МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное   
учреждение высшего образования   
«Санкт-Петербургский политехнический университет Петра Великого»  
**Институт среднего профессионального образования**

|  |  |
| --- | --- |
|  |  |
|  |

**КУРСОВОЙ ПРОЕКТ**

Тема: «Игра с графическим интерфейсом 2048»

специальность 09.02.07 группа 32919/7

Студент Казанцев А.В.  
 (подпись) (ФИО)  
Преподаватель Прокофьев А.А.  
 (подпись) (ФИО)

Санкт-Петербург

2025

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙФЕДЕРАЦИИ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

федеральное государственное автономное образовательное учреждение

высшего образования

«Санкт-Петербургский политехнический университет Петра Великого»

(ФГАОУ ВО «СПБПУ»)

**Институт среднего профессионального образования**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отделение информационных технологий

**Задание на курсовое проектирование**

**по МДК 01.01 «Разработка программных модулей»**

Специальность: 09.02.07 Информационные системы и программирование

**студенту** Казанцев А.В. группа\_\_32919/7\_\_\_

(фамилия, инициалы)

**Тема:** **Программа для игры в поле-чудес**

**Исходные данные к проекту:**

1. Литература по описанию объекта информатизации
2. Литература по инструментальным средствам разработки прикладных программ
3. Лекции, стандарты.
4. Интернет – сайты по предметной области

**Перечень подлежащих разработке вопросов:**

1. Аналитический обзор программных средств, технологий.
2. Разработка ПО для решения поставленной задачи.
3. Разработка алгоритмов программных модулей.
4. Тестирование работы приложения.
5. Оформление документации (руководство оператора).

**Руководитель Прокофьев А.А**

(подпись, дата) (инициалы, фамилия)

**Задание принял к выполнению:**

**Студент Казанцев А.В.**

(подпись, дата) (инициалы, фамилия)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc201066411)

[1. Теоретические основы разработки 5](#_Toc201066412)

[1.1. Описание предметной области 5](#_Toc201066413)

[1.2. Анализ методов решения 5](#_Toc201066414)

[1.3. Обзор средств программирования 7](#_Toc201066415)

[1.4. Описание языка программирования 8](#_Toc201066416)

[2. Практическая часть 16](#_Toc201066422)

[2.1. Постановка задачи 16](#_Toc201066423)

[2.2. Проектирование приложения 16](#_Toc201066424)

[2.3. Текст программы 18](#_Toc201066425)

[2.4. Описание программы 19](#_Toc201066426)

[2.5. Руководство оператора 21](#_Toc201066433)

[2.6. Программа и методика испытаний 23](#_Toc201066437)

[2.7. Протокол испытаний 26](#_Toc201066444)

[ЗАКЛЮЧЕНИЕ 29](#_Toc201066445)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 30](#_Toc201066446)

[Приложение А 2](#_Toc201066447)

[Приложение Б 2](#_Toc201066448)

[Приложение В 22](#_Toc201066449)

# **ВВЕДЕНИЕ**

В последние годы игры становятся неотъемлемой частью образовательного процесса, предоставляя студентам и школьникам не только развлечение, но и возможность развивать когнитивные и аналитические способности. В частности, логические игры помогают улучшить внимание, память, реакцию и способность к решению нестандартных задач.

Цель работы — развитие когнитивных способностей пользователя, посредством развлекательного приложение «2048».

Для достижения цели разработки игры «2048» были поставлены следующие задачи:

* анализ требований и постановка задачи;
* проектирование архитектуры приложения;
* разработка приложения;
* тестирование
* подготовить пояснительную записку к проекту.

Одной из таких популярных игр является 2048, которая, несмотря на свою простоту, требует от игрока высокого уровня стратегического мышления и концентрации. Игра представляет собой головоломку, в которой необходимо объединять одинаковые числа, чтобы в конечном итоге достичь числа 2048. Это способствует развитию логики и принятия решений, а также тренирует скорость реакции и память.

Область применения работы представляет собой индустрию развлечений, в которой игровая индустрия является подотраслью, сосредоточенной на разработке, производстве и распространении видеоигр, которые предоставляют пользователям уникальные и интерактивные формы досуга.

# **Теоретические основы разработки**

# **Описание предметной области**

Компьютерные игры — это интерактивная программа, в которой пользователь взаимодействует с виртуальным миром, с целью развлечения. Зачастую игрок управляет персонажами или элементами игры при помощи компьютера.

Компьютерные игры давно стали значимой частью современного цифрового мира. Среди различных жанров особое место занимают игры-головоломки, которые сочетают в себе элементы логического мышления и стратегического планирования.

Одним из ярких представителей этого жанра является игра **«**2048**»**, разработанная в 2014 году. Основная цель игры — объединять одинаковые числовые плитки на игровом поле размером 4×4 так, чтобы получить плитку с числом 2048.

# **Анализ методов решения**

Для реализации игры 2048 можно рассмотреть различные вариации игровой механики, которые расширяют или изменяют классический геймплей. Например, можно добавить выбор размерности игрового поля 4x4, 5x5, 6x6, изменить конечное число, которое игрок должен собрать из плиток, к примеру «1337», добавить ограничить время на ход.

Для разработки таких вариаций и самой игры могут быть использованы различные движки и фреймворки:

1. WPF и WinForms

WPF и WinForms — две платформы для создания desktop-приложений на .NET. В WinForms отрисовка графики осуществляется с помощью класса Graphics, а в WPF используется более современный подход с элементами управления, такими как Canvas, и поддержкой анимаций и трансформаций. Обработка нажатий клавиш происходит через события KeyDown и KeyUp, а обновление игры реализуется с помощью таймера, который регулярно перерисовывает кадры. Ресурсы загружаются из файлов или встроенных ресурсов проекта. Для работы с базой данных обычно применяется SQL Server с подключением через ADO.NET или ORM, что обеспечивает удобное управление данными [3].

Достоинства: простота и доступность для разработчиков, хорошая интеграция с Windows, стандартные средства отрисовки графики и мощные инструменты для работы с базами данных.

Недостатки: ограниченные возможности для сложной анимации по сравнению с игровыми движками, меньшая производительность, особенно в WinForms, необходимость большого объема ручного кода для игровой логики и визуальных эффектов, а также ориентация платформы на бизнес-приложения, что усложняет разработку игр.

1. Unity

Unity — мощный игровой движок для создания 2D и 3D игр, где игровая сцена строится из объектов со спрайтами. Управление логикой и движением реализуется скриптами на C#, обрабатывающими ввод пользователя, физику, столкновения и подсчёт очков. Отрисовка и анимация выполняются средствами движка, обеспечивая плавность и высокую производительность. Для хранения данных, таких как рекорды и статистика, используется локальный сервер MySQL с PHP-скриптами, которые принимают запросы от Unity, взаимодействуют с базой и возвращают результаты. Обмен данными с сервером осуществляется через классы UnityWebRequest или WWW.

Достоинства: Unity является мощным игровым движком с широкими возможностями для создания 2D и 3D графики, анимации и физики. Он оснащён удобным визуальным редактором и инструментами для быстрой разработки и отладки, обеспечивает простую интеграцию с серверными технологиями через HTTP-запросы и позволяет использовать локальный сервер MySQL с PHP для безопасного и масштабируемого хранения данных.

Недостатки: Для работы с Unity требуется изучение специфики движка и языка C#, что может быть сложно для новичков. Кроме того, движок создаёт более высокую нагрузку на систему по сравнению с простыми приложениями, требует настройки локального сервера и написания серверных скриптов для работы с базой данных, а итоговый размер приложения и зависимость от движка могут быть больше, чем у нативных решений[1].

Игра «2048» не требует создания сложны сцен и анимации, исходя из этого для разработки был выбран язык программирования C# в связке с WPF.

# **Обзор средств программирования**

Для разработки можно использовать следующие средства программирования:

1. WPF – это технология от Microsoft для создания графических пользовательских интерфейсов в приложениях под Windows. При разработке используется XAML-разметка и язык программирования C# [4]

Достоинства: WPF предоставляет гибкие средства для построения интерфейсов и визуальных компонентов, включая поддержку привязки данных, шаблонов, анимаций и обработки пользовательских событий.

