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

|  |  |
| --- | --- |
|  | **Допустить к защите** Заместитель директора  по УМР \_\_\_\_\_\_\_\_\_\_\_\_ Е.Г. Конакина\_  (Подпись) (ФИО)  «\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2022 г. |
|  |

**курсовой проект**Тема «Игра с графическим интерфейсом 2048»

специальность 09.02.07 группа 32919/3

Студент (ка) Пешенкова А.В.  
 (подпись) (ФИО)  
Преподаватель Ильин Ю.П.

(подпись) (ФИО)

Санкт-Петербург  
2022

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc102442361)

[1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ 4](#_Toc102442362)

[1.1 Описание предметной области 4](#_Toc102442363)

[1.2 Анализ методов решения 4](#_Toc102442364)

[1.3 Обзор средств программирования 5](#_Toc102442365)

[1.4 Описание языка C# 6](#_Toc102442366)

[1.4.1 Общие сведения 6](#_Toc102442367)

[1.4.2 Элементы языка 7](#_Toc102442368)

[2 ПРАКТИЧЕСКАЯ ЧАСТЬ 11](#_Toc102442369)

[2.1 Постановка задачи 11](#_Toc102442370)

[2.1.1 Основания для разработки 11](#_Toc102442371)

[2.1.2 Назначение программы 11](#_Toc102442372)

[2.2 Проектирование приложения 11](#_Toc102442373)

[2.2.1 Диаграмма прецедентов 11](#_Toc102442374)

[2.2.2 Диаграмма последовательностей 12](#_Toc102442375)

[2.2.3 Диаграмма классов 13](#_Toc102442376)

[2.3 Текст программы 14](#_Toc102442377)

[2.4 Описание программы 14](#_Toc102442378)

[2.4.1 Общие сведения 14](#_Toc102442379)

[2.4.2 Назначение программы 14](#_Toc102442380)

[2.4.3 Описание логической структуры 15](#_Toc102442381)

[2.4.4 Используемые технические и программные средства 16](#_Toc102442382)

[2.4.5 Вызов и загрузка 16](#_Toc102442383)

[2.5 Руководство оператора 16](#_Toc102442384)

[2.5.1 Назначение программы 16](#_Toc102442385)

[2.5.2 Условия выполнения программы 17](#_Toc102442386)

[2.5.3 Выполнение программы и сообщения оператору 17](#_Toc102442387)

[2.6 Программа и методика испытаний 18](#_Toc102442388)

[2.6.1 Объект испытаний 18](#_Toc102442389)

[2.6.2 Цель испытаний 18](#_Toc102442390)

[2.6.3 Требования к программе 18](#_Toc102442391)

[2.6.4 Требования к программной документации 19](#_Toc102442392)

[2.6.5 Средства и порядок испытаний 19](#_Toc102442393)

[2.6.6 Методы испытаний 19](#_Toc102442394)

[2.7 Протокол испытаний 21](#_Toc102442395)

[ЗАКЛЮЧЕНИЕ 24](#_Toc102442396)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 25](#_Toc102442397)

[ПРИЛОЖЕНИЕ А 28](#_Toc102442398)

[ПРИЛОЖЕНИЕ Б 35](#_Toc102442399)

[ПРИЛОЖЕНИЕ В 39](#_Toc102442400)

[ПРИЛОЖЕНИЕ Г 43](#_Toc102442401)

# ВВЕДЕНИЕ

В настоящее время технологии окружают нас повсюду, поэтому неудивительно, что телефон перестал быть исключительно средством связи и теперь используется для развлечения. Просмотр фильмов, прослушивание музыки, игры разных жанров – всё это доступно каждому человеку с помощью смартфона или компьютера.

Большое количество игровых программ способствует повышению интереса пользователей к компьютерным технологиям. Многие игры носят не только развлекательный характер: они повышают скорость реакции, развивают память, внимание, логическое мышление, скорость принятия решения в нестандартных ситуациях. Именно по этой причине актуальность разработки игр в настоящее время постоянно возрастает.

Целью данного курсового проекта является разработка игры – головоломки с графическим интерфейсом «2048». Для выполнения данной работы необходимо выполнить следующие задачи:

* Проанализировать правила игры «2048»;
* Разработать алгоритмы, необходимые для реализации игры «2048»;
* Разработать логику программы;
* Разработать программу;
* Провести тестирование разработанного приложения.

Разработанная программа может быть использована в развлекательных целях.1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ

## 1.1 Описание предметной области

Темой данной курсовой работы является однопользовательская игра «2048» в жанре головоломка.

Головоломка – название жанра компьютерных игр, целью которых является решение логических задач, требующих от игрока использования логики, стратегии и интуиции.

Суть игры «2048» состоит в том, чтобы игрок по полю 4 на 4 передвигал с помощью стрелок на клавиатуре «плитки» с числами в одном из четырёх направлений, после чего будет появляться еще одна плитка с числом два или четыре. Если в результате хода две плитки с одинаковыми числами соединились, будет получаться одна плитка с числом, равным сумме соединённых чисел на плитках. Побеждает игрок, когда получает плитку с числом «2048». Проигрывает же, если на поле не осталось места и возможности соединить плитки.

Игра подходит для всех возрастов, начиная с 3 лет и старше. Игра была создана в 2014 году, она имеет много положительных отзывов и более шестидесяти миллионов установок. Даже существуют версии, разработанные для детей. Чтобы упростить игровой процесс, вместо плиток с цифрами, разработчики используют клетки с графическими изображениями. При этом основная суть игры остается неизменной.

После анализа данного приложения было сформировано мнение, что игры подобного жанра очень интересны аудитории за счёт своей простоты, но в тоже время пользы в виде развития логического мышления, внимательности и стратегического планирования.

## 1.2 Анализ методов решения

Жанр моей игры – головоломка. Головоломка «2048» может быть реализована следующими способами:

1. Ручной способ: в виде настольной игры. Этот способ требует времени на подготовку материалов игры и участия двух человек: первый – игрок, который будет сдвигать плитки в одну из четырёх сторон, и второй, который расставляет новые плитки, следит за правильностью сдвига игроком плиток и ведёт подсчёт очков. Данный способ является весьма трудоёмким, при этом не столь зрелищным и интересным.
2. Автоматизированный способ: создание поля, появление новых плиток, соединение одинаковых плиток, ведение счёта и сохранение лучшего результата происходит автоматически в компьютерной программе, а игрок передвигает с помощью стрелок на клавиатуре плитки в выбранную сторону.
3. Автоматический способ: создание поля, появление новых плиток, соединение одинаковых плиток, ведение счёта, как и в автоматизированном способе, происходит автоматически в программе, однако и движение плиток осуществляется автоматически с помощью программы.

