 4 треугольникам внутри каждого (всего 16 итераций);
- для каждого треугольника вычисляется его индекс (от 0 до 15);
- проверяется соответствующий бит в currentData: (currentData & (1 << index)) != 0;
- если бит равен 1, вычисляются координаты вершин этого треугольника на PictureBox и он закрашивается с помощью метода Graphics.FillPolygon;
- независимо от того, закрашен треугольник или нет, рисуется его контур с помощью Graphics.DrawPolygon для сохранения видимости сетки.

Для обеспечения интерактивности в редакторе (когда пользователь кликает по треугольнику, чтобы изменить его состояние) необходим алгоритм, который по координатам клика определяет, в какой из 16 треугольников попал курсор.

1. Обработчик PictureBox_MouseClick: этот метод срабатывает при клике мыши по PictureBox. Он получает координаты клика e.X и e.Y.

2. Определение квадранта: сначала по глобальным координатам клика определяется, в какой из 4 больших квадрантов попал курсор.

3. Преобразование координат: глобальные координаты клика преобразуются в локальные координаты внутри этого квадранта.

4. Проверка принадлежности треугольнику (IsPointInTriangle): далее необходимо определить, в какой из 4-х треугольников внутри квадранта попала точка. Для этого используется вспомогательный метод IsPointInTriangle. Этот метод принимает координаты точки P и координаты трех вершин треугольника A, B, C. Он работает на основе векторного

произведения. Для точки, лежащей внутри треугольника, она всегда будет находиться "с одной стороны" от каждой из его сторон (AB, BC, CA). Алгоритм вычисляет знаки трех выражений. Если все знаки одинаковы (или некоторые равны нулю), точка находится внутри или на границе треугольника.

5. Изменение данных: последовательно проверяется принадлежность точки каждому из 4-х треугольников квадранта. Как только находится нужный треугольник, вычисляется его глобальный индекс (0-15), и соответствующий бит в currentData инвертируется с помощью операции XOR: currentData ^= (ushort)(1 << triangleIndex).

6. Перерисовка: вызывается метод pictureBox.Invalidate(), который инициирует перерисовку PictureBox с уже измененными данными.

2.4. Проектирование пользовательского интерфейса

Пользовательский интерфейс спроектирован с акцентом на простоту, функциональность и интуитивность. Он состоит из нескольких окон (форм), каждая из которых отвечает за свой участок работы.

MainMenu (главное меню): это стартовое окно приложения. Оно содержит заголовок и две основные кнопки для навигации: «Редактор фигур» и «Играть». Дизайн минималистичен, чтобы не перегружать пользователя. (Рисунок 9).

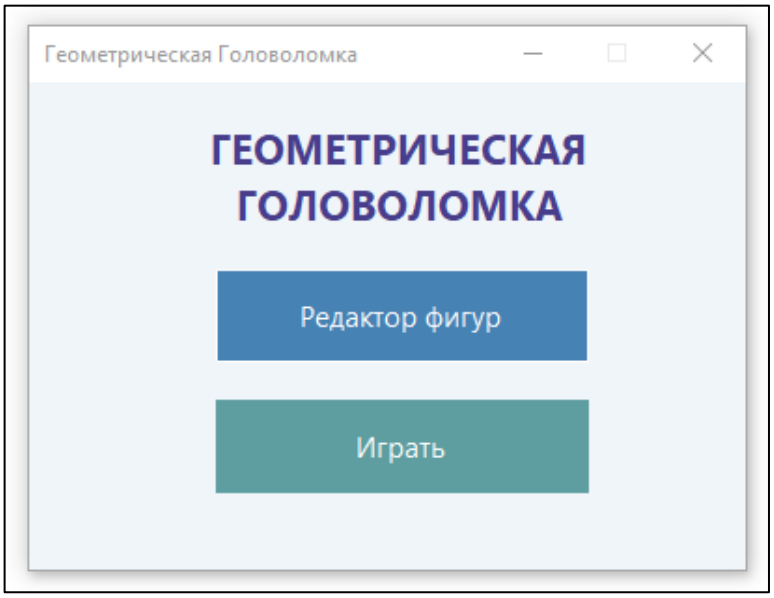


Рисунок 9 – Окно главного меню

ShapeSelectorForm (окно выбора фигур): открывается при нажатии на кнопку «Редактор фигур». Это окно служит каталогом всех сохраненных фигур. Фигуры отображаются в виде списка, где каждый элемент содержит миниатюру фигуры и ее название. Пользователь может выбрать существующую фигуру для редактирования или нажать кнопку «Добавить новую фигуру», чтобы открыть пустой редактор. (Рисунок 10).

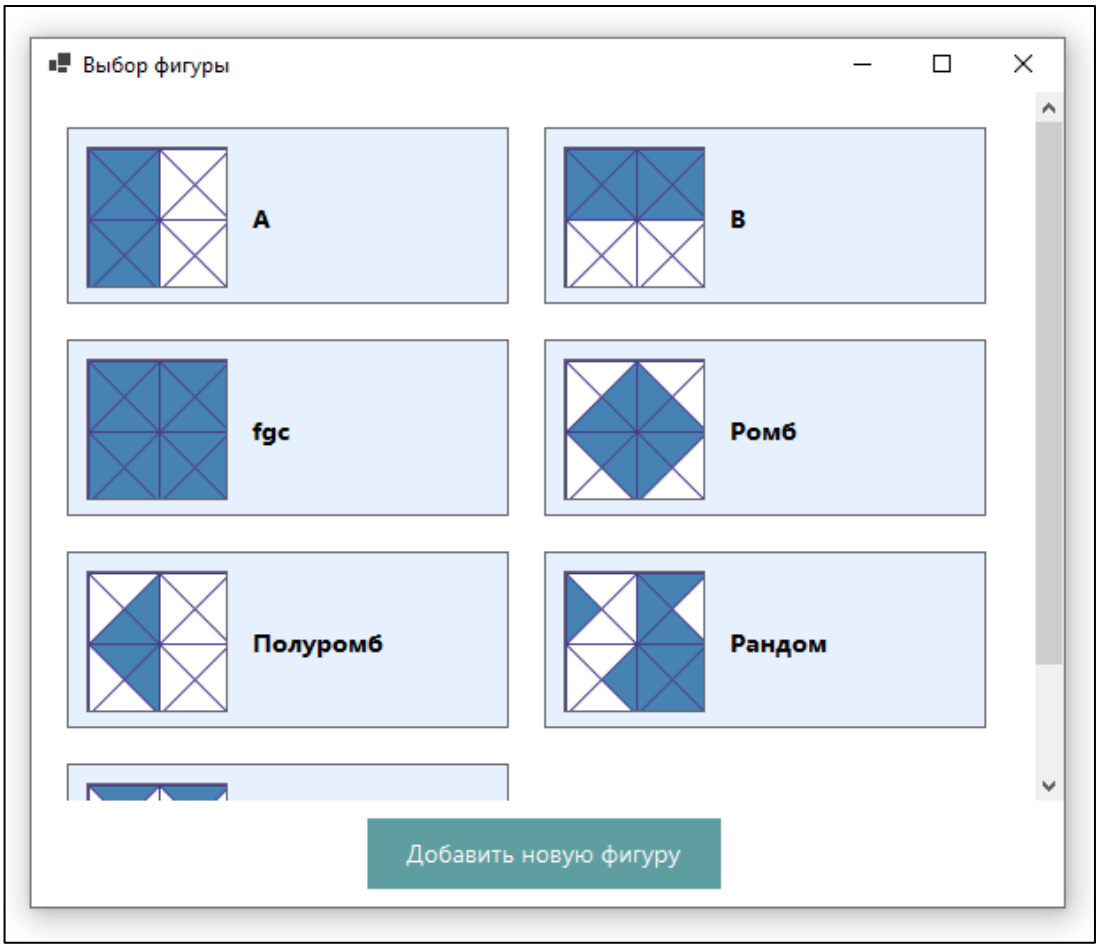


Рисунок 10 – Окно выбора фигур

ShapeEditorForm (окно редактора фигур): Центральный элемент творческого режима. Окно разделено на две части: большая область для рисования (PictureBox) слева и панель управления справа. Панель управления содержит: поле для ввода имени фигуры, кнопки трансформаций («Поворот влево», «Поворот вправо», «Отзеркалить»), кнопку генерации случайной

фигуры и кнопку «Сохранить». Внизу отображается текущее 16-битное представление фигуры. (Рисунок 11).