Недостатки: WPF изначально не предназначен для создания игр с насыщенной графикой, сложной физикой или трехмерной анимацией. Отсутствие встроенного игрового движка означает, что разработчику приходится самостоятельно реализовывать всю игровую логику, отрисовку, таймеры и анимации.

1. Visual Studio и Visual Studio Code — интегрированные среды разработки для написания, редактирования и отладки скриптов на языке C#, который используется в Unity для реализации игровой логики и взаимодействия компонентов.

Достоинства: среди ключевых преимуществ – удобные инструменты отладки и интеллектуальное автодополнение кода, которые значительно ускоряют процесс разработки и тестирования игровой логики. Глубокая интеграция с Unity позволяет мгновенно переключаться между редактором движка и кодом, повышая продуктивность работы.

Недостатки: начинающим разработчикам может быть сложно освоить эти среды из-за их насыщенного функционала и множества настроек. Для оптимальной работы с Unity часто требуется дополнительная конфигурация, что может представлять определенные трудности для новичков.

1. C# — основной язык программирования в Unity, позволяющий создавать скрипты для управления объектами, обработки пользовательского ввода, реализации физики и игровой логики.

Достоинства: C# позволяет легко реализовать игровую логику, обработку ввода и взаимодействие с физикой. Он обеспечивает хорошую производительность и имеет отличную поддержку со стороны Unity. Кроме того, у C# большое сообщество разработчиков и множество обучающих материалов, что упрощает обучение и решение проблем.

Недостатки: для работы с C# требуются базовые знания программирования и понимание объектно-ориентированного подхода, что может затруднить начало работы для новичков. Также при разработке сложных игровых механик может потребоваться дополнительное время на отладку и оптимизацию кода [9].

Так как в игре 2048 не требуется реализация сложных трехмерных анимаций и сложной физики. Для создания игры был выбран WPF и C#. Средой для разработки был выбран Visual Studio, который предоставляет широкий набор функций и интеграций.

# **Описание языка программирования**

# **Общие сведения**

C# — это современный объектно-ориентированный язык программирования, разработанный компанией Microsoft. Он поддерживает множество парадигм программирования, включая императивное, декларативное, функциональное, объектно-ориентированное и компонентно-ориентированное программирование. C# отличается статической типизацией, сильной типизацией и наличием мощной стандартной библиотеки.

C# создавался в рамках инициативы Microsoft .NET под руководством Андерса Хейлсберга как альтернатива Java с целью интеграции преимуществ объектно-ориентированного подхода и удобства для разработки приложений под Windows. Разработка началась в 1999 году под кодовым названием Cool. В 2000 году язык был переименован в C#, а его первая версия выпущена в 2002 году вместе с .NET Framework.

Основные особенности C#:

* автоматическое управление памятью;
* единая система типов;
* обработка исключений;
* лямбда-выражения и асинхронные операции;
* кроссплатформенность.

C# — универсальный язык, используемый в различных сферах разработки:

* разработка приложений для Windows;
* веб-разработка;
* игровая индустрия;
* мобильные приложения;
* облачные вычисления и IoT [9].

# **Элементы языка**

Алфавит языка C# состоит из следующих элементов:

* идентификаторы;
* ключевые слова;
* знаки операций;
* константы;
* разделители.

Переменные представляют собой именованные области памяти для хранения данных. Изменение значения переменной возможно в процессе выполнения программы. Синтаксис определения переменной: тип имя\_переменной. Переменные могут быть инициализированы при объявлении: string name = "Tom".

Константы определяются с помощью ключевого слова const и должны быть инициализированы при объявлении. Значение константы не может быть изменено после определения. Например, const string NAME = "Tom";

C# поддерживает несколько базовых типов данных:

* int: целые числа (123);
* double: числа с плавающей запятой (19.99);
* char: одиночные символы ('A');
* string: строки текста ("Hello");
* bool: логические значения (true или false).

Выражения в C# состоят из операндов и операторов. Правила составления выражений включают:

* приоритет операций;
* ассоциативность.

C# включает различные группы операторов, которые приведены в таблице 1.

Таблица 1 - Операторы языка C#

|  |  |
| --- | --- |
| **Группа операторов** | **Описание** |
| Арифметические | Выполняют математические операции (+, -, \*, /, %) |
| Сравнения | Сравнивают значения (==, !=, >, <, >=, <=) |
| Логические | Выполняют логические операции (&&, ||, !, ^) |
| Побитовые | Выполняют операции на уровне битов (&, |, ^, ~, <<, >>) |
| Присваивания | Присваивают значения переменным (+=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=) |

Арифметический оператор сложения:

* синтаксис: int sum = a + b;
* семантика: складывает значения переменных a и b.

Оператор сравнения:

* синтаксис: bool isEqual = a == b;
* семантика: проверяет равенство значений a и b.

Логический оператор И:

* синтаксис: bool result = condition1 && condition2;
* семантика: возвращает true только если оба условия истинны.

Оператор присваивания:

* синтаксис: x = 10;
* семантика: присваивает значение 10 переменной x.

Побитовый оператор AND:

* синтаксис: int result = a & b;
* семантика: выполняет побитовое И между значениями a и b, возвращая число, у которого в каждом бите стоит 1, только если в соответствующих битах a и b тоже стоят 1 [9].

# **Способы структурирования программы**

Программа на C# обычно состоит из следующих основных элементов:

* пространства имен;
* классы и структуры;
* точка входа;
* комментарии.

Пример минимальной структуры программы:

using System;

namespace MyApplication

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

В C# все подпрограммы называются методами. Различие между функцией и процедурой следующее:

* функция — метод, который возвращает значение;
* процедура — метод, который не возвращает значения.

Объявление процедуры:

void PrintMessage(string message)

{

Console.WriteLine(message);

}

Объявление функции:

int Sum(int a, int b)

{

return a + b;

}

Вызов методов:

PrintMessage("Привет, мир!");

int result = Sum(2, 3);

Параметры объявляются в определении метода, в то время как аргументы передаются при вызове метода.

Способы передачи параметров:

* по значению: в метод передаётся копия значения переменной, поэтому изменения внутри метода не отражаются на исходной переменной;
* по ссылке: с использованием ключевых слов ref или out метод получает доступ к исходной переменной и может изменять её значение;
* параметры с модификатором params: позволяет передавать в метод произвольное кол-во аргументов одного типа [9].

# **Встроенные элементы**

Библиотечные модули, использованные в разработанной программе:

1. TMPro — расширенный модуль для работы с текстом в Unity, обеспечивающий высокое качество рендеринга и гибкие возможности форматирования текста [3].

Основные методы и свойства:

* .text — получение и установка текста;
* .placeholder — свойство подсказки;
* .gameObject.SetActive(bool) — управление видимостью UI-элементов.

Используемые компоненты:

* TMP\_InputField — поле ввода текста с поддержкой TextMesh Pro;
* TMP\_Text и TextMeshProUGUI — компоненты для отображения текста в UI.

1. UnityEngine - основной модуль Unity, предоставляющий фундаментальные классы для создания игр и приложений [3].

Основные классы и методы:

* MonoBehaviour — базовый класс для скриптов;
* Start(), Awake(), Update() – методы жизненного цикла;
* GameObject — игровой объект сцены;
* Transform — позиция, вращение и масштаб объекта;
* Time.timeScale — управление скоростью времени;
* AudioSource.Play() — воспроизведение звука;
* Destroy(GameObject) — уничтожение объекта;
* GetComponent<T>() — получение компонента на объекте;
* InvokeRepeating(string methodName, float time, float repeatRate) — повторный вызов метода;
* CancelInvoke(string methodName) — отмена повторного вызова.

Корутинные методы:

* StartCoroutine(IEnumerator) — запуск асинхронных операций.

UnityEngine.UI - библиотека компонентов пользовательского интерфейса Unity.

Основные компоненты:

* Button — кнопка, реагирующая на нажатия.

Основные методы:

* onClick.AddListener(Action) — добавление обработчика события нажатия кнопки.

1. UnityEngine.Networking - модуль для выполнения HTTP-запросов

Основные классы и методы:

* UnityWebRequest.Post(string url, WWWForm form) — создание POST-запроса;
* SendWebRequest() — отправка запроса;
* Свойства result, error, downloadHandler.text — результат запроса, ошибки и полученный текст.

System, System.Collections - стандартные библиотеки C#.