Автоматический способ не может быть использован, так как игра предполагает участие человека. Ручной способ не используется из-за его трудоёмкости и не столь интересного процесса игры.

В результате анализа для данного проекта был выбран автоматизированный способ реализации.

## 1.3 Обзор средств программирования

Требования к курсовому проекту включали пользовательский интерфейс, поэтому при написании данного курсового проекта моя задача состояла в выборе между языками программирования C# и Java. Во многом C# и Java схожи, но было бы неверно отождествлять эти языки.

Основные сходства:

* + Безопасность типов. Ошибка типа возникает, когда тип данных одного объекта ошибочно назначается другому объекту, создавая непреднамеренные побочные эффекты. И C#, и Java работают на то, чтобы гарантировать выявление таких типов незаконных приведений во время компиляции.
  + Сборка мусора: На С# и Java есть встроенная сборка мусора, которая помогает предотвратить утечку памяти путем удаления объектов, которые больше не используются приложением.
  + Одиночное наследование. Оба языка поддерживают одиночное наследование – это означает, что существует только один путь из любого базового класса в любой из его производных классов. Это ограничивает непреднамеренные побочные эффекты, которые могут возникать при наличии нескольких путей между несколькими базовыми классами и производными классами.
  + Интерфейсы. Интерфейс представляет собой абстрактный класс, где все методы абстрактны. Абстрактным методом является тот метод, который объявлен, но не содержит подробностей его реализации. Код, определяющий любые методы или свойства, определенные интерфейсом, должен предоставляться классом, который его реализует. Это помогает избежать двусмысленности паттерна diamond, поскольку всегда ясно, какой базовый класс реализует данный производный класс во время выполнения. Результатом является чистая иерархия линейных классов одиночного наследования в сочетании с некоторой универсальностью множественного наследования. Фактически использование абстрактных классов является одним из способов множественного наследования языков, которые могут преодолеть проблему паттерна diamond.

Основные различия:

* + Windows vs open-source. C# в основном используется в разработке для платформ Microsoft – .NET Framework CLR и является наиболее широко используемой реализацией CLI. На другом конце спектра Java имеет огромную экосистему с открытым исходным кодом и у него открылось второе дыхание отчасти благодаря тому, что Google использует JVM для Android.
  + Поддержка делегатов (указателей): В C# есть делегаты, которые по существу служат в качестве методов, которые могут быть вызваны без знания целевого объекта. Для достижения такой же функциональности в Java вам необходимо использовать интерфейс с одним методом или другим способом обхода, который может потребовать нетривиального количества дополнительного кода, в зависимости от приложения.
  + Проверяемые исключения: Java различает два типа исключений – проверяемые и непроверяемые. C# выбрал более минималистский подход, имея только один тип исключения. Хотя способность ловить исключения может быть полезна, она также может отрицательно влиять на масштабируемость и контроль версий.
  + Полиморфизм: C# и Java используют очень разные подходы к полиморфизму. Java допускает полиморфизм по умолчанию, C# же должен вызывать ключевое слово «virtual» в базовом классе и ключевое слово «override» в производном классе.
  + Перечисления (Enums): в C# перечисления представляют собой простые списки именованных констант, где базовый тип должен быть целым. Java представляет перечисления более глубоко, рассматривая его как именованный экземпляр типа, что упрощает добавление пользовательского поведения к отдельным перечислениям.

Таким образом, средством программирования данного курсового проекта был выбран язык С#, в связи с тем, что он лучше подходит для работы с Windows forms, а средой разработки выбрана Visual Studio 2017.

## 1.4 Описание языка C#

### 1.4.1 Общие сведения

C# - это строго типизированный объектно-ориентированный язык, призванный обеспечить оптимальное сочетание удобства, простоты, выразительности и производительности. На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. На нем пишутся самые различные приложения: от небольших компьютерных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

По сравнению с другими языками C# достаточно молодой. В июне 2000 года компания Microsoft анонсировала платформу .NET и новый язык программирования, получивший название C# (CSharp). Текущей версией языка является версия C# 10.0, которая вышла 8 ноября 2021 года вместе с релизом .NET 6.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если человек знаком с одним из этих языков, то овладеть C# ему будет легче.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. C# продолжает активно развиваться, с каждой новой версией появляется все больше интересного функционала, например, лямбды, динамическое связывание, асинхронные методы и т.д.

### 1.4.2 Элементы языка

#### 1.4.2.1 Алфавит языка C#

Алфавит (или множество литер) языка программирования С# составляют символы таблицы кодов ASCII. Он включает в себя:

* строчные и прописные буквы латинского алфавита;
* цифры от 0 до 9;
* пробельные символы (пробел и символы табуляции);
* символы перевода строки;
* символ «\_» (подчеркивание считается буквой);
* набор специальных символов: ", {, }, [, ], +, —, %, /, \, ', :, ?, <, >, =, !, &, #, ~, \*, -.

#### 1.4.2.2 Константы и переменные

Константы и переменные являются основными объектами, которые используются в программе. Константы и переменные – это именованные объекты, для которых выделяется память.

Значение константы обязательно устанавливается при ее объявлении в программе. В отличие от переменной значение константы не может изменяться программно. Если константа уже объявлена, то при попытке присвоить константе какое-нибудь новое значение компилятор выдаст сообщение об ошибке.

Все переменные и константы должны быть объявлены до их применения. Это нужно, чтобы сообщить компилятору о типе данных, который сохраняется в переменной. Если известен тип переменной, то компилятор осуществляет соответствующую компиляцию любого оператора, в котором используется данная переменная.

Константа объявляется с помощью оператора следующей формы:

*const тип имя\_константы = значение;*

где

* const – ключевое слово, которое определяет, что именно это есть константа;
* тип – тип данных, которые сохраняются в константе;
* имя\_константы – имя константы;
* значение – значение константы, которое есть неизменяемым в программе.

Переменные объявляются с помощью оператора следующей формы:

*тип имя\_переменной;*

где

* тип – это тип данных, которые сохраняются в переменной;
* имя\_переменной – это ее имя.

Инициализация позволяет задать начальное значение переменной при ее объявлении.

Общая форма инициализации переменной имеет следующий вид:

*тип имя\_переменной = значение;*

где

* тип – это тип данных, которые сохраняются в переменной;
* имя\_переменной – это ее имя;
* значение – это конкретное значение, которое задается при создании переменной.