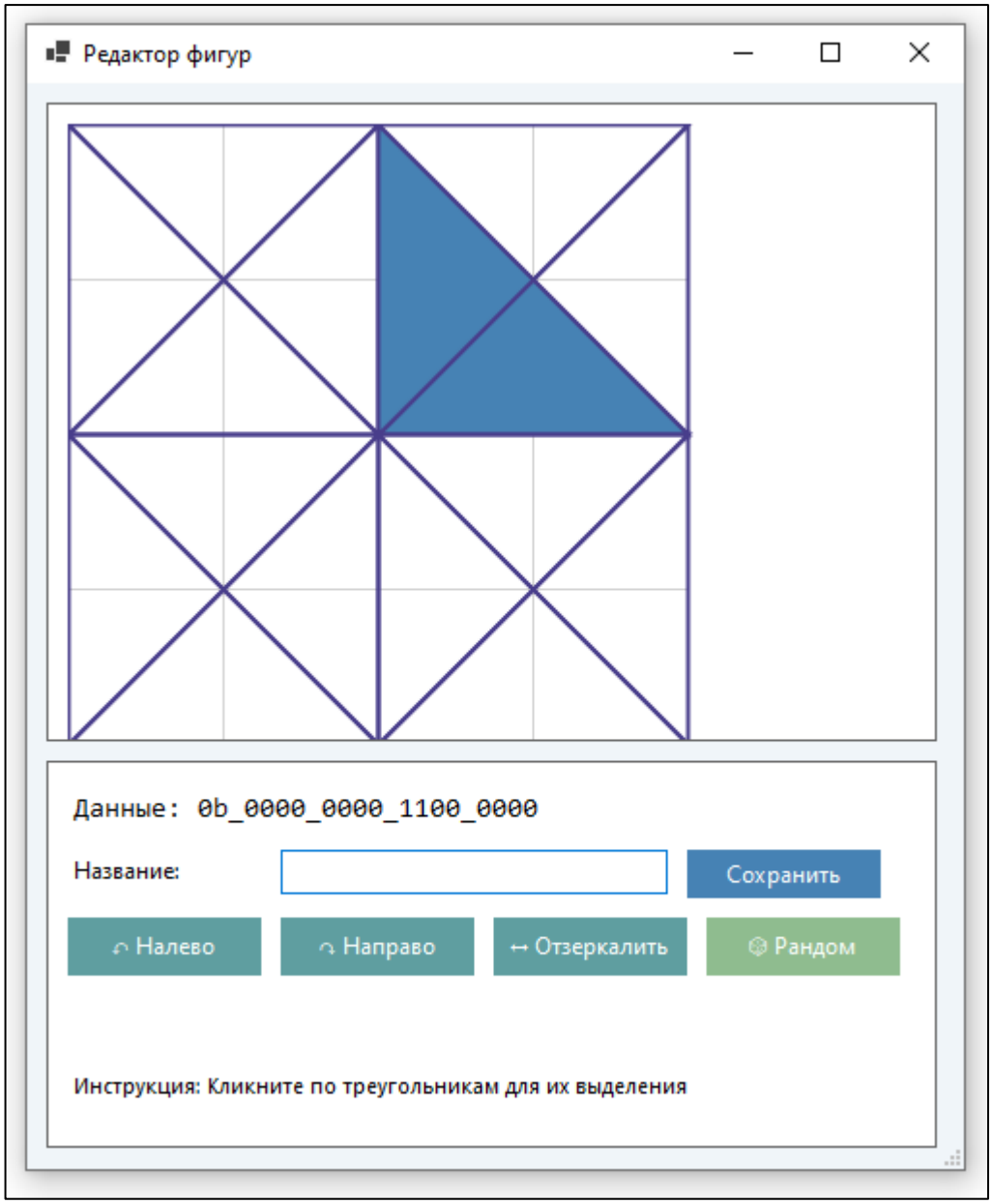


Рисунок 11 – Окно редактора фигур

GameModeSelectionForm (окно выбора режима игры): модальное окно, появляющееся после нажатия кнопки «Играть». Предоставляет пользователю выбор из двух режимов: «Стандартные фигуры» (используются фигуры из файла shapes.txt) и «Случайные фигуры» (фигуры генерируются на лету). (Рисунок 12).

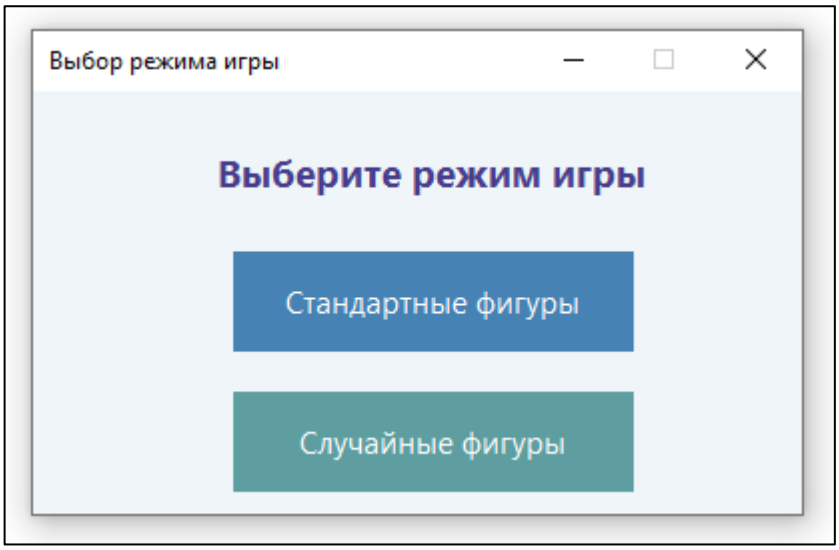


Рисунок 12 – Окно выбора режима игры

ShapeGameForm (основное игровое окно): самое насыщенное по элементам окно. Вверху расположена панель с текущим счетом, выпадающим списком для выбора сложности и кнопкой «Новая игра». Центральная часть разделена на две области: верхняя область задания, где отображаются две исходные фигуры (A и B) и знак операции между ними, и нижняя область с шестью вариантами ответа в виде PictureBox. Пользователь должен кликнуть на один из вариантов. (Рисунок 13).

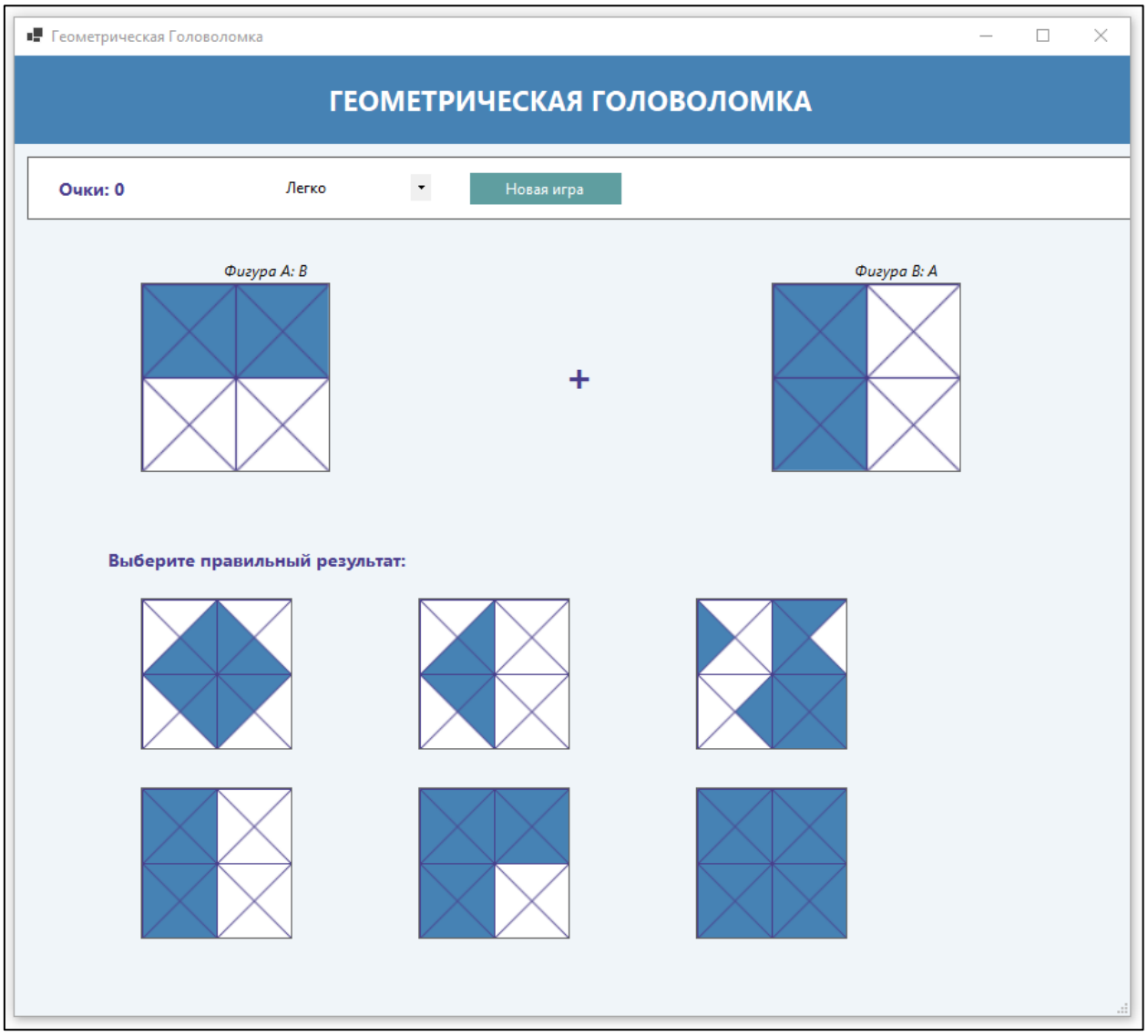


Рисунок 13 – Основное игровое окно

2.5. Проектирование структуры хранения данных

Для сохранения пользовательских фигур был выбран максимально простой и прозрачный формат хранения – обычный текстовый файл shapes.txt. Этот выбор обусловлен следующими причинами:

- простота реализации: работа с текстовыми файлами в C# осуществляется очень просто с помощью классов System.IO.File;
- человекочитаемость: файл можно открыть в любом текстовом редакторе, просмотреть, отредактировать или даже вручную добавить новые фигуры, что удобно для отладки и администрирования;

- отсутствие зависимостей: не требуется установка и настройка систем управления базами данных (СУБД).