Используемые компоненты:

* Делегаты Action<bool> — передача результатов асинхронных операций;
* Интерфейс IEnumerator — реализация корутин и асинхронного выполнения кода.

# **Средства отладки**

Отладчик позволяет выполнять программу по одному оператору за раз, используя команды:

* Step Over — выполнить текущую строку, не заходя внутрь вызова метода;
* Step Into — перейти внутрь вызываемого метода.

Можно устанавливать точки останова на нужных строках кода, чтобы выполнение программы автоматически приостанавливалось в этих местах:

* обычные точки останова – останавливают выполнение на определенной строке кода;
* условные точки останова - останавливают выполнение только при выполнении определённого условия;
* функциональные точки останова - срабатывают при входе в определённую функцию;
* логические точки останова — не останавливают выполнение, а выводят диагностическое сообщение в консоль.

Во время остановки программы можно просматривать значения переменных в различных окнах:

* окно Watch позволяет отслеживать значения выбранных переменных и выражений в процессе отладки;
* окно «Locals» показывает все локальные переменные текущего контекста;
* окно Autos автоматически отображает переменные, которые используются рядом с текущей строкой кода.
* окно Call Stack отображает последовательность вызовов функций, которые привели к текущей точке выполнения [8].

# **ПРАКТИЧЕСКАЯ ЧАСТЬ**

# **Постановка задачи**

Постановка задачи изложена в разделах программного документа «Техническое задание» (Приложение А), включающих: «Основания для разработки», «Назначение разработки», «Требования к программе», «Требования к программной документации», «Стадии и этапы разработки» и «Порядок контроля и приемки».

# **Проектирование приложения**

На этапе проектирования были разработаны диаграмма прецедентов и диаграмма классов.

Согласно диаграмме прецедентов, представленной на рисунке 1, пользователь может выполнять следующие действия:

* запуск игры;
* просмотр счета;
* перемещение плиток;
* сброс игры;
* возврат в главное меню;



Рисунок 1 - Диаграмма прецедентов UML

Согласно диаграмме классов, представленной на рисунке 2, программа состоит из трех независимых классов, связанных иерархическими и ассоциативными отношениями.



Рисунок 2 – Диаграмма классов UML

# **Текст программы**

Текст программы представляет собой документ, выполненный машинным способом (приложение В).

# **Описание программы**

# **Общие сведения**

Полное название программы – «2048». Приложение разработано на языке программирования C# и функционирует под управлением операционной системы

# **Назначение программы**

Программа предоставляет пользователю развлекательное приложение. Приложение развивает когнитивные навыки пользователя и имеет развлекательный характер.

# **Описание логической структуры**

Исходный текст программы находится в следующих файлах: Tile.cs, GameBoard.cs, GameWindowxaml.cs.

Tile.cs – класс, представляет собой простую структуру для хранения информации о плитке в игре.

* Конструктор Tile — который нужен для создания объектов класса Tile;

GameBoard.cs – класс, представляющий собой игровое поле и управляет логикой игры.

Назначение методов:

* AddRandomTile() — выбирает пустую ячейку и устанавливает в нее значение 2 или 4 с разной вероятностью;
* IsGameOver() — проверяет наличие пустых ячеек и возможных ходов, и при их отсутствии завершает игру ;
* Reset() — полностью состояние игры к начальному.
* MoveLeft() — обеспечивает движение плиток влево;
* MoveRight() — обеспечивает движение плиток вправо;
* MoveUp() — обеспечивает движение плиток вверх;
* MoveDown() — обеспечивает движение плиток вниз;

GameWindowxaml.cs – класс, представляет основное окно приложения и управляет пользовательским интерфейсом.

Назначение методов:

* InitGridUi() — отчищает сетку GameGrid и настраивает метки для Label для каждой плитки;
* DrawBoard() — отрисовывает игровое поле;
* OnKeyDownHandler(object sender, KeyEventArgs e) — обрабатывает нажатие клавиш, в зависимости от нажатой клавиши вызывает методы движения плиток, обновляет текущий счет, вызывает метод добавления плитки
* Reset\_Click(object sender, RoutedEventArgs e) — обрабатывает нажатие на кнопку Reset, вызывает метод Reset(), сбрасывает счет и перерисовывает поле ;
* private void MenuButton\_Click(object sender, RoutedEventArgs e) — обрабатывает нажатие на кнопку Menu, переносит пользователя в главное меню;
* LoadBestScore() —считывает лучший счет из файла bestscore.txt.
* SaveBestScore(int score) – Сохраняет лучший счет в файл betscore.txt

# **Используемые технические и программные средства**

Требования к составу и параметрам технических средств:

* персональный компьютер, включающий клавиатуру, мышь и монитор;
* процессор Intel Core i3 8-ого поколения или эквивалентный AMD и новее;
* объем оперативной памяти минимум 4 Гб;
* свободный объем памяти на жестком диске минимум 500 Мб;
* видеокарта или встроенное графическое ядро процессора

Требования к информационной и программной совместимости:

* операционная система Windows 7, 8, 10 или 11;
* среда выполнения .NET Framework 4.7.2.

# **Вызов и загрузка**

Для запуска программы необходимо:

* Дважды щелкнуть по исполняемому файлу с названием Game2048 c расширением .exe.

# **Входные и выходные данные**

Входными данными являются:

* события управления от пользователя – нажатия клавиш и клики мыши, представленные в виде булевых значений;
* состояние игровых объектов, представленное в виде экземпляров классов;

Выходными данными являются:

* графическое отображение текущего состояния игры;
* текстовые сообщения, отображаемые в пользовательском интерфейсе.

# **Руководство оператора**

# **Назначение программы**

Программа предоставляет пользователю развлекательное приложение. Приложение развивает когнитивные навыки пользователя и имеет развлекательный характер.

# **Условия выполнения программы**

Требования к составу и параметрам технических средств:

* персональный компьютер, включающий клавиатуру, мышь и монитор;
* процессор Intel Core i3 8-ого поколения или эквивалентный AMD и новее;
* объем оперативной памяти минимум 4 Гб;
* свободный объем памяти на жестком диске минимум 500 Мб;
* видеокарта или встроенное графическое ядро процессора

Требования к информационной и программной совместимости:

* операционная система Windows 7, 8, 10 или 11;
* среда выполнения .NET Framework 4.7.2.

# **Выполнение программы и сообщение оператору**

Для запуска игры пользователю необходимо открыть исходный файл, Game2048.exe.

Загружается главное меню. Для начала игры пользователю необходимо нажать кнопку «Play», расположенную в центре экрана. Управление перемещением плиток осуществляется при помощи стрелок на клавиатуре. При каждом нажатии стрелки «Влево» «Вправо» «Вверх» «Вниз» плитки перемещаются в нужном направлении соответственно. При каждом новом нажатии в свободной ячейке появляется новая плитка со значением «2» или «4». По возможности плитки объединяются в сумму двух плиток. При отсутствии свободных ячеек и не возможности объединить соседние плитки пользователю выводится сообщение: «Игра окончена! Ваш счёт: » и игра начинается заново. Если же пользователю удалось собрать плитку «2048», выводится сообщение: «Поздравляем! Вы собрали плитку 2048!». Во время игры при соединении плиток счет обновляется.  
 При нажатии на кнопку «Menu» пользователю открывается главное меню приложения. При нажатии кнопки «Reset» сбрасывается игровое поле и обнуляется счет.

В главном меню пользователь может заново начать игру, нажав кнопку «Play» или же выйти из приложения, нажав кнопку «Quit»

Игрок может повторять попытки, чтобы увеличить счет.

# **Программа и методика испытаний**

# **Объект испытаний**

Объектом испытаний является разработанная программа – игра «2048», включающая игровую логику, графический интерфейс.

# **Цель испытания**

Цель испытаний — подтвердить, что программа соответствует требованиям, изложенным в техническом задании (приложение А).