#### 1.4.2.3 Типы данных языка C#

В языке C# есть следующие базовые типы данных:

* bool: хранит значение true или false (логические литералы). Представлен системным типом System.Boolean;
* byte: хранит целое число от 0 до 255 и занимает 1 байт. Представлен системным типом System.Byte;
* sbyte: хранит целое число от -128 до 127 и занимает 1 байт. Представлен системным типом System.SByte;
* short: хранит целое число от -32768 до 32767 и занимает 2 байта. Представлен системным типом System.Int16;
* ushort: хранит целое число от 0 до 65535 и занимает 2 байта. Представлен системным типом System.UInt16;
* int: хранит целое число от -2147483648 до 2147483647 и занимает 4 байта. Представлен системным типом System.Int3. Все целочисленные литералы по умолчанию представляют значения типа int;
* uint: хранит целое число от 0 до 4294967295 и занимает 4 байта. Представлен системным типом System.UInt32;
* long: хранит целое число от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 и занимает 8 байт. Представлен системным типом System.Int64;
* ulong: хранит целое число от 0 до 18 446 744 073 709 551 615 и занимает 8 байт. Представлен системным типом System.UInt64;
* float: хранит число с плавающей точкой от -3.4\*1038 до 3.4\*1038 и занимает 4 байта. Представлен системным типом System.Single;
* double: хранит число с плавающей точкой от ±5.0\*10-324 до ±1.7\*10308 и занимает 8 байта. Представлен системным типом System.Double;
* decimal: хранит десятичное дробное число. Если употребляется без десятичной запятой, имеет значение от ±1.0\*10-28 до ±7.9228\*1028, может хранить 28 знаков после запятой и занимает 16 байт. Представлен системным типом System.Decimal;
* char: хранит одиночный символ в кодировке Unicode и занимает 2 байта. Представлен системным типом System.Char. Этому типу соответствуют символьные литералы:
* string: хранит набор символов Unicode. Представлен системным типом System.String. Этому типу соответствуют строковые литералы;
* object: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе. Представлен системным типом System.Object, который является базовым для всех других типов и классов .NET.

#### 1.4.2.4 Операции и правила составления выражений

Таблица 1

|  |  |  |
| --- | --- | --- |
| Обозначение операции | Название операции | Пример использования операции |
| **«.»** | операция доступа к полю/свойству/методу класса. | /\*someObject – объект какого-то класса, SomeFiled ­– некое поле\*/  someObject.SomeField |
| **«()»** | операция вызова метода | //SomeMethod – некий метод  SomeMethod(); |
| **«[]»** | операция доступа к элементу массива | //someArray – некий массив  someArray[0]; |
| «**++**» и «**--**» после операнда | операции постинкремента и постдекремента | someValue++; //Увеличить значение на 1  someValue--; //Уменьшить значение на 1 |
| «++» и «--» перед операндом | операции преинкремента и предекремента | ++someValue;  --someValue; |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Обозначение операции | Название операции | Пример использования операции |
| «new» | операция создания объекта/массива | /\*SomeType – некий тип данных,  someObject – создаваемый объект,  someArray – создаваемый массив\*/  SomeType someObject = new SomeType(); //Создание нового объекта  SomeType[] someArray = new SomeType[10]; //Создание нового массива |
| «typeof» | операция получения информации о типе данных | /\*SomeType – некое тип данных,  someInfo – объект типа Type\*/  Type someInfo = typeof(SomeType); |
| «-» перед операндом | операция отрицания | int a = 5; int b = -a; //= -5 |
| «!» перед операндом | операция логического отрицания | bool a = true;  bool b = !a; //= false |
| «~» перед операндом | операция двоичного (побитового) отрицания | int a = 10; int b = ~a; |
| «(…)» перед операндом | операция приведения типа | /\*someObject – объект какого-то типа, OtherType – другой тип данных\*/  (OtherType) someObject; |
| «**\***» | операция умножения | int a = 3 \* 2; //=6 |
| «**/**» | операция деления | int a = 6 / 2; //=3 |
| «%» | операция, возвращающая остаток от деления | if(someValue % 2 == 0)  { //Если четное значение }  else  { //Нечетное значение } |
| «**+**» | операция сложения | int a = 15 + 5; //=20 |
| «-» | операция вычитания | int a = 15 – 5; //=10 |
| «**<<**» и «**>>**» | операции побитового сдвига влево и вправо | someValue = someValue << 2; //Сдвиг влево всех битов на две позиции |
| «>» | операция больше | bool a = (4>2); //=true |
| «<» | операция меньше | bool a = (21<4); //=false |
| «>=» | операция больше или равно | bool a = (11>=2); //=true |
| «<=» | операция меньше или равно | bool a = (5<=5); //=true |
| «is» | операция проверки возможности приведения типа | if(someValue is SomeType == true)  { //someValue можно привести к типу SomeType }  else  { //someValue нельзя привести к типу SomeType } |
| «as» | операция преобразования типа, без генерации исключения в случае неудачи | /\*someValue – некая переменная, otherObject – ссылка типа SomeType, SomeType – какой-то тип данных\*/  SomeType otherObject = someValue as SomeType;  if(otheObject != null)  { //someValue преобразовано к типу SomeType и сохранено в otheObject }  else  { //someValue нельзя привести к типу SomeType, otherObject == null } |

Продолжение таблицы 1

|  |  |  |
| --- | --- | --- |
| Обозначение операции | Название операции | Пример использования операции |
| «==» и «!=» | операции эквивалентно и неэквивалентно | bool a = (5 == 7); //=false |
| «&» | операция побитового «И» | bool SecondOperand()  { return true; }  bool a = false & SecondOperand(); //=False |
| «**^**» | операция побитового «ИЛИ» | Console.WriteLine(true ^ true); // False  Console.WriteLine(true ^ false);  //True |
| «|» | операция побитового «ИЛИ» | bool SecondOperand()  { return true; }  bool a = true | SecondOperand(); //=True |
| «&&» | операция условного «И» | /\*firstValue и secondValue –переменные типа bool \*/  if(firstValue && secondValue)  { //Обе переменные имеют значение true  }else  { //Как минимум одна переменная имеет значение false } |
| «**||**» | операция условного «ИЛИ» | if(firstValue || secondValue)  {// Как минимум одна переменная имеет значение true}  else  { //Обе переменные имеют значение false } |
| «??» | операция проверки на null | string restultString = someString ?? "Пустая ссылка";  /\*Если переменная someString == null, то restultString = «Пустая ссылка», в противном случае, restultString = someString \*/ |
| «?» | условная операция | int resultValue = someValue < 0 ? -someValue : someValue;  /\* Если переменная someValue меньше 0, то resultValue = someValue, в противном случае, resultValue = -someValue\*/ |
| «**=**» | операция присваивания | int b = 5; |
| «\*=», «/=», «%=», «+=», «-=», «<<=», «>>=», «&=», «^=», «|=» | операции составного присваивания | someValue += 5; //someValue = someValue + 5; |

#### 1.4.2.5 Операторы языка C#

Операторы ветвления:

* условный оператор if
* оператор выбора switch

Операторы цикла:

* цикл с предусловием while
* цикл с постусловием do {} while()
* цикл с параметром for
* цикл перебора foreach

Операторы передачи управления:

* оператор безусловного перехода goto
* оператор выхода из цикла break
* оператор перехода к следующей итерации цикла continue
* оператор возврата return
* оператор генерации исключения throw

#### 1.4.2.6 Способы структурирования программы

Программа на языке C# состоит из одного или нескольких файлов. Каждый файл может содержать или не содержать пространства имен. Пространство имен может содержать типы, такие как классы, структуры, интерфейсы, перечисления и делегаты или другие пространства имен.