Формат файла представляет собой набор строк, где каждая строка описывает одну фигуру по принципу «ключ=значение». Ключ – имя фигуры, заданное пользователем. Значение – 16-битное число (ushort), представляющее данные фигуры, в десятичном формате.

Такой формат легко парсится: строка читается, делится по символу =, первая часть идет в Shape.Name, вторая преобразуется в ushort и записывается в Shape.Data. Схематично структура файла показана на рисунке 14.

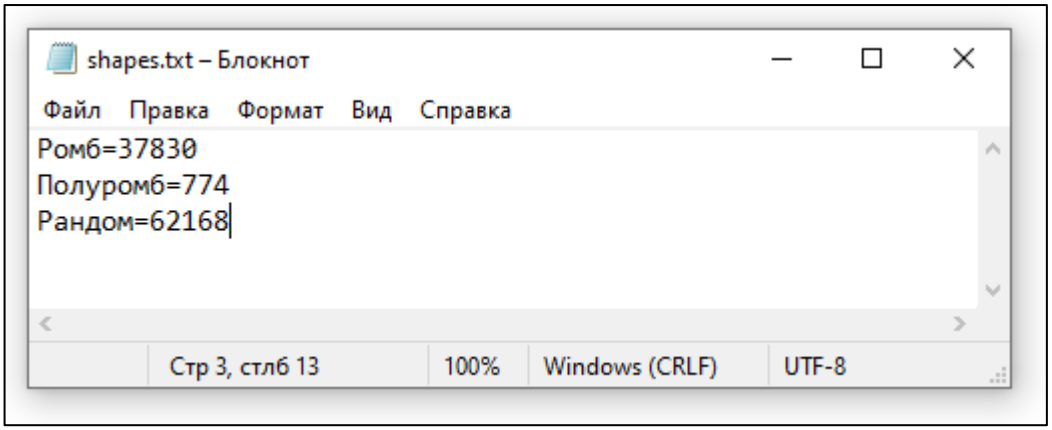


Рисунок 14 – Структура файла хранения данных

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ

Этот раздел посвящен детальному описанию программной реализации спроектированных модулей, классов и алгоритмов. Здесь теоретические концепции из второго раздела находят свое воплощение в коде на языке C#.

3.1. Описание основных классов и модулей

Рассмотрим ключевые классы в каждом из спроектированных пространств имен.

3.1.1. Пространство имен GeometryGame.Core

Это ядро приложения, содержащее фундаментальную логику.

Класс Shape – это простая POCO (Plain Old CLR Object) модель, служащая для хранения данных о фигуре. Как было описано в разделе проектирования, она содержит два автосвойства: public string Name { get; set; } и public ushort Data { get; set; }. Отсутствие логики в этом классе соответствует принципу единой ответственности.

Класс ShapeOperations – это статический класс-утилита, который является сердцем всей алгоритмической части проекта. Он содержит методы для выполнения всех операций над ushort-представлениями фигур:

- Add(ushort a, ushort b), Subtract(ushort a, ushort b), Multiply(ushort a, ushort b): эти три метода являются простой оберткой над соответствующими побитовыми операторами (|, & ~, &), как и было спроектировано. Они возвращают новый ushort с результатом операции;
- RotateRight(ushort data), RotateLeft(ushort data), Mirror(ushort data): в этих методах реализованы более сложные алгоритмы трансформации, основанные на циклах и пересчете индексов битов. Логика их работы в точности соответствует описанию в п. 2.3.2. Они создают и возвращают новое значение ushort, не изменяя исходное;
- GenerateRandomShape(): вспомогательный метод, используемый в игровом режиме и в редакторе. Он создает экземпляр класса Random и в цикле

от 0 до 15 с 50% вероятностью устанавливает каждый бит в 1, генерируя таким образом случайную фигуру.

Класс TriangleHelper – еще один статический класс-утилита, решающий узкоспециализированную задачу. Он содержит метод IsPointInTriangle(PointF p, PointF a, PointF b, PointF c), который реализует алгоритм проверки принадлежности точки треугольнику, описанный в п. 2.3.4. Он вызывает приватный метод Sign для вычисления знака векторного произведения. Этот класс полностью независим и может быть использован в любом другом проекте, где требуется подобная функциональность.

3.1.2. Пространство имен GeometryGame.Editor

Здесь сосредоточена логика и интерфейс редактора фигур.

Класс ShapeSelectorForm – форма, отвечающая за отображение списка сохраненных фигур. Содержит список List<Shape> shapes для хранения загруженных из файла фигур и FlowLayoutPanel flpShapes для их динамического отображения.

Класс содержит следующие методы и обработчики:

- LoadShapes(): при инициализации формы и после каждого сохранения этот метод читает файл shapes.txt построчно, парсит строки и заполняет коллекцию shapes;
- DisplayShapes(): очищает FlowLayoutPanel и для каждой фигуры в коллекции shapes динамически создает панель (Panel), на которой размещает PictureBox с миниатюрой фигуры и Label с ее именем. Миниатюра рисуется в методе DrawShapePreview, который является упрощенной версией алгоритма отрисовки из редактора;
- обработчики событий: клик по любой панели с фигурой открывает ShapeEditorForm для редактирования этой фигуры. Клик по кнопке «Добавить новую» открывает ShapeEditorForm в режиме создания новой фигуры. После закрытия редактора с результатом DialogResult.OK, данные перезагружаются и отображаются заново.

Класс ShapeEditorForm – основная форма редактора.

Класс содержит следующие методы и обработчики:

- InitializeComponents(): программно создает и настраивает все элементы управления на форме: их размеры, положение, цвета, шрифты и привязывает обработчики событий. Это сделано в коде, а не в дизайнере, для большего контроля над внешним видом;
- обработчик PictureBox_Paint(object sender, PaintEventArgs e): главный метод отрисовки. При каждом вызове Invalidate() он полностью перерисовывает содержимое PictureBox на основе текущего значения currentData, вызывая вспомогательные методы DrawGrid, DrawBigSquare, DrawTriangle;
- обработчик PictureBox_MouseClick(object sender, MouseEventArgs e): реализует интерактивность. Выполняет алгоритм определения попадания курсора в треугольник (как описано в п. 2.3.4) и инвертирует соответствующий бит в currentData. После изменения данных вызывает pictureBox.Invalidate() для обновления изображения;
- обработчики кнопок трансформации (btnRotateLeft_Click, и т.д.): эти обработчики (реализованные через лямбда-выражения) вызывают соответствующий метод из ShapeOperations, передавая ему currentData, и присваивают результат обратно в currentData. Затем вызывают Invalidate() для перерисовки;
- обработчик BtnSave_Click(object sender, EventArgs e): Проверяет, введено ли имя фигуры. Если да, вызывает метод SaveShape, который обновляет файл shapes.txt и закрывает форму.

3.1.3. Пространство имен GeometryGame.Game

Модуль, содержащий логику и интерфейс игрового режима.

Класс GameModeSelectionForm – простая форма с двумя кнопками. Обработчики кликов по кнопкам создают и показывают экземпляр ShapeGameForm, передавая в его конструктор булевый флаг useRandomShapes, который определяет режим игры.

Класс ShapeGameForm – наиболее сложная форма приложения, управляющая всем игровым циклом.

Класс содержит следующие методы:

- LoadShapes(): загружает фигуры из файла shapes.txt, если игра идет в стандартном режиме. Если фигур в файле меньше 6, добавляет несколько фигур по умолчанию, чтобы игра всегда была возможна;
- StartNewGame(): это ядро игрового цикла. Он выполняется в начале игры и после каждого ответа. Метод выбирает две случайные фигуры shapeA и shapeB из пула. Затем выбирает случайную операцию (+, -, ×) в зависимости от уровня сложности (GetRandomOperation) и вычисляет правильный результат shapeC, применяя операцию к shapeA.Data и shapeB.Data. Далее отрисовывает shapeA и shapeB в соответствующих PictureBox и формирует список из 6 вариантов ответа: один правильный (shapeC.Data) и 5 случайных неправильных. В конце метод перемешивает список вариантов и отрисовывает их в optionBoxes. Каждому PictureBox в свойство Tag записывается ushort значение фигуры, которую он отображает;
- CheckAnswer(int index): обработчик клика по одному из вариантов ответа. Он сравнивает Tag нажатого PictureBox с currentAnswer. Если они совпадают, начисляет очки, показывает поздравительное сообщение и вызывает StartNewGame() для перехода к следующему раунду. В противном случае снимает очки и показывает сообщение об ошибке;
- методы DrawShape и DrawTriangle: вспомогательные методы для отрисовки миниатюр фигур в PictureBox. Аналогичны методам отрисовки в редакторе, но адаптированы под меньший размер.

3.1.4. Пространство имен GeometryGame.UI

Точка входа и главное меню.

Класс MainMenu – простая форма с двумя кнопками, которые открывают ShapeSelectorForm (для редактора) или GameModeSelectionForm (для игры). Реализует точку навигации по приложению.

Класс Program – стандартный класс, генерируемый Visual Studio. Содержит статический метод Main(), который является точкой входа в приложение. Он инициализирует и запускает главный цикл обработки сообщений Windows для формы MainMenu.

3.2. Диаграммы классов

Для наглядного представления структуры классов и связей между ними используется UML-диаграмма классов. На рисунке 15 представлена упрощенная диаграмма, отражающая ключевые классы и их взаимоотношения.