# **Требования к программе**

В процессе проведения испытаний согласно пункту «Требования к программе» технического задания (приложение А) могут быть проверены следующие требования:

Требования к составу выполняемых функций:

* управление перемещения плиток;
* объединение плиток;
* начисление очков после каждого объединения плиток;
* отображение лучшего результата пользователя;
* запуск игры по нажатию кнопки «Play»;
* перенос пользователя в главное меню при нажатии кнопки «Menu»
* правильная работа кнопки «Restart»;
* программа автоматически завершает игру при заполненном игровом поле и невозможности хода или получения плитки со значением «2048»

Требования к организации входных и выходных данных:

* автоматическая запись и сохранение лучшего счета в .txt файл

Требования к временным характеристикам:

* обновление информации о лучшем счете в игре сразу после запуска игры

# **Требования к программной документации**

* текст программы (ГОСТ 19.401-78);
* описание программы (ГОСТ 19.402-78);
* руководство оператора (ГОСТ 19.505-79);
* программа и методика испытаний (ГОСТ 19.301-79);
* описание языка (ГОСТ 19.507-79).

# **Средства и порядок испытаний**

Технические средства, необходимые для проведения испытаний:

* стандартные устройства ввода и вывода: персональный компьютер, клавиатура, мышь, монитор;
* процессор Intel Core i3 8-ого поколения или эквивалентный AMD и новее;
* объем оперативной памяти минимум 4 Гб;
* свободный объем памяти на жестком диске минимум 100 Мб;
* видеокарта или встроенное графическое ядро процессора
* источник бесперебойного питания.

Программные средства, необходимые для проведения испытаний:

* операционная система Windows 7, 8, 10 или 11;
* среда выполнения .NET Framework 4.7.2;

Порядок проверки требований:

* проверка выполнения требований к программной документации;
* проверка упаковки и маркировки носителя с программой;
* проверка требований к функциональным характеристикам программы.

# **Методы испытаний**

Для проверки функциональных характеристик программы применялся метод тестирования «черного ящика», который основан на проверке соответствия поведения приложения заданным требованиям без анализа внутренней структуры кода. В таблице 2 приведены сценарии тестирования, включающие взаимодействие с элементами интерфейса и ожидаемые результаты, что позволяет оценить корректность работы игровых функций и пользовательских операций с точки зрения конечного пользователя. Такой подход обеспечивает выявление ошибок и подтверждение соответствия функционала спецификациям за счёт тестирования внешнего поведения программы.

Помимо метода тестирования «черного ящика» использовалось модульное тестирование, основанный на проверки корректности работы отдельных модулей программы

# **Протокол испытаний**

Таблица 2 – Протокол Методов испытаний

| **Проверяемые требования** | **Методика** | **Ожидаемые результаты** | **Результаты** |
| --- | --- | --- | --- |
| Тестирование инициализации игрового поля | Модульное тестирование, белый ящик | Счет на игровом поле появляется две плитки | Пройден  (Рис. 6) |
| Тестирование перемещения и объединения плиток | Модульное тестирование, белый ящик | Плитки с одинаковыми значениями | Пройден  (Рис. 6) |
| Тестирование окончания игры при получении плитки со значением «2048» | Модульное тестирование, белый ящик | Игра заканчивается и пользователю выводится сообщение: «Поздравляем! Вы собрали плитку 2048!» | Пройден  (Рис. 6) |
| Проверка способности программы перезапускать игру | Функциональное тестирование, черный ящик | Пользователю откроется игровое поле, на котором уже будут сгенерированы 2 плитки со случайными значениями | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 8) |
| Проверка способности программы увеличивать счет. | Функциональное тестирование, черный ящик | Под надписью «Best Score» будет отображаться самый большой счет, который набирали в игре на данном компьютере. Если пользователь побил лучший счет, то счет автоматически будет увеличиваться вместе с обычным счетом | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 6) |
| Проверка способности программы открывать главное меню | Функциональное тестирование, черный ящик | Поле нажатия на кнопку пользователю откроется главное меню | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 9) |

Таблица 3– Протокол испытаний

| **Проверяемые требования** | **Сообщения программы и вводимые значения** | **Ожидаемые результаты** | **Результаты** |
| --- | --- | --- | --- |
| Проверка способности программы осуществлять игровой процесс. | Нажать на кнопку «Play» | Пользователю откроется игровое поле, на котором уже будут сгенерированы 2 плитки со случайными значениями | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 1-2) |
| Проверка способности программы перемещать плитки. | Нажать на стрелку вниз на клавиатуре | Плитки переместятся в последнюю свободную ячейку в строке или столбце в зависимости от нажатой клавши | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 3) |
| Проверка способности программы соединять одинаковые плитки | Продолжить играть и соединить плитки с одинаковым значением, например «2» и «2» | Ячейки объединяться в одну со значением «4» и ячейка изменит свой цвет | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 4) |
| Проверка способности программы увеличивать счет. | Продолжить играть и при каждом объединении плиток счет будет увеличиваться | При объединении ячейки «2» и «2» счет будет равен «4» и т.д | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 5) |
| Проверка способности программы выводить лучший счет пользователя. | - | Под надписью «Best Score» будет отображаться самый большой счет, который набирали в игре на данном компьютере. Если пользователь побил лучший счет, то счет автоматически будет увеличиваться вместе с обычным счетом | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 6) |
| Проверка способности программы отображать окончание игры при неуспешной попытке | Продолжить играть | Когда не останется свободных ячеек и не будет возможности объединить плитки, пользователю выведется сообщение: «Игра окончена! Ваш счёт: » | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 7) |
| Проверка способности программы перезапускать игру | Нажать на кнопку «Reset» | Поле полностью отчистится и появятся две случайные плитки в случайных ячейках поля. Счет обнулится | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 8) |
| Проверка способности программы открывать главное меню | Нажать на кнопку «Menu» | Поле нажатия на кнопку пользователю откроется главное меню | Пройден  (ПРИЛОЖЕНИЕ Б Рис. 9) |

Проверка корректность начального игрового состояния, проверка окончания игры при достижении плитки со значением «2048», проверка метода перемещения плиток представлена на рисунке 6.

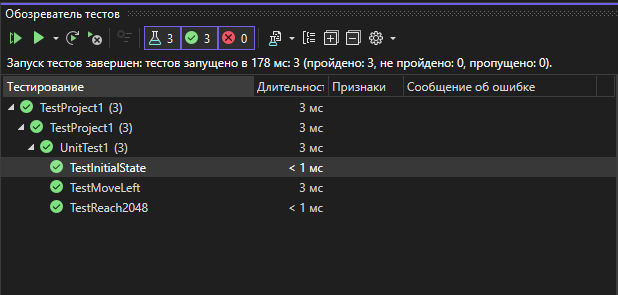


Рисунок 6 - Unit-Test

# **ЗАКЛЮЧЕНИЕ**

В ходе разработки программы «2048» были успешно выполнены все задачи, поставленные в техническом задании. Результатом работы стало настольное приложение — интерактивная головоломка, адаптированная для операционной системы Windows. Программа реализует основные игровые механики.

Ключевые достоинства созданного приложения:

* простой и выдержанный в одной стилистике пользовательский интерфейс;
* возможность отслеживания лучшего счета;

Разработанная игра может использоваться в сфере цифровых развлечений как настольная логическая игра, способствующая развитию концентрации, стратегического мышления и внимания. Также благодаря функциональности сохранения результатов и онлайн-сравнения она может применяться для организации пользовательских соревнований.

Помимо развлекательной составляющей, проект может выполнять и образовательную функцию — он подходит для демонстрации принципов построения пользовательского интерфейса и логики работы игровых приложений на языке программирования C# с использованием технологии WPF.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Unity documentation [Электронный ресурс]. – URL: <https://docs.unity3d.com/> – Режим доступа: свободный.
2. Nadia Comăneci alege Excellence Crème cu Acid Hialuronic [Электронный ресурс]. – URL: <https://www.lorealparis.ro/> – Режим доступа: свободный.
3. Документация по .NET // Microsoft Learn [Электронный ресурс]. –URL: <https://learn.microsoft.com/ru-ru/dotnet/> – Режим доступа: свободный.
4. Windows Presentation Foundation (WPF) // Microsoft Docs [Электронный ресурс]. – URL: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/> – Режим доступа: свободный.
5. Руководство по C# — управляемый язык .NET // Microsoft Learn [Электронный ресурс]. URL – <https://learn.microsoft.com/ru-ru/dotnet/csharp/> – Режим доступа: свободный.
6. Компьютерная игра Рувики: Интернет-энциклопедия [Электронный ресурс]. URL – <https://ruwiki.ru/> – Режим доступа: <https://ruwiki.ru/>, свободный.
7. ГОСТ 19.105–78 Система стандартов по информации, библиотечному и издательскому делу. Общие требования к программным документам.
8. ГОСТ 19.106–78 Система стандартов по информации, библиотечному и издательскому делу. Требования к программным документам, выполненным печатным способом.
9. ГОСТ 19.201–78 Система стандартов по информации, библиотечному и издательскому делу. Техническое задание. Требования к содержанию и оформлению.
10. ГОСТ 19.301–78 Система стандартов по информации, библиотечному и издательскому делу. Программа и методика испытаний. Требования к содержанию и оформлению.
11. ГОСТ 19.401–78 Система стандартов по информации, библиотечному и издательскому делу. Текст программы. Требования к содержанию и оформлению.
12. ГОСТ 19.402–78 Система стандартов по информации, библиотечному и издательскому делу. Описание программы. Требования к содержанию и оформлению.
13. ГОСТ 19.505–79 Система стандартов по информации, библиотечному и издательскому делу. Руководство оператора. Требования к содержанию и оформлению.
14. ГОСТ 19.701–90 Система стандартов по информации, библиотечному и издательскому делу. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