# 2 ПРАКТИЧЕСКАЯ ЧАСТЬ

## 2.1 Постановка задачи

### 2.1.1 Основания для разработки

Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» и утверждена ФГАОУ ВО СПБПУ ИСПО.

### 2.1.2 Назначение программы

Игровая программа с графическим интерфейсом «2048»: игровое поле имеет форму квадрата 4x4, в каждой клетке которого может содержаться плитка с числом - некоторой степенью двойки. Ход заключается в том, что игрок переносит все плитки в одну из сторон (указывая направление с клавиатуры), при этом, если плитки с одинаковыми числами столкнулись друг с другом, то они объединяются в одну, число на которой равно сумме чисел объединившихся плиток, т.е. следующей степени двойки. Для победы в игре необходимо получить плитку с числом «2048».

Полный текст технического задания приведен в приложении А.

## 2.2 Проектирование приложения

На этапе проектирования были разработаны диаграмма прецедентов, диаграмма последовательностей и диаграмма классов.

### 2.2.1 Диаграмма прецедентов

Диаграмма вариантов прецедентов – это тип поведенческой диаграммы UML, который часто используется для анализа различных систем. Они позволяют визуализировать различные типы ролей в системе и то, как эти роли взаимодействуют с системой.

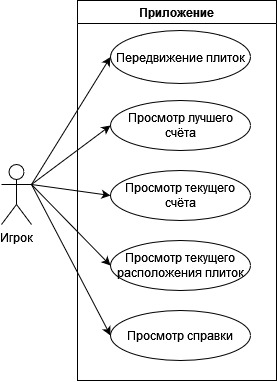


Рисунок 1 – Диаграмма прецедентов

### 2.2.2 Диаграмма последовательностей

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций. На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

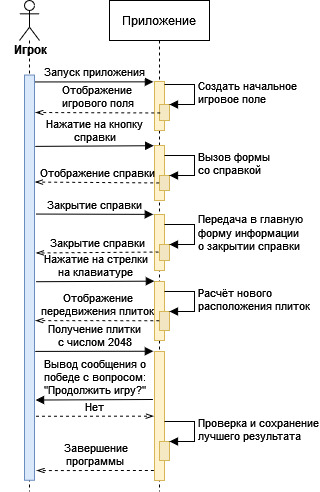


Рисунок 2 Диаграмма последовательностей

### 2.2.3 Диаграмма классов

На рисунке 3 показана диаграмма классов проекта. Программа содержит 5 классов: Form1-класс формы игры, класс Help\_window содержит форму справки, класс Model содержит методы, описывающие движение плиток и проверку окончания игры, класс EmptyTile создан для хранения координат пустых клеток, класс Map\_of\_tile хранит информацию о каждой плитке.

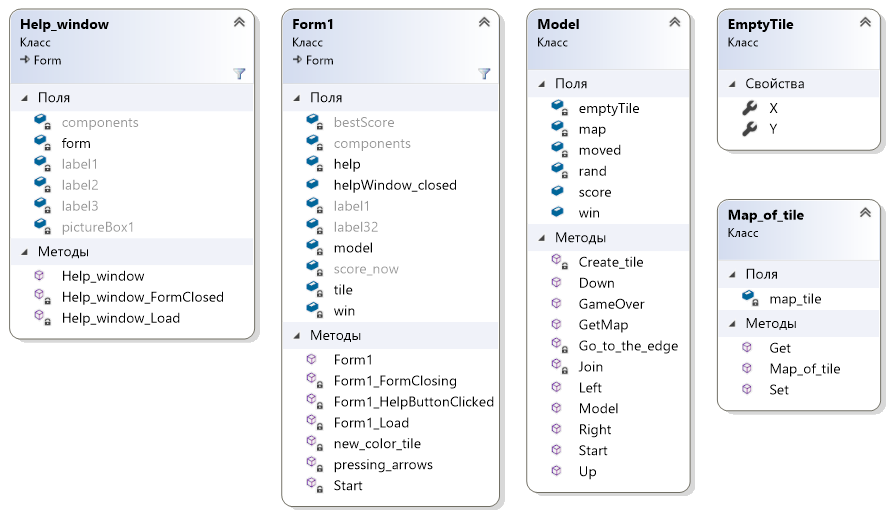


Рисунок 3 Диаграмма классов

## 2.3 Текст программы

Текст программы в соответствии с ГОСТ 19.101-77 (СТ СЭВ 1626-79) и ГОСТ 19.401-79 (СТ СЭВ 3746-82) представляет собой запись программы на исходном языке программирования с необходимыми комментариями. Текст программы представляет собой документ, выполненный машинным способом, и приведен в приложении Г.

## 2.4 Описание программы

### 2.4.1 Общие сведения

Игровая программа с графическим интерфейсом «2048»: человек решает головоломку, передвигая с помощью стрелок на клавиатуре плитки по полю для получения плитки с числом 2048. После каждого хода появляется на случайном месте плитка с числом 2 или 4. Программа написана на языке C# и работает под управлением операционной системы Windows 7.

### 2.4.2 Назначение программы

Основное назначение программного продукта заключается в организации игры с графическим интерфейсом 2048»: человек решает головоломку, передвигая с помощью стрелок на клавиатуре плитки по полю для получения плитки с числом 2048. После каждого хода появляется на случайном месте плитка с числом 2 или 4. Проигрыш происходит тогда, когда поле полностью заполнено и нет возможности соединить соседние плитки.

### 2.4.3 Описание логической структуры

Программа содержит 5 классов:

Form1-класс формы игры, класс Help\_window содержит форму справки, класс Model содержит методы, описывающие движение плиток и проверку окончания игры, класс EmptyTile создан для хранения координат пустых клеток, класс Map\_of\_tile хранит информацию о каждой плитке.

Схемы отдельных методов приведены в приложении В.