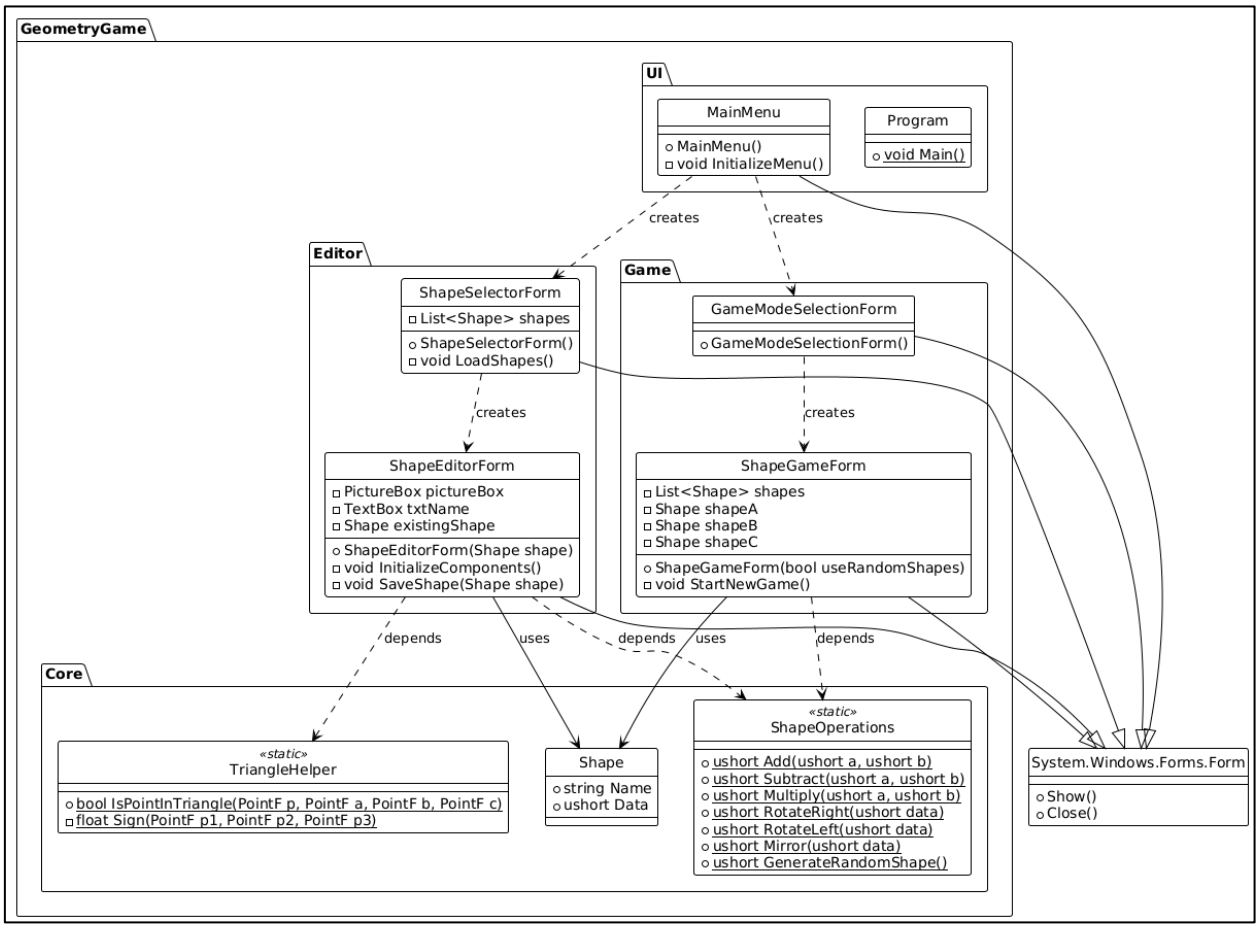


Рисунок 15 – Упрощенная UML-диаграмма классов приложения

Описание связей на диаграмме:

- Наследование: все формы приложения (MainMenu, ShapeEditorForm и т.д.) наследуются от базового класса System.Windows.Forms.Form. Это отношение "является" (is-a);

- Ассоциация/Зависимость: MainMenu создает и отображает экземпляры ShapeSelectorForm и GameModeSelectionForm; ShapeEditorForm и ShapeGameForm используют (ассоциированы с) класс Shape для хранения и обработки данных о фигурах; ShapeEditorForm, ShapeGameForm и другие классы используют статические методы из ShapeOperations и TriangleHelper. Это отношение зависимости (uses).

Эта диаграмма наглядно демонстрирует спроектированную архитектуру: разделение на формы, использование общей модели Shape и общих утилитарных классов с алгоритмами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта была успешно решена задача разработки desktop-приложения «Геометрический тренажер» на языке C# с использованием Windows Forms. Проделанная работа позволила не только создать готовый программный продукт, но и глубоко изучить ключевые аспекты разработки ПО, от анализа и проектирования до реализации и отладки.

В процессе работы были достигнуты все цели и решены все задачи, поставленные во введении. Был проведен детальный анализ предметной области и существующих аналогов, что позволило определить уникальные черты проекта. Был сделан и обоснован выбор технологического стека.

Главным результатом проекта является разработанное приложение, обладающее следующими ключевыми характеристиками:

1) реализована уникальная модель данных на основе 16-битного целого числа (ushort) для компактного и эффективного представления составных геометрических фигур;

2) разработаны и реализованы высокопроизводительные алгоритмы для выполнения логических операций (сложение, вычитание, пересечение) и геометрических трансформаций (поворот, отражение) с использованием побитовых операций;

3) создан полнофункциональный редактор фигур, позволяющий пользователям в интерактивном режиме создавать и модифицировать собственные фигуры, а также сохранять их для дальнейшего использования;

4) реализован увлекательный игровой режим с выбором уровня сложности и возможностью играть как со стандартными, так и со случайно сгенерированными фигурами, что обеспечивает высокую реиграбельность;

5) спроектирована и реализована гибкая модульная архитектура, разделяющая приложение на логические компоненты (ядро, редактор, игра, UI), что упрощает его понимание, поддержку и дальнейшее расширение.

Разработанный программный продукт имеет практическую значимость и может быть использован в образовательных целях для развития логического и пространственного мышления, а также в качестве развлекательного приложения.

Несмотря на полноту реализации поставленных задач, проект имеет значительный потенциал для дальнейшего развития:

- расширение набора операций: можно добавить новые, более сложные трансформации (например, масштабирование, сдвиг) или логические операции (например, XOR – симметрическая разность);
- создание системы уровней: разработать редактор уровней и кампанию с последовательностью усложняющихся задач, а не только случайную генерацию;
- переход на 3D: адаптировать идею побитового кодирования для представления и манипуляции простыми трехмерными объектами (вокселями);
- создание онлайн-версии: разработать веб-приложение или клиент-серверную архитектуру для проведения соревнований между игроками в реальном времени;
- портирование на другие платформы: используя .NET MAUI или другие кроссплатформенные технологии, можно портировать приложение на мобильные устройства (iOS, Android) и другие операционные системы (macOS, Linux).

В целом, курсовой проект можно считать успешно завершенным. Он демонстрирует владение современными технологиями программирования, умение решать нетривиальные алгоритмические задачи и применять принципы программной инженерии на практике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Троелсен, Э. Язык программирования C# 7 и платформы .NET и .NET Core / Э. Троелсен, Ф. Джепикс. – 8-е изд. – СПб.: Диалектика, 2019. – 1328 с.
2. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер. – 4-е изд. – СПб.: Питер, 2017. – 896 с.
3. Windows Forms. Официальная документация Microsoft [Электронный ресурс]. – URL: https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/ (дата обращения: 20.05.2025).
4. Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч [и др.]. – 3-е изд. – М.: Вильямс, 2016. – 720 с.
5. Warren Jr., H. S. Hacker's Delight / H. S. Warren Jr. – 2nd ed. – Addison-Wesley Professional, 2012. – 512 p.
6. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. – 3-е изд. – М.: Вильямс, 2013. – 1328 с.
7. Petzold, C. Programming Windows: The Definitive Guide to the Win32 API / C. Petzold. – 5th ed. – Microsoft Press, 1998. – 1472 p.