Приложение А

(справочное)

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное автономное образовательное учреждение высшего   
профессионального образования  
«Санкт-Петербургский государственный политехнический университет Петра Великого»  
(ФГАОУ ВО «СПбПУ)  
**Институт среднего профессионального образования**

Компьютерная игра «2048» с графическим интерфейсом  
**Техническое задание**

ВЫПОЛНИЛ  
Студент группы 32919/7  
\_\_\_\_\_\_\_\_\_\_\_ Казанцев А.В.  
\_\_.\_\_. 2025

2025

1. **ВВЕДЕНИЕ**
   1. **Наименование программы**

Наименованием программной разработки является игра «2048».

* 1. **Краткая характеристика и области применения**

Компьютерная игра "2048" с графическим интерфейсом: пользователь перемещает плитки с числами по игровому полю. Задача игрока — объединять одинаковые плитки, чтобы получить плитку с числом 2048 и более высокие значения, набирая при этом максимальное количество очков. Игра заканчивается, когда игровое поле полностью заполнено и нет возможных ходов.

Область применения представляет собой индустрию развлечений, в которой игровая индустрия является подотраслью, сосредоточенной на разработке, производстве и распространении видеоигр, которые предоставляют пользователям уникальные и интерактивные формы досуга.

1. **ОСНОВАНИЯ ДЛЯ РАЗРАБОТКИ**

Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Разработка программных модулей» и утверждена Институтом среднего профессионального образования.

Проект утвержден Прокофьевым Антоном Андреевичем, именуемым далее заказчиком и Казанцеву Артемию Владимировичу, именуемой далее исполнителем.

Согласно проекту, исполнитель должен разработать программу и предоставить ее заказчику не позднее 20.06.2025.

* 1. **Наименование темы**

Наименованием темы является «Игра с графическим интерфейсом 2048».

1. **НАЗНАЧЕНИЕ РАЗРАБОТКИ**
   1. **Функциональное назначение**

Основное назначение программного продукта заключается в организации компьютерной игры с графическим интерфейсом «2048»: пользователь играет, перемещая плитки с числами по игровому полю. Задача игрока — объединять одинаковые плитки, чтобы получить плитку с числом «2048», набирая при этом максимальное количество очков. Игра заканчивается, когда игровое поле полностью заполнено и нет возможных ходов.

* 1. **Эксплуатационное назначение**

Эксплуатационное назначение программного продукта: программа предназначена для широкого круга пользователей, без ограничения по возрасту, не требует внесения денежных средств или использования платёжных систем для игры, предназначена для развлекательных целей.

1. **ТРЕБОВАНИЯ К ПРОГРАММЕ**
   1. **Требования к функциональным характеристикам**

* Программа должна случайным образом генерировать новые плитки с числами (2 или 4) на игровом поле после каждого хода игрока;
* Программа должна обеспечивать перемещение плиток по игровому полю (вверх, вниз, влево, вправо) и их слияние при совпадении чисел;
* Программа должна отслеживать счет игрока и отображать текущий счет;
* Программа должна автоматически завершать игру, когда игровое поле заполнено и нет возможных ходов.
  + 1. **Требования к составу выполняемых функций**
* Пользователь управляет игровым процессом с помощью стрелок на клавиатуре. Каждое нажатие клавиши перемещает все плитки на игровом поле в соответствующем направлении. При совпадении одинаковых плиток они объединяются, увеличивая значение.
* При каждом ходе на игровом поле появляется новая плитка с числом «2» или «4» в случайной свободной ячейке.
* Игра начинается при нажатии на кнопку «Старт» на главном экране.
* Игра завершается, когда на игровом поле не остается возможных ходов (нет свободных ячеек и невозможны объединения).
* За каждое объединение плиток начисляются очки. Текущее количество очков отображается на экране во время игры.
* Игра имеет таблицу лидеров, в которой отображаются пользователи с результатами среди всех зарегистрированных пользователей.
  + 1. **Требования к организации входных и выходных данных**

Информация о пользователях записывается после завершения игры в текстовом файле

* + 1. **Требования к временным характеристикам**
  1. **Требования к надежности**

Вероятность безотказной работы программы при условии соблюдения требований к обеспечению устойчивого функционирования должна составлять 99.99%.

* + 1. **Требования к обеспечению устойчивого функционирования**
* Использование лицензированного программного обеспечения.
* Регулярная проверка программы на наличие вредоносного программного обеспечения.
* Использование бесперебойного питания.
* Создание резервной копии программы.
  + 1. **Время восстановления после отказа**

Время восстановления зависит от периода, необходимого пользователю для устранения неисправности и повторного запуска программы.

* + 1. **Отказы из-за некорректных действий оператора или пользователя**

В программу внедрены механизмы валидации вводимых данных. В случае некорректных действий пользователя система информирует о необходимости исправления ошибок перед тем, как продолжить работу.

* 1. **Условия эксплуатации**
* Общий функционал игры доступен в оффлайн-режиме без необходимости постоянного подключения к интернету.
  + 1. **Климатические условия эксплуатации**

Климатические условия эксплуатации не требуются.

* + 1. **Требования к видам обслуживания**

Программный продукт не требует проведения каких-либо видов обслуживания.

* + 1. **Требования к численности и квалификации персонала**

Программист:

* Опыт работы с WPF.
* Знание языков программирования C# и XAML разметки.
  + 1. **Требования к составу и параметрам технических средств**
* персональный компьютер, включающий клавиатуру, мышь и монитор;
* процессор Intel Core i3 8-ого поколения или эквивалентный AMD и новее.
* объем оперативной памяти минимум 4 Гб.
* свободный объем памяти на жестком диске минимум 500 Мб.
* видеокарта или встроенное графическое ядро процессора.
* источник бесперебойного питания.
  1. **Требования к информационной и программной совместимости**
* операционная система Windows 7, 8, 10 или 11.
* среда выполнения .NET Framework 4.7.2.
  1. **Требования к маркировке и упаковке**

Программа должна поставляться в виде инсталлятора, после установки должны присутствовать следующие файлы:

* исполняемый файл «Game2048» запуска игры с расширением «.exe».
* динамическая библиотека «Game2048.dll»
* конфигурационные файлы.
  1. **Требования к транспортированию и хранению**

Программа распространяется в электронном виде. Требования к транспортировке и хранению не предъявляются.

* 1. **Специальные требования**

Программа должна обеспечивать взаимодействие с пользователем посредством графического пользовательского интерфейса.

1. **ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ**
   1. **Предварительный состав программной документации**

* Текст программы (ГОСТ 19.401-78).
* Описание программы (ГОСТ 19.402-78).
* Руководство оператора (ГОСТ 19.505-79).
* Программа и методика испытаний (ГОСТ 19.301-79).
* Описание языка (ГОСТ 19.507-79).
  1. **Перечень материалов пояснительной записки**

Перечень материалов пояснительной записки представлен на рисунке 1.