#### 2.4.3.1 Описание методов класса Form1

private void Start() метод, который подготавливает игровое поле.

private void new\_color\_tile(int i, int j) метод, меняющий цвет плитки в зависимости от числа на ней.

private void pressing\_arrows(object sender, KeyEventArgs e) метод, который срабатывает при нажатии на стрелки на клавиатуре. В нём происходит движение плиток, проверка на окончание игры, вывод сообщений о победе или проигрыше и вывод счёта с помощью методов, описанных в классе Model.

private void Form1\_FormClosing(object sender, FormClosingEventArgs e) метод, срабатывающий при закрытии формы. В нём сохраняется лучший результат игры.

private void Form1\_HelpButtonClicked(object sender, System.ComponentModel.CancelEventArgs e) метод, срабатывающий при нажатии на кнопку вызова справки.

#### 2.4.3.2 Описание методов класса Help\_window

private void Help\_window\_Load(object sender, EventArgs e) метод, происходящий при загрузке формы. В нём задаётся её владелец.

private void Help\_window\_FormClosed(object sender, FormClosedEventArgs e) метод, происходящий при закрытии формы. Передаёт в основную форму данные о том, что справка закрыта.

#### 2.4.3.3 Описание методов класса Map\_of\_tile

public Map\_of\_tile() конструктор, выделяющий память под массив чисел на плитках.

public int Get(int x, int y) метод, возвращающий число на указанной плитке.

public void Set(int x, int y, int num) метод, устанавливающий число на указанную плитку.

#### 2.4.3.4 Описание методов класса Model

public Model() конструктор, выделяющий память под объект класса Map\_of\_tile.

public void Start() метод, устанавливающий значения для начала игры.

private void Create\_tile() метод, создающий новые плитки.

void Go\_to\_the\_edge(int x, int y, int newX, int newY) метод, осуществляющий сдвиг плиток.

void Join(int x, int y, int newX, int newY) метод, осуществляющий соединение плиток.

public void Right() метод, осуществляющий движение плиток вправо.

public void Left() метод, осуществляющий движение плиток влево.

public void Up() метод, осуществляющий движение плиток вверх.

public void Down() метод, осуществляющий движение плиток вниз.

public bool GameOver() метод, проверяющий условия поражения.

public int GetMap(int i, int j) метод, возвращающий число с указанной плитки.

### 2.4.4 Используемые технические и программные средства

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

* процессор Intel или другой совместимый;
* объем свободной оперативной памяти ~25 Мб;
* объем необходимой памяти на жестком диске ~2Мб;
* стандартный VGA-монитор или совместимый;
* стандартная клавиатура;
* манипулятор «мышь».

### 2.4.5 Вызов и загрузка

Программа может быть загружена как с флэш-накопителя, так и с жесткого диска. В последнем случае требуется предварительно переписать программу с флэш-накопителя на жесткий диск.

Исполняемым файлом программы является файл 2048.exe. Для его запуска необходимо дважды щелкнуть по исполняемому файлу левой кнопкой мышки.

## 2.5 Руководство оператора

### 2.5.1 Назначение программы

Основное назначение программного продукта заключается в организации игры с графическим интерфейсом 2048»: человек решает головоломку, передвигая с помощью стрелок на клавиатуре плитки по полю для получения плитки с числом 2048. После каждого хода появляется на случайном месте плитка с числом 2 или 4. Проигрыш происходит тогда, когда поле полностью заполнено и нет возможности соединить соседние плитки.

### 2.5.2 Условия выполнения программы

#### 2.5.2.1 Технические средства

Для нормального функционирования данной информационной системы необходим компьютер и следующие технические средства:

* процессор Intel или другой совместимый;
* объем свободной оперативной памяти ~25 Мб;
* объем необходимой памяти на жестком диске ~2Мб;
* стандартный VGA-монитор или совместимый;
* стандартная клавиатура;
* манипулятор «мышь».

#### 2.5.2.1 Программные средства

Для нормального функционирования данной информационной системы необходимы следующие программные средства:

* операционная система Windows 7 или совместимая;
* исполняемый файл программы 2048.exe.

### 2.5.3 Выполнение программы и сообщения оператору

Для запуска программы дважды щелкните левой кнопкой мыши по исполняемому файлу 2048.exe.

Процесс игры, этапы работы приложения показаны в виде иллюстраций в приложении Б.

При запуске работы приложения загружается главная форма, на которой находится игровое поле 4х4 и две плитки с числами 2 или 4. Пользователь с помощью стрелок на клавиатуре может передвигать плитки в одну из четырёх сторон. В случае, когда две плитки с одинаковыми числами «сталкиваются» во время хода, они сливаются в одну и их числа складываются. Целью игры является получение плитки с числом 2048. Когда пользователь получает её, появляется сообщение о победе с предложением продолжить игру для получения больших чисел. В случае, когда всё поле занято плитками и нет возможности их соединить, выводится сообщение о поражении с предложением попробовать сыграть заново.

Вверху основной формы находятся текстовые поля с текущим счётом и лучшим счётом. Счёт является суммой чисел, которые появлялись на новых плитках в результате слияния.

При нажатии на кнопку со знаком вопроса появляется форма со справкой, содержащей краткие правила и сведения об игре.

## 2.6 Программа и методика испытаний

### 2.6.1 Объект испытаний

Объектом испытаний является игровая программа 2048.exe. Человек решает головоломку, передвигая с помощью стрелок на клавиатуре плитки по полю для получения плитки с числом 2048. После каждого хода появляется на случайном месте плитка с числом 2 или 4. Проигрыш происходит тогда, когда поле полностью заполнено и нет возможности соединить соседние плитки.

### 2.6.2 Цель испытаний

Целью испытаний является проверка соответствия программы требованиям Технического Задания.

### 2.6.3 Требования к программе

В процессе испытаний подлежат проверке следующие требования к программе:

#### 2.6.3.1 Требования к функциональным характеристикам

* В начале игры на поле в случайном порядке должны находиться две плитки с цифрами «2»;
* При столкновении двух плиток с одинаковыми числами они должны соединяться, удваивая число;
* Во время хода все плитки перемещаются в одну, выбранную игроком, сторону;
* Для победы нужно собрать плитку с числом «2048»;
* После каждого хода на случайной пустой клетке появляется новая плитка номиналом «2» или «4»;
* Ведётся подсчёт очков;
* Происходит сохранение наилучшего результата;
* Программа должна содержать справку;
* Игра заканчивается, если у игрока не осталось ходов (все клетки заполнены плитками, возможностей их объединить нет).

#### 2.6.3.2 Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы Microsoft Windows 7 или совместимой. Язык интерфейса – русский.

#### 2.6.3.3 Требования к маркировке и упаковке

Программа должна поставляться на флеш – накопителе в виде исполняемого (еxе) файла, документации и проекта. Флеш - накопитель должен быть упакован в конверт с подписью «Игра с графическим интерфейсом «2048»» и указанием ФИО разработчика с годом выпуска.