ПРИЛОЖЕНИЕ А. Листинг программного кода

Листинг 2 — Класс ShapeOperations:

```
public static class ShapeOperations
{
    public static ushort Add(ushort a, ushort b) => (ushort)(a | b);
     public static ushort Subtract(ushort a, ushort b) => (ushort)(a &
~b);
    public static ushort Multiply(ushort a, ushort b) => (ushort)(a & b);

    public static ushort RotateRight(ushort data)
    {
        ushort result = 0;

        for (int bigSquare = 0; bigSquare < 4; bigSquare++)
        {
            int row = bigSquare / 2;
            int col = bigSquare % 2;

            // Новые координаты после поворота
            int newRow = col;
            int newCol = 1 - row;
            int newBigSquare = newRow * 2 + newCol;

            // Поворот треугольников внутри квадрата
            for (int tri = 0; tri < 4; tri++)
            {
                int oldIndex = bigSquare * 4 + tri;
                int newTri = (tri + 1) % 4;
                int newIndex = newBigSquare * 4 + newTri;

                if ((data & (1 << oldIndex)) != 0)
                {
                    result |= (ushort)(1 << newIndex);
                }
            }
        }

        return result;
    }

    public static ushort RotateLeft(ushort data)
    {
        ushort result = 0;

        for (int bigSquare = 0; bigSquare < 4; bigSquare++)
        {
            int row = bigSquare / 2;
            int col = bigSquare % 2;

            // Новые координаты после поворота
            int newRow = 1 - col;
            int newCol = row;
            int newBigSquare = newRow * 2 + newCol;

            // Поворот треугольников внутри квадрата
            for (int tri = 0; tri < 4; tri++)
            {
                int oldIndex = bigSquare * 4 + tri;
                int newTri = (tri + 3) % 4; // Эквивалент -1 mod 4
```

```
                int newIndex = newBigSquare * 4 + newTri;

                if ((data & (1 << oldIndex)) != 0)
                {
                    result |= (ushort)(1 << newIndex);
                }
            }
        }

        return result;
    }

    public static ushort Mirror(ushort data)
    {
        ushort result = 0;

        for (int bigSquare = 0; bigSquare < 4; bigSquare++)
        {
            int row = bigSquare / 2;
            int col = bigSquare % 2;
            int newCol = 1 - col;
            int newBigSquare = row * 2 + newCol;

            for (int tri = 0; tri < 4; tri++)
            {
                int newTri = tri;
                // Меняем местами правые и левые треугольники
                if (tri == 1) newTri = 3;
                else if (tri == 3) newTri = 1;

                int oldIndex = bigSquare * 4 + tri;
                int newIndex = newBigSquare * 4 + newTri;

                if ((data & (1 << oldIndex)) != 0)
                {
                    result |= (ushort)(1 << newIndex);
                }
            }
        }

        return result;
    }

    public static ushort GenerateRandomShape()
    {
        Random rnd = new Random();
        ushort data = 0;
        for (int i = 0; i < 16; i++)
        {
            if (rnd.Next(2) == 1)
            {
                data |= (ushort)(1 << i);
            }
        }
        return data;
    }
}
```

Листинг 3 — Класс TriangleHelper:

```
public static class TriangleHelper
{
    public static bool IsPointInTriangle(PointF p, PointF a, PointF b,
PointF c)
    {
        float d1 = Sign(p, a, b);
        float d2 = Sign(p, b, c);
        float d3 = Sign(p, c, a);
        bool hasNegative = (d1 < 0) || (d2 < 0) || (d3 < 0);
        bool hasPositive = (d1 > 0) || (d2 > 0) || (d3 > 0);
        return !(hasNegative && hasPositive);
    }

    private static float Sign(PointF p1, PointF p2, PointF p3)
    {
        return (p1.X - p3.X) * (p2.Y - p3.Y) - (p2.X - p3.X) * (p1.Y -
p3.Y);
    }
}
```

Листинг 4 —ShapeEditorForm:

```
public class ShapeEditorForm : Form
{
    private PictureBox pictureBox;
    private Label lblData;
    private TextBox txtName;
    private Button btnSave;
    private ushort currentData;
    private const int SquareSize = 80;
    private Button btnRotateLeft;
    private Button btnRotateRight;
    private Button btnMirror;
    private Button btnRandom;
    private Label lblStatus;

    private Shape existingShape;

    public ShapeEditorForm(Shape shape = null)
    {
        existingShape = shape;
        currentData = shape?.Data ?? 0;
        InitializeComponents();

        if (shape != null)
        {
            txtName.Text = shape.Name;
        }
        UpdateDisplay();
    }

    public ShapeEditorForm()
    {
        InitializeComponents();
        currentData = 0;
        UpdateDisplay();
    }

    private void InitializeComponents()
    {
        this.Size = new Size(500, 600);
```

```
this.Text = "Редактор фигур";
this.BackColor = Color.FromArgb(240, 245, 249);
this.Font = new Font("Segoe UI", 9);

// Панель редактора
Panel editorPanel = new Panel
{
    Location = new Point(10, 10),
    Size = new Size(460, 330),
    BackColor = Color.White,
    BorderStyle = BorderStyle.FixedSingle
};
this.Controls.Add(editorPanel);

pictureBox = new PictureBox
{
    Size = new Size(330, 330),
    Location = new Point(10, 10),
    BackColor = Color.White
};
pictureBox.Paint += PictureBox_Paint;
pictureBox.MouseClick += PictureBox_MouseClick;
editorPanel.Controls.Add(pictureBox);

// Панель управления
Panel controlPanel = new Panel
{
    Location = new Point(10, 350),
    Size = new Size(460, 200),
    BackColor = Color.White,
    BorderStyle = BorderStyle.FixedSingle
};
this.Controls.Add(controlPanel);

lblData = new Label
{
    Location = new Point(10, 15),
    Size = new Size(440, 20),
    Text = "Данные: 0b_0000_0000_0000_0000",
    Font = new Font("Consolas", 10)
};
controlPanel.Controls.Add(lblData);

Label lblName = new Label
{
    Location = new Point(10, 45),
    Size = new Size(100, 20),
    Text = "Название:",
    TextAlign = ContentAlignment.MiddleLeft
};
controlPanel.Controls.Add(lblName);

txtName = new TextBox
{
    Location = new Point(120, 45),
    Size = new Size(200, 25),
    PlaceholderText = "Введите название фигуры"
};
controlPanel.Controls.Add(txtName);

btnSave = new Button
{
    Location = new Point(330, 45),
    Size = new Size(100, 25),
```

```
    Text = "Сохранить",
    BackColor = Color.FromArgb(70, 130, 180),
    ForeColor = Color.White,
    FlatStyle = FlatStyle.Flat
};
btnSave.FlatAppearance.BorderSize = 0;
btnSave.Click += BtnSave_Click;
controlPanel.Controls.Add(btnSave);

// Панель для кнопок операций
Panel operationsPanel = new Panel
{
    Location = new Point(10, 80),
    Size = new Size(440, 40),
    BackColor = Color.Transparent
};
controlPanel.Controls.Add(operationsPanel);

btnRotateLeft = new Button
{
    Location = new Point(0, 0),
    Size = new Size(100, 30),
    Text = "↶ Налево",
    BackColor = Color.FromArgb(95, 158, 160),
    ForeColor = Color.White,
    FlatStyle = FlatStyle.Flat
};
btnRotateLeft.FlatAppearance.BorderSize = 0;
btnRotateLeft.Click += (s, e) =>
{
    currentData = ShapeOperations.RotateLeft(currentData);
    UpdateDisplay();
    pictureBox.Invalidate();
};
operationsPanel.Controls.Add(btnRotateLeft);

btnRotateRight = new Button
{
    Location = new Point(110, 0),
    Size = new Size(100, 30),
    Text = "↷ Направо",
    BackColor = Color.FromArgb(95, 158, 160),
    ForeColor = Color.White,
    FlatStyle = FlatStyle.Flat
};
btnRotateRight.FlatAppearance.BorderSize = 0;
btnRotateRight.Click += (s, e) =>
{
    currentData = ShapeOperations.RotateRight(currentData);
    UpdateDisplay();
    pictureBox.Invalidate();
};
operationsPanel.Controls.Add(btnRotateRight);

btnMirror = new Button
{
    Location = new Point(220, 0),
    Size = new Size(100, 30),
    Text = "↔ Отзеркалить",
    BackColor = Color.FromArgb(95, 158, 160),
    ForeColor = Color.White,
    FlatStyle = FlatStyle.Flat
};
btnMirror.FlatAppearance.BorderSize = 0;
```

```
            btnMirror.Click += (s, e) =>
            {
                currentData = ShapeOperations.Mirror(currentData);
                UpdateDisplay();
                pictureBox.Invalidate();
            };
            operationsPanel.Controls.Add(btnMirror);

            btnRandom = new Button
            {
                Location = new Point(330, 0),
                Size = new Size(100, 30),
                Text = "🎲 Рандом",
                BackColor = Color.FromArgb(143, 188, 143),
                ForeColor = Color.White,
                FlatStyle = FlatStyle.Flat
            };
            btnRandom.FlatAppearance.BorderSize = 0;
            btnRandom.Click += (s, e) =>
            {
                currentData = ShapeOperations.GenerateRandomShape();
                UpdateDisplay();
                pictureBox.Invalidate();
            };
            operationsPanel.Controls.Add(btnRandom);

            lblStatus = new Label
            {
                Location = new Point(10, 130),
                Size = new Size(440, 30),
                ForeColor = Color.Green
            };
            controlPanel.Controls.Add(lblStatus);

            // Инструкция
            Label instruction = new Label
            {
                Location = new Point(10, 160),
                Size = new Size(440, 30),
                    Text = "Инструкция: Кликните по треугольникам для их
выделения",
                Font = new Font("Segoe UI", 8)
            };
            controlPanel.Controls.Add(instruction);
        }

        private void PictureBox_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
            DrawGrid(e.Graphics);
        }

        private void DrawGrid(Graphics g)
        {
            using (Pen gridPen = new Pen(Color.LightGray, 1))
            {
                for (int i = 0; i <= 4; i++)
                {
                    int pos = i * SquareSize;
                    g.DrawLine(gridPen, 0, pos, SquareSize * 4, pos);
                    g.DrawLine(gridPen, pos, 0, pos, SquareSize * 4);
                }
            }
```

```
            for (int row = 0; row < 2; row++)
            {
                for (int col = 0; col < 2; col++)
                {
                    int bigSquareX = col * SquareSize * 2;
                    int bigSquareY = row * SquareSize * 2;
                    DrawBigSquare(g, bigSquareX, bigSquareY, row, col);
                }
            }
        }

        private void DrawBigSquare(Graphics g, int x, int y, int bigRow, int
bigCol)
        {
            PointF center = new PointF(x + SquareSize, y + SquareSize);
              PointF[] topTriangle = { new PointF(x, y), new PointF(x +
SquareSize * 2, y), center };
             PointF[] rightTriangle = { new PointF(x + SquareSize * 2, y), new
PointF(x + SquareSize * 2, y + SquareSize * 2), center };
              PointF[] bottomTriangle = { new PointF(x + SquareSize * 2, y +
SquareSize * 2), new PointF(x, y + SquareSize * 2), center };
            PointF[] leftTriangle = { new PointF(x, y + SquareSize * 2), new
PointF(x, y), center };

            int baseIndex = (bigRow * 2 + bigCol) * 4;

            DrawTriangle(g, topTriangle, (currentData & (1 << baseIndex)) !=
0);
            DrawTriangle(g, rightTriangle, (currentData & (1 << (baseIndex +
1))) != 0);
             DrawTriangle(g, bottomTriangle, (currentData & (1 << (baseIndex
+ 2))) != 0);
             DrawTriangle(g, leftTriangle, (currentData & (1 << (baseIndex +
3))) != 0);
        }

        private void DrawTriangle(Graphics g, PointF[] points, bool filled)
        {
             using (Brush fillBrush = new SolidBrush(Color.FromArgb(70, 130,
180)))
            using (Pen outlinePen = new Pen(Color.DarkSlateBlue, 2))
            {
                if (filled)
                {
                    g.FillPolygon(fillBrush, points);
                }
                g.DrawPolygon(outlinePen, points);
            }
        }

        private void PictureBox_MouseClick(object sender, MouseEventArgs e)
        {
            int x = e.X;
            int y = e.Y;

            int bigCol = x / (SquareSize * 2);
            int bigRow = y / (SquareSize * 2);
            int localX = x % (SquareSize * 2);
            int localY = y % (SquareSize * 2);

            if (bigRow >= 2 || bigCol >= 2) return;

            PointF center = new PointF(SquareSize, SquareSize);
            PointF clickPoint = new PointF(localX, localY);
```

```
            int baseIndex = (bigRow * 2 + bigCol) * 4;
            int triangleIndex = -1;

            if (TriangleHelper.IsPointInTriangle(clickPoint,
                new PointF(0, 0),
                new PointF(SquareSize * 2, 0),
                center))
            {
                triangleIndex = baseIndex;
            }
            else if (TriangleHelper.IsPointInTriangle(clickPoint,
                new PointF(SquareSize * 2, 0),
                new PointF(SquareSize * 2, SquareSize * 2),
                center))
            {
                triangleIndex = baseIndex + 1;
            }
            else if (TriangleHelper.IsPointInTriangle(clickPoint,
                new PointF(SquareSize * 2, SquareSize * 2),
                new PointF(0, SquareSize * 2),
                center))
            {
                triangleIndex = baseIndex + 2;
            }
            else if (TriangleHelper.IsPointInTriangle(clickPoint,
                new PointF(0, SquareSize * 2),
                new PointF(0, 0),
                center))
            {
                triangleIndex = baseIndex + 3;
            }

            if (triangleIndex >= 0)
            {
                currentData ^= (ushort)(1 << triangleIndex);
                UpdateDisplay();
                pictureBox.Invalidate();
            }
        }

        private void UpdateDisplay()
        {
             string binValue = Convert.ToString(currentData, 2).PadLeft(16,
'0');
                    lblData.Text   =   $"Данные:   0b_{binValue.Substring(0,
4)}_{binValue.Substring(4,                          4)}_{binValue.Substring(8,
4)}_{binValue.Substring(12, 4)}";
        }

        private void BtnSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrWhiteSpace(txtName.Text))
            {
                MessageBox.Show("Введите название фигуры", "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            SaveShape(new Shape { Name = txtName.Text, Data = currentData });
            lblStatus.Text = $"Фигура '{txtName.Text}' сохранена!";
        }

        private void SaveShape(Shape shape)
```

```
        {
            try
            {
                List<string> lines = new List<string>();
                string filePath = "shapes.txt";

                if (File.Exists(filePath))
                {
                    lines = File.ReadAllLines(filePath).ToList();
                }

               // Если редактируем существующую фигуру, удаляем старую запись
                if (existingShape != null)
                {
                   lines = lines.Where(l => !l.StartsWith(existingShape.Name
+ "=")).ToList();
                }

                // Добавляем новую запись
                lines.Add($"{shape.Name}={shape.Data}");

                File.WriteAllLines(filePath, lines);

                // Закрываем форму после успешного сохранения
                this.DialogResult = DialogResult.OK;
                this.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка сохранения: {ex.Message}", "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
```