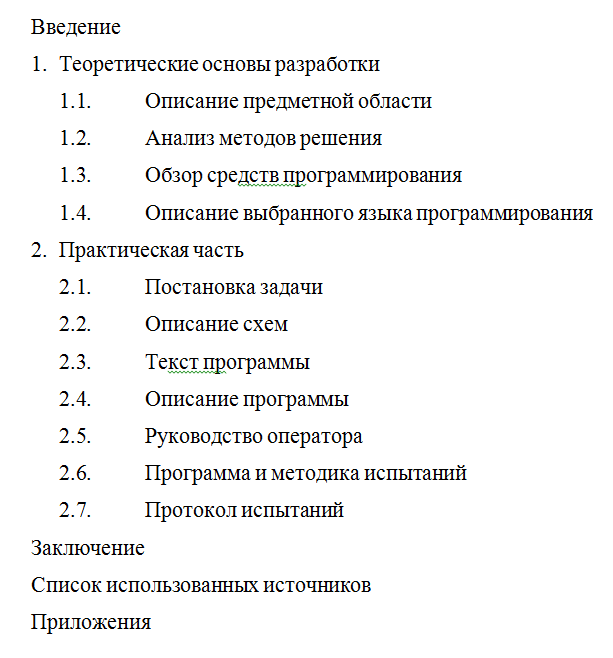


Рисунок 1 – Перечень материалов пояснительной записки

1. **ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ**

Технико-экономические показатели не рассчитываются.

1. **СТАДИИ И ЭТАПЫ РАЗРАБОТКИ**

Стадии и этапы разработки представлены в таблице 1.

Таблица А.1 – Стадии и этапы разработки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Содержание стадии** | **Содержание этапа** | **Срок 2025 г.** | | **Форма отчетности** |
| **начало** | **конец** |
| Техническое задание | Составление технического задания | 21.04.2025 | 25.04.2025 | Техническое задание |
| Эскизный проект | Проектирование программы | 28.04.2025 | 09.05.2025 | UML-диаграммы |
| Рабочий проект | Разработка спецификаций | 12.05.2025 | 16.05.2025 | Спецификации отдельных компонентов |
| Составление программы | 19.05.2025 | 06.06.2025 | Программная документация |
| Приёмо-сдаточные испытания | 09.06.2025 | 13.06.2025 | Протокол испытаний (п. 2.7 пояснительной записки) |
| Приёмка | Защита курсового проекта | 16.06.2025 | 20.06.2025 | Оценка за курсовой проект |

1. **ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ**
   1. **Порядок контроля**

Контроль выполнения должен осуществляться руководителем курсового проекта (преподавателем) в соответствие с п.7.

* 1. **Порядок приемки**

Приемка должна осуществляться с участием руководителя после проведения приемо-сдаточных испытаний. В результате защиты курсового проекта должна быть выставлена оценка за курсовой проект.

Приложение Б

(справочное)

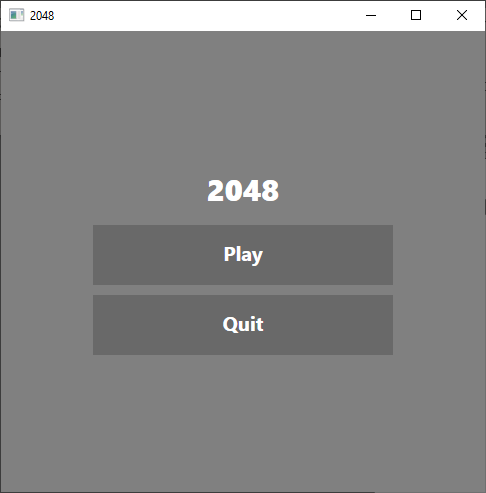


Рисунок Б.1 – Запуск программы

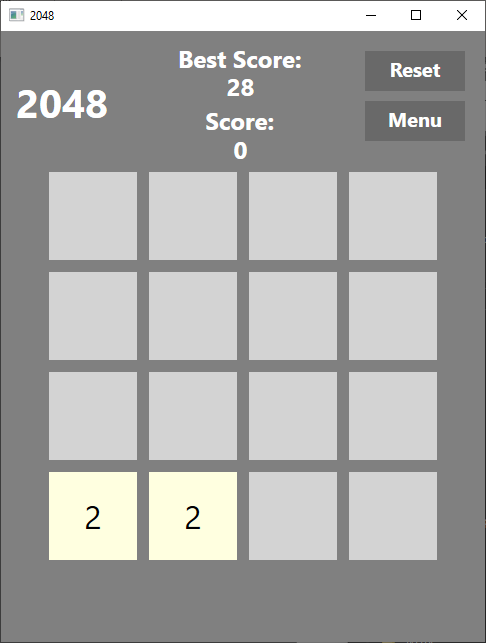


Рисунок Б.2 – Игровое поле

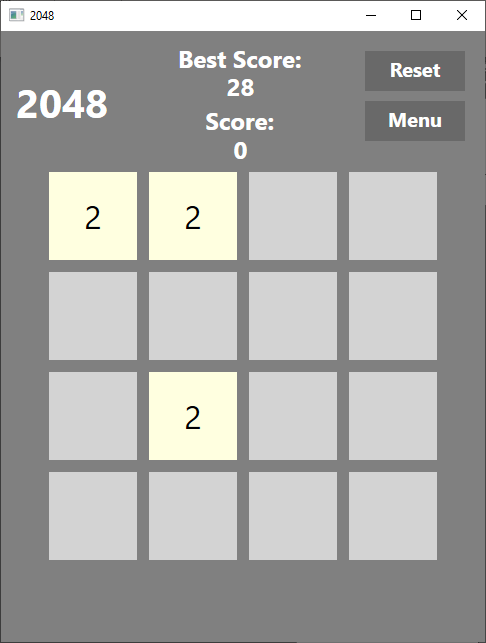


Рисунок Б.3 – Перемещение плиток (вверх)