### 2.6.4 Требования к программной документации

На испытания должны быть представлены следующие программные документы:

* техническое задание;
* текст программы;
* описание программы;
* руководство оператора;
* описание языка.

### 2.6.5 Средства и порядок испытаний

Для проведения испытаний необходимы компьютер и следующие технические средства:

* процессор Intel или другой совместимый;
* объем свободной оперативной памяти ~25 Мб;
* объем необходимой памяти на жестком диске ~2Мб;
* стандартный VGA-монитор или совместимый;
* стандартная клавиатура;
* манипулятор «мышь».

Для проведения испытаний необходимы следующие программные средства:

Операционная система Windows 7.

Испытания проводятся в следующем порядке:

1. проверяется наличие и комплектность программной документации (п.2.6.4);
2. проверяется соответствие требованиям к маркировке и упаковке (п.2.6.3.3);
3. проверяется соответствие требованиям к функциональным характеристикам (п.2.6.3.1);
4. проверяется соответствие требованиям к информационной и программной совместимости (п.2.6.3.2).

### 2.6.6 Методы испытаний

2.6.6.1 Для проверки того, что в начале игры на поле будут находиться две плитки с цифрами «2», необходимо:

* запустить программу;
* убедиться, что на поле находятся две плитки с цифрами «2».

2.6.6.2 Для проверки того, что при столкновении двух плиток с одинаковыми числами они будут соединяться, удваивая число, необходимо:

* запустить программу;
* сдвинуть плитки таким образом, чтобы они в следующий ход соединились;
* соединить плитки;
* убедиться, что число на полученной плитке равно сумме чисел тех, что были соединены.

2.6.6.3 Для проверки того, что все плитки сдвигаются в одну сторону, необходимо:

* запустить программу;
* нажать любую стрелку на клавиатуре;
* убедиться, что все плитки были сдвинуты в одну сторону.

2.6.6.4 Для проверки того, что по достижению плитки с числом «2048» появится сообщение о победе, необходимо:

В связи со сложностью получения плитки с числом «2048» осуществление проверки нужно провести с плиткой с числом «256», заранее изменив в коде условие победы. Для проверки требуется:

* запустить программу;
* соединять плитки до тех пор, пока не будет получена плитка с числом «256»;
* убедиться в том, что сообщение о победе было выведено.

2.6.6.5 Для проверки появления после хода новой плитки с числом «2» или «4», необходимо:

* запустить программу;
* сдвинуть плитки в любую сторону;
* убедиться в том, что количество плиток на поле стало на одну больше, или если плитки, которые появились в начале игры, были соединены, то количество плиток осталось равным двум.

2.6.6.6 Для проверки ведения счёта, необходимо:

* запустить программу;
* соединить плитки;
* убедиться, что количество очков увеличилось на значение, которое появилось на полученной в результате соединения плитке.

2.6.6.7 Для проверки сохранения лучшего результата, необходимо:

* запустить программу;
* достичь счёта, который больше того, что указан в поле «Лучший счёт»;
* закрыть программу;
* запустить программу;
* убедиться, что в поле «Лучший результат» написан тот счёт, который был достигнут в прошлой игре.

2.6.6.8 Для проверки того, что в программе есть справка, необходимо:

* запустить программу;
* нажать на кнопку вверху формы со знаком вопроса;
* убедиться, что была открыта справка.

2.6.6.9 Для проверки того, что игра оканчивается тогда, когда поле заполнено и нет возможности соединить соседние плитки, необходимо:

* запустить программу;
* произвольно передвигать плитки до тех пор, пока не будет заполнено поле;
* в случае, если есть возможность соединить плитки – соединить их;
* повторять ходы до тех пор, пока для этого есть возможность;
* когда не будет возможности выполнить ход, убедиться, что появилось сообщение о поражении.

## 2.7 Протокол испытаний

Результаты испытаний программы представлены в таблице 2, рисунки приведены в приложении Б.

**Результаты испытаний программы**

Таблица 2

|  |  |  |
| --- | --- | --- |
| **Проверяемые требования** | **Сообщения программы и вводимые значения** | **Результаты** |
| В начале игры на поле будут находиться две плитки с цифрами «2» | * запустить программу; * убедиться, что на поле находятся две плитки с цифрами «2». | На поле 4х4 присутствуют две плитки с цифрой «2» (рисунок 4). |
| При столкновении двух плиток с одинаковыми числами они будут соединяться | * запустить программу; * сдвинуть плитки таким образом, чтобы они в следующий ход соединились; * соединить плитки; * убедиться, что число на полученной плитке равно сумме чисел тех, что были соединены. | В результате соединения двух плиток с числами «2» появилась новая плитка с числом «4» (см. рисунки 4 и 5). |
| Все плитки сдвигаются в одну сторону | * запустить программу; * нажать любую стрелку на клавиатуре; * убедиться, что все плитки были сдвинуты в одну сторону. | Плитки сдвинуты в ту сторону, которая была выбрана с помощью стрелки на клавиатуры (см рисунок 5). |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| **Проверяемые требования** | **Сообщения программы и вводимые значения** | **Результаты** |
| При достижении плитки с числом «2048» появление сообщения о победе | * Заранее изменить в коде условие победы на достижение числа «256»; * запустить программу; * соединять плитки до тех пор, пока не будет получена плитка с числом «256»; * убедиться в том, что сообщение о победе было выведено. | Сообщение о победе выводится (рисунок 6). |
| Появление после хода новой плитки с числом «2» или «4» | * запустить программу; * сдвинуть плитки в любую сторону; * убедиться в том, что количество плиток на поле стало на одну больше, или если плитки, которые появились в начале игры, были соединены, то количество плиток осталось равным двум. | Появляется новая плитка с числом «2» или «4» после каждого поля, если есть доступные для этого места (см. рисунок 5). |
| Ведения счёта | * запустить программу; * соединить плитки; * убедиться, что количество очков увеличилось на значение, которое появилось на полученной в результате соединения плитке. | Увеличение счёта на полученное число происходит (см. рисунок 5) |
| Сохранение лучшего результата | * запустить программу; * достичь счёта, который больше того, что указан в поле «Лучший счёт»; * закрыть программу; * запустить программу; * убедиться, что в поле «Лучший результат» написан тот счёт, который был достигнут в прошлой игре. | Лучший результат сохраняется корректно (рисунки 7 и 8). |
| В программе есть справка | * запустить программу; * нажать на кнопку вверху формы со знаком вопроса; * убедиться, что была открыта справка. | Справка присутствует (рисунок 9). |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| **Проверяемые требования** | **Сообщения программы и вводимые значения** | **Результаты** |
| Игра оканчивается тогда, когда поле заполнено и нет возможности соединить соседние плитки | * запустить программу; * произвольно передвигать плитки до тех пор, пока не будет заполнено поле; * в случае, если есть возможность соединить плитки – соединить их; * повторять ходы до тех пор, пока для этого есть возможность; * когда не будет возможности выполнить ход, убедиться, что появилось сообщение о поражении. | Сообщение выводится. (рисунок 10). |

# ЗАКЛЮЧЕНИЕ

Разработанная в ходе выполнения курсового проекта программа удовлетворяет всем требованиям технического задания, что подтверждается протоколом испытаний.