Листинг 4 — Класс ShapeSelectorForm:

```
    public class ShapeSelectorForm : Form
    {
        private FlowLayoutPanel flpShapes;
        private Button btnAddNew;
        private List<Shape> shapes = new List<Shape>();

        public ShapeSelectorForm()
        {
            InitializeComponents();
            LoadShapes();
            DisplayShapes();
        }

        private void InitializeComponents()
        {
            this.Text = "Выбор фигуры";
            this.Size = new Size(600, 500);
            this.BackColor = Color.FromArgb(240, 245, 249);

            flpShapes = new FlowLayoutPanel
            {
                Dock = DockStyle.Fill,
                AutoScroll = true,
                Padding = new Padding(10),
                BackColor = Color.White
```

```
            };
            this.Controls.Add(flpShapes);

            Panel bottomPanel = new Panel
            {
                Dock = DockStyle.Bottom,
                Height = 60,
                BackColor = Color.White
            };
            this.Controls.Add(bottomPanel);

            btnAddNew = new Button
            {
                Text = "Добавить новую фигуру",
                Size = new Size(200, 40),
                Location = new Point(190, 10),
                BackColor = Color.FromArgb(95, 158, 160),
                ForeColor = Color.White,
                FlatStyle = FlatStyle.Flat,
                Font = new Font("Segoe UI", 10)
            };
            btnAddNew.FlatAppearance.BorderSize = 0;
            btnAddNew.Click += (s, e) =>
            {
                var editor = new ShapeEditorForm();
                if (editor.ShowDialog() == DialogResult.OK)
                {
                    LoadShapes();
                    DisplayShapes();
                }
            };
            bottomPanel.Controls.Add(btnAddNew);
        }

        private void LoadShapes()
        {
            shapes.Clear();
            if (File.Exists("shapes.txt"))
            {
                foreach (string line in File.ReadLines("shapes.txt"))
                {
                    string[] parts = line.Split('=');
                     if (parts.Length == 2 && ushort.TryParse(parts[1], out
ushort data))
                    {
                        shapes.Add(new Shape { Name = parts[0], Data = data
});
                    }
                }
            }
        }

        private void DisplayShapes()
        {
            flpShapes.Controls.Clear();

            foreach (var shape in shapes)
            {
                Panel shapePanel = new Panel
                {
                    Size = new Size(250, 100),
                    Margin = new Padding(10),
                    BackColor = Color.FromArgb(230, 240, 255),
                    BorderStyle = BorderStyle.FixedSingle,
```

```
            Cursor = Cursors.Hand,
            Tag = shape
        };

        PictureBox pb = new PictureBox
        {
            Size = new Size(80, 80),
            Location = new Point(10, 10),
            BackColor = Color.White,
            BorderStyle = BorderStyle.FixedSingle,
            Cursor = Cursors.Hand
        };
        DrawShapePreview(pb, shape.Data);
        shapePanel.Controls.Add(pb);

        Label lblName = new Label
        {
            Text = shape.Name,
            Location = new Point(100, 35),
            Size = new Size(140, 30),
            Font = new Font("Segoe UI", 10, FontStyle.Bold),
            TextAlign = ContentAlignment.MiddleLeft,
            Cursor = Cursors.Hand
        };
        shapePanel.Controls.Add(lblName);

        // Обработчик для всей панели
        shapePanel.Click += (s, e) => OpenShapeEditor(shape);

        // Обработчики для дочерних элементов
        pb.Click += (s, e) => OpenShapeEditor(shape);
        lblName.Click += (s, e) => OpenShapeEditor(shape);

        flpShapes.Controls.Add(shapePanel);
    }
}

private void OpenShapeEditor(Shape shape)
{
    var editor = new ShapeEditorForm(shape);
    if (editor.ShowDialog() == DialogResult.OK)
    {
        LoadShapes();
        DisplayShapes();
    }
}

private void DrawShapePreview(PictureBox pb, ushort data)
{
    Bitmap bmp = new Bitmap(pb.Width, pb.Height);
    using (Graphics g = Graphics.FromImage(bmp))
    {
        g.Clear(Color.White);
        g.SmoothingMode = SmoothingMode.AntiAlias;

        for (int row = 0; row < 2; row++)
        {
            for (int col = 0; col < 2; col++)
            {
                int x = col * 40;
                int y = row * 40;
                PointF center = new PointF(x + 20, y + 20);
```

```
                        PointF[] top = { new PointF(x, y), new PointF(x + 40,
y), center };
                        PointF[] right = { new PointF(x + 40, y), new PointF(x
+ 40, y + 40), center };
                         PointF[] bottom = { new PointF(x + 40, y + 40), new
PointF(x, y + 40), center };
                        PointF[] left = { new PointF(x, y + 40), new PointF(x,
y), center };

                         int baseIndex = (row * 2 + col) * 4;

                        DrawTriangle(g, top, (data & (1 << baseIndex)) != 0);
                          DrawTriangle(g, right, (data & (1 << (baseIndex +
1))) != 0);
                          DrawTriangle(g, bottom, (data & (1 << (baseIndex +
2))) != 0);
                        DrawTriangle(g, left, (data & (1 << (baseIndex + 3)))
!= 0);
                    }
                }
            }
            pb.Image = bmp;
        }

        private void DrawTriangle(Graphics g, PointF[] points, bool filled)
        {
            using (Brush fillBrush = new SolidBrush(Color.FromArgb(70, 130,
180)))
            using (Pen outlinePen = new Pen(Color.DarkSlateBlue, 1))
            {
                if (filled) g.FillPolygon(fillBrush, points);
                g.DrawPolygon(outlinePen, points);
            }
        }
    }
```