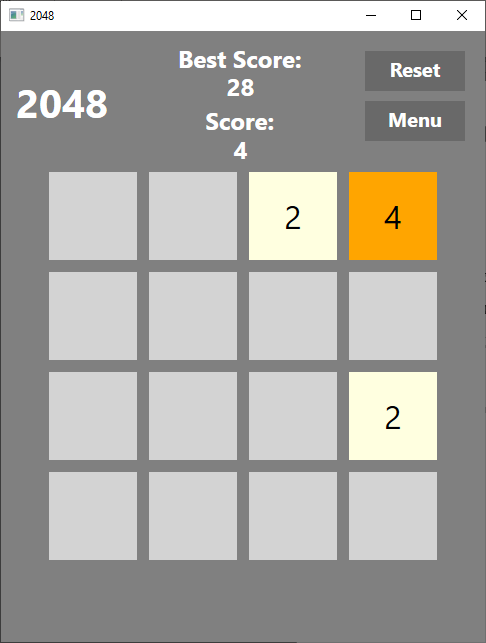


Рисунок Б.4 – Объединение плиток

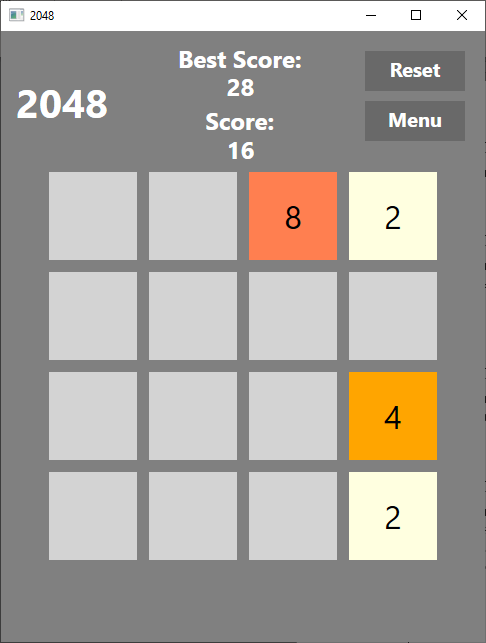


Рисунок Б.5 – Увеличение счета при объединении плиток

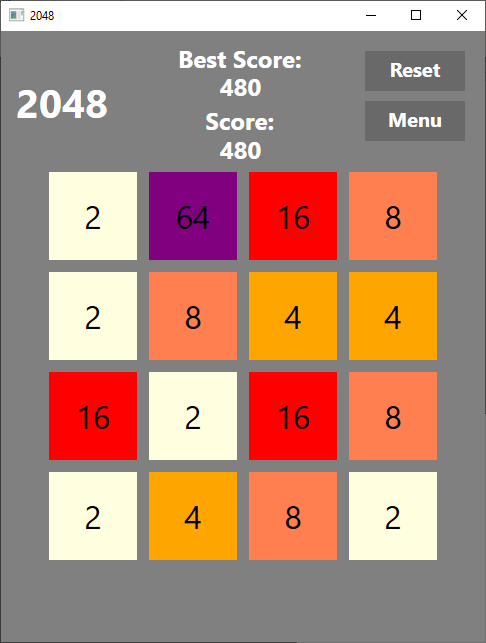


Рисунок Б.6 – Вывод лучшего счета

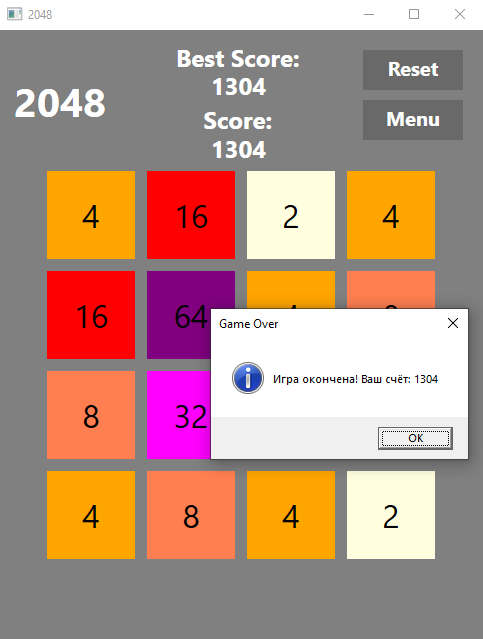


Рисунок Б.7 – Сообщение об окончании игры

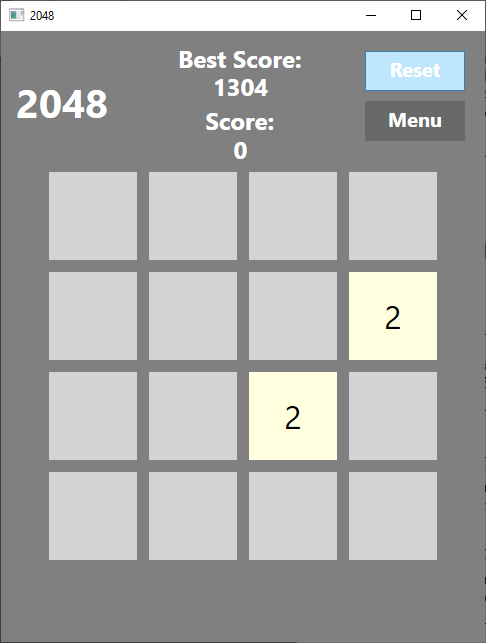


Рисунок Б.8 – Нажатие кнопки «Reset»

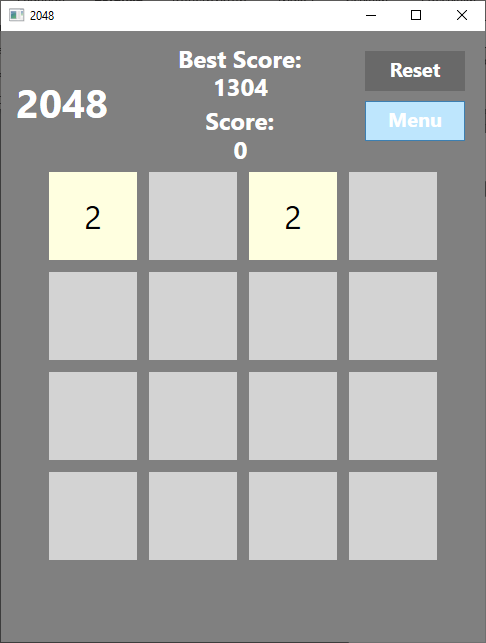


Рисунок Б.9 – Нажатие кнопки «Menu»

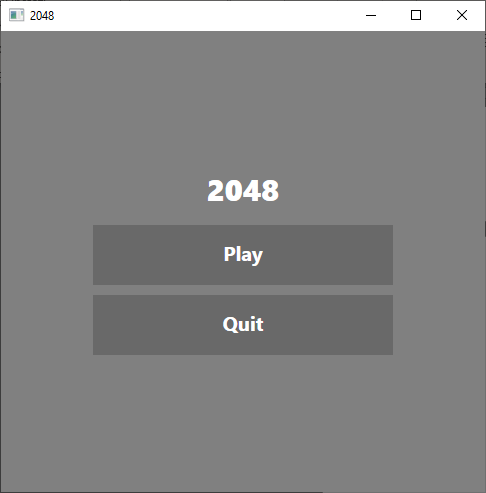


Рисунок Б.10 – Открытое главное меню

Приложение В

(справочное)

**Исходный код GameWindowxaml.cs**

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using System.IO;

namespace Game2048

{

public partial class GameWindow : Window

{

private GameBoard board;

private Label[,] labels;

private int bestScore;

private string scoreFilePath = "bestscore.txt";

public GameWindow()

{

InitializeComponent();

board = new GameBoard();

labels = new Label[GameBoard.Size, GameBoard.Size];

InitGridUI();

DrawBoard();

bestScore = LoadBestScore();

BestScoreText.Text = bestScore.ToString();

this.KeyDown += OnKeyDownHandler;

}

private void InitGridUI()

{

GameGrid.Children.Clear();

for (int i = 0; i < GameBoard.Size; i++)

{

for (int j = 0; j < GameBoard.Size; j++)

{

var label = new Label

{

FontSize = 32,

HorizontalContentAlignment = HorizontalAlignment.Center,

VerticalContentAlignment = VerticalAlignment.Center,

BorderThickness = new Thickness(1),

BorderBrush = Brushes.Gray,

Background = Brushes.LightGray,

Margin = new Thickness(5)

};

labels[i, j] = label;

Grid.SetRow(label, i);

Grid.SetColumn(label, j);

GameGrid.Children.Add(label);

}

}

}

private readonly Dictionary<int, Brush> tileColors = new Dictionary<int, Brush>

{

{ 0, Brushes.LightGray },

{ 2, Brushes.LightYellow },

{ 4, Brushes.Orange },

{ 8, Brushes.Coral },

{ 16, Brushes.Red },

{ 32, Brushes.Magenta },

{ 64, Brushes.Purple },

{ 128, Brushes.Yellow },

{ 256, Brushes.Green },

{ 512, Brushes.Cyan },

{ 1024, Brushes.Blue },

{ 2048, Brushes.Gold }

};

private void DrawBoard()

{

for (int i = 0; i < GameBoard.Size; i++)

{

for (int j = 0; j < GameBoard.Size; j++)

{

int value = board.Tiles[i, j].Value;

var label = labels[i, j];

label.Content = value == 0 ? "" : value.ToString();

label.Background = tileColors.ContainsKey(value) ? tileColors[value] : Brushes.Orange;

}

}

ScoreText.Text = board.Score.ToString();

}

private void OnKeyDownHandler(object sender, KeyEventArgs e)

{

bool moved = false;

switch (e.Key)

{

case Key.Left:

moved = board.MoveLeft();

break;

case Key.Right:

moved = board.MoveRight();

break;

case Key.Up:

moved = board.MoveUp();

break;

case Key.Down:

moved = board.MoveDown();

break;

}

if (moved)

{

board.AddRandomTile();

DrawBoard();

bool has2048 = false;

for (int i = 0; i < GameBoard.Size; i++)

{

for (int j = 0; j < GameBoard.Size; j++)

{

if (board.Tiles[i, j].Value == 2048)

{

has2048 = true;

break;

}

}

if (has2048) break;

}

if (has2048)

{

MessageBox.Show("Поздравляем! Вы собрали плитку 2048!", "Победа!", MessageBoxButton.OK, MessageBoxImage.Information);

}

if (board.Score > bestScore)

{

bestScore = board.Score;

BestScoreText.Text = bestScore.ToString();

SaveBestScore(bestScore);

}

if (board.IsGameOverState)

{

MessageBox.Show($"Игра окончена! Ваш счёт: {board.Score}", "Game Over", MessageBoxButton.OK, MessageBoxImage.Information);

return;

}

}

}

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

{

board.Reset();

ScoreText.Text = "0";

DrawBoard();

}

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

{

MainWindow mainWindow = new MainWindow();

mainWindow.Show();

this.Close();

}

private int LoadBestScore()