Разработанная программа может быть использована в развлекательных целях, а также для развития логического мышления, внимательности и стратегического планирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А. Хейлсберг, М. Торгерсен, С. Вилтамут, П. Голд ., Язык программирования C#, 4-е издание, СПб:-ПИТЕР, 2011
2. Фленов М., Библия C#, 3-е издание, СПб:-БХВ, 2016
3. Культин Н., Microsoft Visual C# в задачах и примерах, 2-е издание, СПб:-БХВ, 2015
4. Стандарты Единой Системы Программной Документации:

ГОСТ 19.105-78 Общие требования к программным документам

ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом

ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению

ГОСТ 19.301-78 Программа и методика испытаний. Требования к содержанию и оформлению

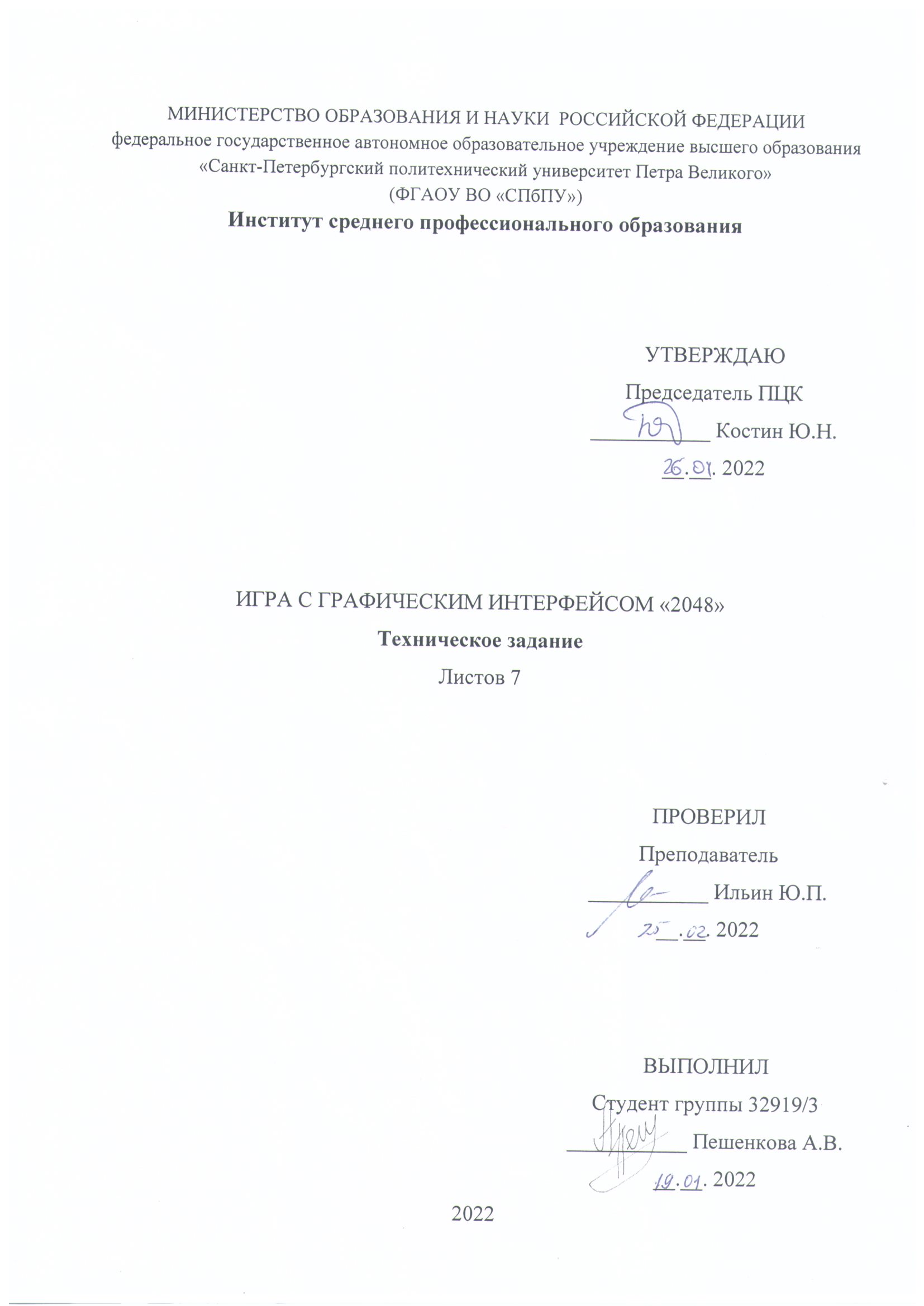
ГОСТ 19.401-78 Текст программы. Требования к содержанию и оформлению

ГОСТ 19.402-78 Описание программы. Требования к содержанию и оформлению

ГОСТ 19.505-79 Руководство оператора. Требования к содержанию и оформлению

ГОСТ 19.701-90 Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения

ПРИЛОЖЕНИЕ А



1 ВВЕДЕНИЕ

* 1. Полное наименование программной разработки: «Игра с графическим интерфейсом «2048»».
  2. Игровая программа с графическим интерфейсом «2048»: игровое поле имеет форму квадрата 4x4, в каждой клетке которого может содержаться плитка с числом - некоторой степенью двойки. Ход заключается в том, что игрок переносит все плитки в одну из сторон (указывая направление с клавиатуры), при этом, если плитки с одинаковыми числами столкнулись друг с другом, то они объединяются в одну, число на которой равно сумме чисел объединившихся плиток, т.е. следующей степени двойки. После каждого хода в пустой клетке появляется новая плитка с числом «2» или «4». Игра заканчивается, если невозможно совершить ход (т.е. всё поле занято плитками, и соединить их невозможно).
  3. В соответствии с заданием программный продукт состоит из теоретической и практической частей. Теоретическая часть включает подробное описание работы с приложением, практическая часть включает разработку и реализацию с использованием среды программирования C# программных модулей программного продукта.
  4. Программа предназначена для развлекательных целей и развития логического мышления. Программа должна продемонстрировать навыки, полученные в ходе изучения дисциплины «Разработка программных модулей программного обеспечения для компьютерных систем».