Листинг 5 — Класс GameModeSelectionForm:

```
public class GameModeSelectionForm : Form
{
    public GameModeSelectionForm()
    {
        InitializeComponents();
    }

    private void InitializeComponents()
    {
        this.Text = "Выбор режима игры";
        this.Size = new Size(400, 250);
        this.BackColor = Color.FromArgb(240, 245, 249);
        this.FormBorderStyle = FormBorderStyle.FixedDialog;
        this.StartPosition = FormStartPosition.CenterScreen;
        this.MaximizeBox = false;

        Label lblTitle = new Label
        {
            Text = "Выберите режим игры",
            Font = new Font("Segoe UI", 14, FontStyle.Bold),
            ForeColor = Color.DarkSlateBlue,
            Size = new Size(380, 40),
            Location = new Point(10, 20),
            TextAlign = ContentAlignment.MiddleCenter
```

```
        };
        this.Controls.Add(lblTitle);

        Button btnStandard = new Button
        {
            Text = "Стандартные фигуры",
            Location = new Point(100, 80),
            Size = new Size(200, 50),
            BackColor = Color.FromArgb(70, 130, 180),
            ForeColor = Color.White,
            FlatStyle = FlatStyle.Flat,
            Font = new Font("Segoe UI", 11)
        };
        btnStandard.FlatAppearance.BorderSize = 0;
        btnStandard.Click += (s, e) =>
        {
            this.Hide();
            new ShapeGameForm(false).ShowDialog();
            this.Close();
        };

        Button btnRandom = new Button
        {
            Text = "Случайные фигуры",
            Location = new Point(100, 150),
            Size = new Size(200, 50),
            BackColor = Color.FromArgb(95, 158, 160),
            ForeColor = Color.White,
            FlatStyle = FlatStyle.Flat,
            Font = new Font("Segoe UI", 11)
        };
        btnRandom.FlatAppearance.BorderSize = 0;
        btnRandom.Click += (s, e) =>
        {
            this.Hide();
            new ShapeGameForm(true).ShowDialog();
            this.Close();
        };

        this.Controls.Add(btnStandard);
        this.Controls.Add(btnRandom);
    }
}
```