{

if (File.Exists(scoreFilePath))

{

string text = File.ReadAllText(scoreFilePath);

if (int.TryParse(text, out int savedScore))

return savedScore;

}

return 0;

}

private void SaveBestScore(int score)

{

File.WriteAllText(scoreFilePath, score.ToString());

}

}

}

**Исходный код GameBoard.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Game2048

{

public class GameBoard

{

public const int Size = 4;

public Tile[,] Tiles { get; private set; }

private Random random = new Random();

public int Score { get; private set; } = 0;

private bool gameStop = false;

public bool IsGameOverState => IsGameOver();

public GameBoard()

{

Tiles = new Tile[Size, Size];

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

{

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

{

Tiles[i, j] = new Tile();

}

}

AddRandomTile();

AddRandomTile();

}

public void AddRandomTile()

{

var emptyTiles = new List<(int row, int col)>();

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

{

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

{

if (Tiles[i, j].IsEmpty)

emptyTiles.Add((i, j));

}

}

if (emptyTiles.Count == 0)

return;

var (row, col) = emptyTiles[random.Next(emptyTiles.Count)];

Tiles[row, col].Value = random.NextDouble() < 0.9 ? 2 : 4;

}

public bool IsGameOver()

{

// Проверка: есть ли хотя бы одна пустая ячейка

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

{

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

{

if (Tiles[i, j].IsEmpty)

return false; // ещё можно ставить

}

}

// Проверка: можно ли объединить соседние клетки

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

{

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

{

int value = Tiles[i, j].Value;

if (i < Size - 1 && Tiles[i + 1, j].Value == value)

return false;

if (j < Size - 1 && Tiles[i, j + 1].Value == value)

return false;

}

}

return true; // игра окончена: нет ни пустых, ни возможных объединений

}

public void Reset()

{

Score = 0;

// Очистка всех плиток

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

{

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

{

Tiles[i, j].Value = 0;

}

}

// Сброс состояния

gameStop = false;

// Добавление двух начальных плиток

AddRandomTile();

AddRandomTile();

}

#region "Движения"

public bool MoveLeft()

{

bool moved = false;

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

{

int[] values = new int[Size];

int index = 0;

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

if (Tiles[i, j].Value != 0)

values[index++] = Tiles[i, j].Value;

int[] merged = new int[Size];

index = 0;

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

{

if (j < Size - 1 && values[j] != 0 && values[j] == values[j + 1])

{

int mergedValue = values[j] \* 2;

merged[index++] = mergedValue;

Score += mergedValue;

j++; // Пропускаем следующую плитку

moved = true;

}

else if (values[j] != 0)

{

merged[index++] = values[j];

}

}

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

{

if (Tiles[i, j].Value != merged[j])

{

Tiles[i, j].Value = merged[j];

moved = true;

}

}

}

return moved;

}

public bool MoveRight()

{

bool moved = false;

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

{

int[] values = new int[Size];

int index = Size - 1;

// Собираем ненулевые значения справа налево

for (int j = Size - 1; j >= 0; j--)

{

if (Tiles[i, j].Value != 0)

values[index--] = Tiles[i, j].Value;

}

// Объединяем и добавляем очки

for (int j = Size - 1; j > 0; j--)

{

if (values[j] != 0 && values[j] == values[j - 1])

{

int mergedValue = values[j] \* 2;

values[j] = mergedValue;

values[j - 1] = 0;

Score += mergedValue; // Добавляем очки

moved = true;

}

}

// Сдвигаем снова

int[] newRow = new int[Size];

index = Size - 1;

for (int j = Size - 1; j >= 0; j--)

{

if (values[j] != 0)

newRow[index--] = values[j];

}

// Применяем

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

{

if (Tiles[i, j].Value != newRow[j])

{

Tiles[i, j].Value = newRow[j];

moved = true;

}

}

}

return moved;

}

public bool MoveUp()

{

bool moved = false;

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

{

int[] values = new int[Size];

int index = 0;

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

{

if (Tiles[i, j].Value != 0)

values[index++] = Tiles[i, j].Value;

}

// Объединяем и добавляем очки

for (int i = 0; i < Size - 1; i++)

{

if (values[i] != 0 && values[i] == values[i + 1])

{

int mergedValue = values[i] \* 2;

values[i] = mergedValue;

values[i + 1] = 0;

Score += mergedValue; // Добавляем очки

moved = true;

}

}

// Сдвигаем снова

int[] newCol = new int[Size];

index = 0;

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

{

if (values[i] != 0)

newCol[index++] = values[i];

}

// Применяем

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

{

if (Tiles[i, j].Value != newCol[i])

{

Tiles[i, j].Value = newCol[i];

moved = true;

}

}

}

return moved;

}

public bool MoveDown()

{

bool moved = false;

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

{

int[] values = new int[Size];

int index = Size - 1;

for (int i = Size - 1; i >= 0; i--)

{

if (Tiles[i, j].Value != 0)

values[index--] = Tiles[i, j].Value;

}

// Объединяем и добавляем очки

for (int i = Size - 1; i > 0; i--)

{

if (values[i] != 0 && values[i] == values[i - 1])

{

int mergedValue = values[i] \* 2;

values[i] = mergedValue;

values[i - 1] = 0;

Score += mergedValue; // Добавляем очки

moved = true;

}

}

// Сдвигаем снова

int[] newCol = new int[Size];

index = Size - 1;

for (int i = Size - 1; i >= 0; i--)

{

if (values[i] != 0)

newCol[index--] = values[i];

}

// Применяем

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

{

if (Tiles[i, j].Value != newCol[i])

{

Tiles[i, j].Value = newCol[i];

moved = true;

}

}

}

return moved;

}

#endregion

}

}

**Исходный код Tile.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Game2048

{

public class Tile

{

public int Value { get; set; }

public bool IsEmpty => Value == 0;

public Tile(int value = 0)

{

Value = value;

}

}

}

**Исходный код UnitTest.cs**

using Game2048;

namespace TestProject1

{

[TestClass]

public class UnitTest1

{

private GameBoard board;

[TestInitialize]

public void Setup()

{

board = new GameBoard();

}

[TestMethod]

public void TestInitialState()

{

// Проверяем, что после инициализации:

// 1. Счет равен 0

Assert.AreEqual(0, board.Score);

// 2. На поле есть ровно 2 плитки

int tileCount = 0;

for (int i = 0; i < GameBoard.Size; i++)

{

for (int j = 0; j < GameBoard.Size; j++)

{

if (!board.Tiles[i, j].IsEmpty)

tileCount++;

}

}

Assert.AreEqual(2, tileCount);

}

[TestMethod]

public void TestMoveLeft()

{

// Подготовка: создаем ситуацию с плитками 2-2-0-0

for (int i = 0; i < GameBoard.Size; i++)

{

for (int j = 0; j < GameBoard.Size; j++)

{

board.Tiles[i, j].Value = 0;

}

}

board.Tiles[0, 0].Value = 2;

board.Tiles[0, 1].Value = 2;

// Действие: двигаем влево

bool moved = board.MoveLeft();

// Проверка:

Assert.IsTrue(moved);

Assert.AreEqual(4, board.Tiles[0, 0].Value); // Должны объединиться в 4

Assert.AreEqual(0, board.Tiles[0, 1].Value); // Вторая плитка должна стать пустой

Assert.AreEqual(4, board.Score); // Счет должен увеличиться на 4

}

[TestMethod]

public void TestReach2048()

{

// Подготовка: создаем ситуацию с двумя плитками 1024

for (int i = 0; i < GameBoard.Size; i++)

{

for (int j = 0; j < GameBoard.Size; j++)

{

board.Tiles[i, j].Value = 0;

}

}

board.Tiles[0, 0].Value = 1024;

board.Tiles[0, 1].Value = 1024;

// Действие: двигаем влево для объединения плиток

bool moved = board.MoveLeft();

// Проверка:

Assert.IsTrue(moved);

Assert.AreEqual(2048, board.Tiles[0, 0].Value); // Должны объединиться в 2048

Assert.AreEqual(0, board.Tiles[0, 1].Value); // Вторая плитка должна стать пустой

Assert.AreEqual(2048, board.Score); // Счет должен увеличиться на 2048

// Проверяем, что игра не окончена, так как есть пустые клетки

Assert.IsFalse(board.IsGameOverState);

}

}

}