Листинг 6 — Класс ShapeGameForm:

```
public class ShapeGameForm : Form
{
    private List<Shape> shapes = new List<Shape>();
    private Shape shapeA, shapeB, shapeC;
    private ushort currentAnswer;
    private int score = 0;
    private Label lblScore;
    private PictureBox pbShapeA, pbShapeB;
    private PictureBox[] optionBoxes = new PictureBox[6];
    private Button btnNewGame;
    private ComboBox cmbDifficulty;
    private Random random = new Random();
    private bool useRandomShapes;

    private Label lblShapeA;
    private Label lblShapeB;
```

```
private Label lblOperation;
private Label lblTitle;
private Panel titlePanel;

public ShapeGameForm(bool useRandomShapes)
{
    this.useRandomShapes = useRandomShapes;
    LoadShapes();
    InitializeComponents();
    StartNewGame();
}

private void InitializeComponents()
{
    this.Size = new Size(900, 800);
    this.Text = "Геометрическая Головоломка";
    this.BackColor = Color.FromArgb(240, 245, 249);
    this.Font = new Font("Segoe UI", 9);

    // Панель заголовка
    titlePanel = new Panel
    {
        Dock = DockStyle.Top,
        Height = 70,
        BackColor = Color.FromArgb(70, 130, 180)
    };
    this.Controls.Add(titlePanel);

    lblTitle = new Label
    {
        Text = "ГЕОМЕТРИЧЕСКАЯ ГОЛОВОЛОМКА",
        Dock = DockStyle.Fill,
        Font = new Font("Segoe UI", 16, FontStyle.Bold),
        ForeColor = Color.White,
        TextAlign = ContentAlignment.MiddleCenter
    };
    titlePanel.Controls.Add(lblTitle);

    // Панель управления
    Panel controlPanel = new Panel
    {
        Location = new Point(10, 80),
        Size = new Size(880, 50),
        BackColor = Color.White,
        BorderStyle = BorderStyle.FixedSingle
    };
    this.Controls.Add(controlPanel);

    lblScore = new Label
    {
        Location = new Point(20, 15),
        Size = new Size(150, 20),
        Text = $"Очки: {score}",
        Font = new Font("Segoe UI", 10, FontStyle.Bold),
        ForeColor = Color.DarkSlateBlue
    };
    controlPanel.Controls.Add(lblScore);

    cmbDifficulty = new ComboBox
    {
        Location = new Point(200, 12),
        Size = new Size(120, 25),
        DropDownStyle = ComboBoxStyle.DropDownList,
        FlatStyle = FlatStyle.Flat
```

```
            };
             cmbDifficulty.Items.AddRange(new object[] { "Легко", "Средне",
"Сложно" });
            cmbDifficulty.SelectedIndex = 0;
            cmbDifficulty.SelectedIndexChanged += (s, e) => StartNewGame();
            controlPanel.Controls.Add(cmbDifficulty);

            btnNewGame = new Button
            {
                Location = new Point(350, 12),
                Size = new Size(120, 25),
                Text = "Новая игра",
                BackColor = Color.FromArgb(95, 158, 160),
                ForeColor = Color.White,
                FlatStyle = FlatStyle.Flat
            };
            btnNewGame.FlatAppearance.BorderSize = 0;
            btnNewGame.Click += (s, e) => StartNewGame();
            controlPanel.Controls.Add(btnNewGame);

            // Область фигур
            Panel shapesPanel = new Panel
            {
                Location = new Point(50, 150),
                Size = new Size(800, 200),
                BackColor = Color.Transparent
            };
            this.Controls.Add(shapesPanel);

            lblShapeA = new Label
            {
                Location = new Point(50, 10),
                Size = new Size(200, 20),
                TextAlign = ContentAlignment.MiddleCenter,
                Font = new Font("Segoe UI", 9, FontStyle.Italic)
            };
            shapesPanel.Controls.Add(lblShapeA);

            pbShapeA = new PictureBox
            {
                Size = new Size(150, 150),
                Location = new Point(50, 30),
                BackColor = Color.White,
                BorderStyle = BorderStyle.FixedSingle,
                SizeMode = PictureBoxSizeMode.StretchImage
            };
            shapesPanel.Controls.Add(pbShapeA);

            lblOperation = new Label
            {
                Location = new Point(350, 80),
                Size = new Size(100, 40),
                Text = "+",
                Font = new Font("Segoe UI", 24, FontStyle.Bold),
                ForeColor = Color.DarkSlateBlue,
                TextAlign = ContentAlignment.MiddleCenter
            };
            shapesPanel.Controls.Add(lblOperation);

            lblShapeB = new Label
            {
                Location = new Point(550, 10),
                Size = new Size(200, 20),
                TextAlign = ContentAlignment.MiddleCenter,
```

```
                Font = new Font("Segoe UI", 9, FontStyle.Italic)
            };
            shapesPanel.Controls.Add(lblShapeB);

            pbShapeB = new PictureBox
            {
                Size = new Size(150, 150),
                Location = new Point(550, 30),
                BackColor = Color.White,
                BorderStyle = BorderStyle.FixedSingle,
                SizeMode = PictureBoxSizeMode.StretchImage
            };
            shapesPanel.Controls.Add(pbShapeB);

            // Область вариантов
            Panel optionsPanel = new Panel
            {
                Location = new Point(50, 380),
                Size = new Size(800, 350),
                BackColor = Color.Transparent
            };
            this.Controls.Add(optionsPanel);

            Label lblOptions = new Label
            {
                Text = "Выберите правильный результат:",
                Location = new Point(20, 10),
                Size = new Size(300, 20),
                Font = new Font("Segoe UI", 10, FontStyle.Bold),
                ForeColor = Color.DarkSlateBlue
            };
            optionsPanel.Controls.Add(lblOptions);

            // Создаем 6 вариантов ответа (2 ряда по 3)
            for (int i = 0; i < 6; i++)
            {
                int row = i / 3;
                int col = i % 3;
                optionBoxes[i] = new PictureBox
                {
                    Size = new Size(120, 120),
                    Location = new Point(50 + col * 220, 50 + row * 150),
                    BackColor = Color.White,
                    BorderStyle = BorderStyle.FixedSingle,
                    SizeMode = PictureBoxSizeMode.StretchImage,
                    Cursor = Cursors.Hand
                };
                int index = i;
               // optionBoxes[i].MouseClick += (s, ev) => CheckAnswer(index);
                // optionBoxes[i].MouseEnter += (s, ev) =>
                  //      optionBoxes[index].BackColor = Color.FromArgb(230,
240, 255);
                // optionBoxes[i].MouseLeave += (s, ev) =>
                //     optionBoxes[index].BackColor = Color.White;
                // optionsPanel.Controls.Add(optionBoxes[i]);
                optionBoxes[i].MouseEnter += (s, ev) =>
                {
                     optionBoxes[index].BackColor = Color.FromArgb(230, 240,
255);
                   optionBoxes[index].Paint += DrawCustomBorder; // Добавляем
отрисовку
                    optionBoxes[index].Invalidate();
                };
```

```
                optionBoxes[i].MouseLeave += (s, ev) =>
                {
                    optionBoxes[index].BackColor = Color.White;
                    optionBoxes[index].Paint -= DrawCustomBorder; // Убираем
отрисовку
                    optionBoxes[index].Invalidate();
                };

                optionBoxes[i].MouseClick += (s, ev) => CheckAnswer(index);
                optionsPanel.Controls.Add(optionBoxes[i]);
                }
        }

        private void DrawCustomBorder(object sender, PaintEventArgs e)
        {
            var pb = (PictureBox)sender;
            using (Pen pen = new Pen(Color.Yellow, 4))
            {
                e.Graphics.DrawRectangle(pen, 0, 0, pb.Width-3, pb.Height-3);
            }
        }

        private void LoadShapes()
        {
            if (!useRandomShapes)
            {
                // Загрузка стандартных фигур из файла
                if (File.Exists("shapes.txt"))
                {
                    foreach (string line in File.ReadLines("shapes.txt"))
                    {
                        string[] parts = line.Split('=');
                          if (parts.Length == 2 && ushort.TryParse(parts[1],
out ushort data))
                        {
                              shapes.Add(new Shape { Name = parts[0], Data =
data });
                        }
                    }
                }

                // Добавление фигур по умолчанию, если недостаточно
                if (shapes.Count < 6)
                {
                    shapes.AddRange(new[] {
                                  new Shape { Name = "Квадрат", Data =
0b1111_1111_0000_0000 },
                                new Shape { Name = "Треугольник", Data =
0b1000_0100_0010_0001 },
                          new Shape { Name = "Ромб", Data = 0b0000_0110_0110_0000
},
                                  new Shape { Name = "Шахматы", Data =
0b1010_0101_1010_0101 },
                                new Shape { Name = "Половина", Data =
0b1111_0000_1111_0000 },
                          new Shape { Name = "Крест", Data = 0b1001_0110_0110_1001
}
                    });
                }
            }
            else
            {
                // Генерация случайных фигур
                for (int i = 0; i < 6; i++)
```

```
                {
                    ushort randomData = (ushort)random.Next(0, ushort.MaxValue);
                    shapes.Add(new Shape { Name = $"Случайная {i + 1}", Data = randomData });
                }
            }
        }

        private void StartNewGame()
        {
            if (useRandomShapes)
            {
                // Обновление случайных фигур для новой игры
                shapes.Clear();
                for (int i = 0; i < 6; i++)
                {
                    ushort randomData = (ushort)random.Next(0, ushort.MaxValue);
                    shapes.Add(new Shape { Name = $"Случайная {i+1}", Data = randomData });
                }
            }

            shapeA = shapes[random.Next(shapes.Count)];
            shapeB = shapes[random.Next(shapes.Count)];

            lblShapeA.Text = $"Фигура A: {shapeA.Name}";
            lblShapeB.Text = $"Фигура B: {shapeB.Name}";

            string operation = GetRandomOperation();
            lblOperation.Text = operation;

            switch (operation)
            {
                case "+":
                    shapeC = new Shape { Data = ShapeOperations.Add(shapeA.Data, shapeB.Data) };
                    break;
                case "-":
                    shapeC = new Shape { Data = ShapeOperations.Subtract(shapeA.Data, shapeB.Data) };
                    break;
                case "×":
                    shapeC = new Shape { Data = ShapeOperations.Multiply(shapeA.Data, shapeB.Data) };
                    break;
            }

            DrawShape(pbShapeA, shapeA.Data);
            DrawShape(pbShapeB, shapeB.Data);

            var options = new List<ushort>();
            options.Add(shapeC.Data);
            while (options.Count < 6)
            {
                ushort randomData = shapes[random.Next(shapes.Count)].Data;
                if (!options.Contains(randomData)) options.Add(randomData);
            }

            options = options.OrderBy(x => random.Next()).ToList();
            currentAnswer = shapeC.Data;

            for (int i = 0; i < 6; i++)
```

```
            {
                DrawShape(optionBoxes[i], options[i]);
                optionBoxes[i].Tag = options[i];
            }
        }

        private string GetRandomOperation()
        {
            string difficulty = cmbDifficulty.SelectedItem.ToString();
            return difficulty switch
            {
                "Легко" => "+",
                "Средне" => random.Next(2) == 0 ? "+" : "-",
                "Сложно" => random.Next(3) switch { 0 => "+", 1 => "-", _ =>
"×" },
                _ => "+"
            };
        }

        private void DrawShape(PictureBox pb, ushort data)
        {
            Bitmap bmp = new Bitmap(120, 120);
            using (Graphics g = Graphics.FromImage(bmp))
            {
                g.Clear(Color.White);
                g.SmoothingMode = SmoothingMode.AntiAlias;

                for (int row = 0; row < 2; row++)
                {
                    for (int col = 0; col < 2; col++)
                    {
                        int x = col * 60;
                        int y = row * 60;
                        PointF center = new PointF(x + 30, y + 30);

                        PointF[] top = { new PointF(x, y), new PointF(x + 60,
y), center };
                       PointF[] right = { new PointF(x + 60, y), new PointF(x
+ 60, y + 60), center };
                         PointF[] bottom = { new PointF(x + 60, y + 60), new
PointF(x, y + 60), center };
                       PointF[] left = { new PointF(x, y + 60), new PointF(x,
y), center };

                        int baseIndex = (row * 2 + col) * 4;

                        DrawTriangle(g, top, (data & (1 << baseIndex)) != 0);
                          DrawTriangle(g, right, (data & (1 << (baseIndex +
1))) != 0);
                         DrawTriangle(g, bottom, (data & (1 << (baseIndex +
2))) != 0);
                        DrawTriangle(g, left, (data & (1 << (baseIndex + 3)))
!= 0);
                    }
                }
            }
            pb.Image = bmp;
        }

        private void DrawTriangle(Graphics g, PointF[] points, bool filled)
        {
            using (Brush fillBrush = new SolidBrush(Color.FromArgb(70, 130,
180)))
            using (Pen outlinePen = new Pen(Color.DarkSlateBlue, 1.5f))
```

```
        {
            if (filled) g.FillPolygon(fillBrush, points);
            g.DrawPolygon(outlinePen, points);
        }
    }

    private void CheckAnswer(int index)
    {
        if ((ushort)optionBoxes[index].Tag == currentAnswer)
        {
            score += 10;
            lblScore.Text = $"Очки: {score}";
            optionBoxes[index].BackColor = Color.LightGreen;
            MessageBox.Show("Правильно! +10 очков", "Отлично",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
            StartNewGame();
        }
        else
        {
            score = Math.Max(0, score - 5);
            lblScore.Text = $"Очки: {score}";
            optionBoxes[index].BackColor = Color.LightCoral;
            MessageBox.Show("Неправильно! -5 очков", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
        }
    }
}
```

Листинг 7 — Класс MainMenu:

```
public class MainMenu : Form
{
    public MainMenu()
    {
        InitializeMenu();
    }

    private void InitializeMenu()
    {
        this.Text = "Геометрическая Головоломка";
        this.Size = new Size(400, 300);
        this.StartPosition = FormStartPosition.CenterScreen;
        this.BackColor = Color.FromArgb(240, 245, 249);
        this.FormBorderStyle = FormBorderStyle.FixedDialog;
        this.MaximizeBox = false;

        // Заголовок приложения
        Label titleLabel = new Label
        {
            Text = "ГЕОМЕТРИЧЕСКАЯ ГОЛОВОЛОМКА",
            Font = new Font("Segoe UI", 16, FontStyle.Bold),
            ForeColor = Color.DarkSlateBlue,
            Size = new Size(380, 60),
            Location = new Point(10, 20),
            TextAlign = ContentAlignment.MiddleCenter
        };
        this.Controls.Add(titleLabel);

        // Кнопка открытия редактора фигур
        Button btnEditor = new Button
        {
            Text = "Редактор фигур",
            Location = new Point(100, 100),
```

```
                Size = new Size(200, 50),
                BackColor = Color.FromArgb(70, 130, 180),
                ForeColor = Color.White,
                FlatStyle = FlatStyle.Flat,
                Font = new Font("Segoe UI", 11)
            };
            btnEditor.FlatAppearance.BorderSize = 0;
            btnEditor.Click += (s, e) => new ShapeSelectorForm().Show();
            this.Controls.Add(btnEditor);

            // Кнопка начала игры
            Button btnGame = new Button
            {
                Text = "Играть",
                Location = new Point(100, 170),
                Size = new Size(200, 50),
                BackColor = Color.FromArgb(95, 158, 160),
                ForeColor = Color.White,
                FlatStyle = FlatStyle.Flat,
                Font = new Font("Segoe UI", 11)
            };
            btnGame.FlatAppearance.BorderSize = 0;
                        btnGame.Click     +=     (s,     e)     =>     new
GameModeSelectionForm().ShowDialog();
            this.Controls.Add(btnGame);

            // Стилизация
            this.SetStyle(ControlStyles.OptimizedDoubleBuffer, true);
            this.SetStyle(ControlStyles.AllPaintingInWmPaint, true);
        }
    }
```