ОСНОВАНИЕ ДЛЯ РАЗРАБОТКИ

2.1. Разработка ведётся на основании задания к курсовому проекту по профессиональному модулю ПМ.01 «Разработка программных модулей программного обеспечения для компьютерных систем» МДК 01.01 «Прикладное программирование» и утверждена Институтом среднего профессионального образования. Для разработки используется среда программирования Visual Studio 2017 под управлением операционной системы Windows 7.

НАЗНАЧЕНИЕ РАЗРАБОТКИ

* 1. Основное назначение программного продукта заключается в организации компьютерной игры с графическим интерфейсом «2048»: игровое поле имеет форму квадрата 4x4, в каждой клетке которого может содержаться плитка с числом - некоторой степенью двойки. Ход заключается в том, что игрок переносит все плитки в одну из сторон (указывая направление с клавиатуры), при этом, если плитки с одинаковыми числами столкнулись друг с другом, то они объединяются в одну, число на которой равно сумме чисел объединившихся плиток, т.е. следующей степени двойки. После каждого хода в пустой клетке появляется новая плитка с цифрой «2» или «4». Игра заканчивается, если невозможно совершить ход (т.е. всё поле занято плитками, и соединить их невозможно).

4 ТРЕБОВАНИЯ К РАЗРАБОТКЕ

4.1 Требования к функциональным характеристикам:

1 версия:

* Программа должна иметь игровое поле 4×4;
* Программа должна предоставлять пользователю графический интерфейс;
* В начале игры на поле в случайном порядке должны находиться две плитки с цифрами «2»;
* При столкновении двух плиток с одинаковыми числами они должны соединяться, удваивая число;
* За один ход одна плитка может соединиться с другой только один раз;
* Во время хода все плитки перемещаются в одну, выбранную игроком, сторону;
* После каждого хода на случайной пустой клетке появляется новая плитка номиналом «2»;
* Для победы нужно собрать плитку с числом «2048»;
* Игра заканчивается, если у игрока не осталось ходов (все клетки заполнены плитками, возможностей их объединить нет) или получена плитка с числом «2048».

2 версия:

* После каждого хода на случайной пустой клетке появляется новая плитка номиналом «2» (с вероятностью 90%) или «4» (с вероятностью 10%);
* Возможно продолжение игры после получения плитки номиналом «2048» с целью получение больших чисел;
* Ведётся подсчёт очков – суммы всех соединённых за игру плиток;
* Происходит сохранение наилучшего результата;
* Программа должна содержать справку;
* Плитки имеют определённый цвет, зависящий от числа на плитке.

4.2 Требования к надежности:

* использование лицензированного программного обеспечения;
* проверка на вирусы;
* обработка исключительных ситуаций.

4.3 Требования к составу и параметрам технических средств

Для нормального функционирования данной информационной системы необходим компьютер, клавиатура, мышь и следующие технические средства:

- процессор Intel или другой совместимый;

- объем свободной оперативной памяти: 512 Мб;

- объем необходимой памяти на жестком диске 512 Мб;

- стандартный VGA-монитор или совместимый;

- стандартная клавиатура;

- манипулятор «мышь».

4.4 Требования к информационной и программной совместимости

Для полноценного функционирования данной системы необходимо наличие операционной системы Microsoft Windows 7 или совместимой. Язык интерфейса – русский.

4.5 Требования к маркировке и упаковке

Программа должна поставляться на флеш – накопителе в виде исполняемого (еxе) файла, документации и проекта. Флеш - накопитель должен быть упакован в конверт с подписью «Игра с графическим интерфейсом «2048»» и указанием ФИО разработчика с годом выпуска.

4.6 Требования к транспортировке и хранению

Флеш - накопитель с программой должен храниться вдали от электромагнитных полей и не подвергаться механической деформации. Место и условия хранения должны соответствовать санитарным требованиям отрасли. Сроки хранения устанавливаются в соответствии с гарантийными сроками поставщика флеш - накопителей.

Основные требования к транспортировке – создание условий, исключающих механические повреждения флеш - накопителя.

5 ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

5.1 Предварительный состав программной документации:

- «Техническое задание»;

- разрабатываемые программные модули должны быть самодокументированы, т.е. тексты программ должны содержать все необходимые комментарии;

- разрабатываемое программное обеспечение должно включать справочную систему.

5.2 Перечень материалов пояснительной записки

Введение

1. Теоретические основы разработки
   1. Описание предметной области
   2. Анализ методов решения
   3. Обзор средств программирования
   4. Описание выбранного языка программирования
2. Практическая часть
   1. Постановка задачи
   2. Описание схем
   3. Текст программы
   4. Описание программы
   5. Руководство оператора
   6. Программа и методика испытаний
   7. Протокол испытаний

Заключение

Список использованных источников

Приложения

6 ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Технико-экономические показатели не рассчитываются.

7 СТАДИИ И ЭТАПЫ РАЗРАБОТКИ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Содержание стадии** | **Содержание этапа** | **Срок 2022 г.** | | **Форма  отчетности** |
| **начало** | **конец** |
| Техническое задание | Составление технического задания | 10.01 | 19.01 | Техническое задание |
| Эскизный проект | Разработка спецификаций | 20.01 | 05.02 | Спецификации программного обеспечения |
| Рабочий проект | Проектирование программы | 06.02 | 14.02 | Схема работы системы и спецификации компонентов |
| Составление программы | 15.02 | 16.03 | Программная документация |
| Приёмо-сдаточные испытания | 17.03 | 01.04 | Протокол испытаний (п. 2.7 пояснительной записки) |
| Приёмка | Защита курсового проекта | 02.04 | 21.04 | Оценка за курсовой проект |

8 ПОРЯДОК КОНТРОЛЯ И ПРИЕМКИ

8.1 Порядок контроля

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

8.2 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



Рисунок 4 – Запуск приложения

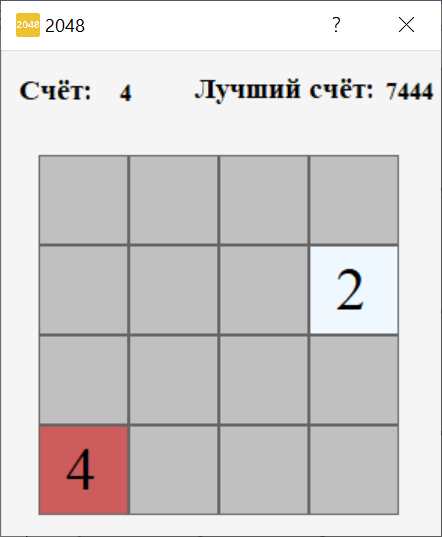


Рисунок 5 – Соединение плиток движением вниз

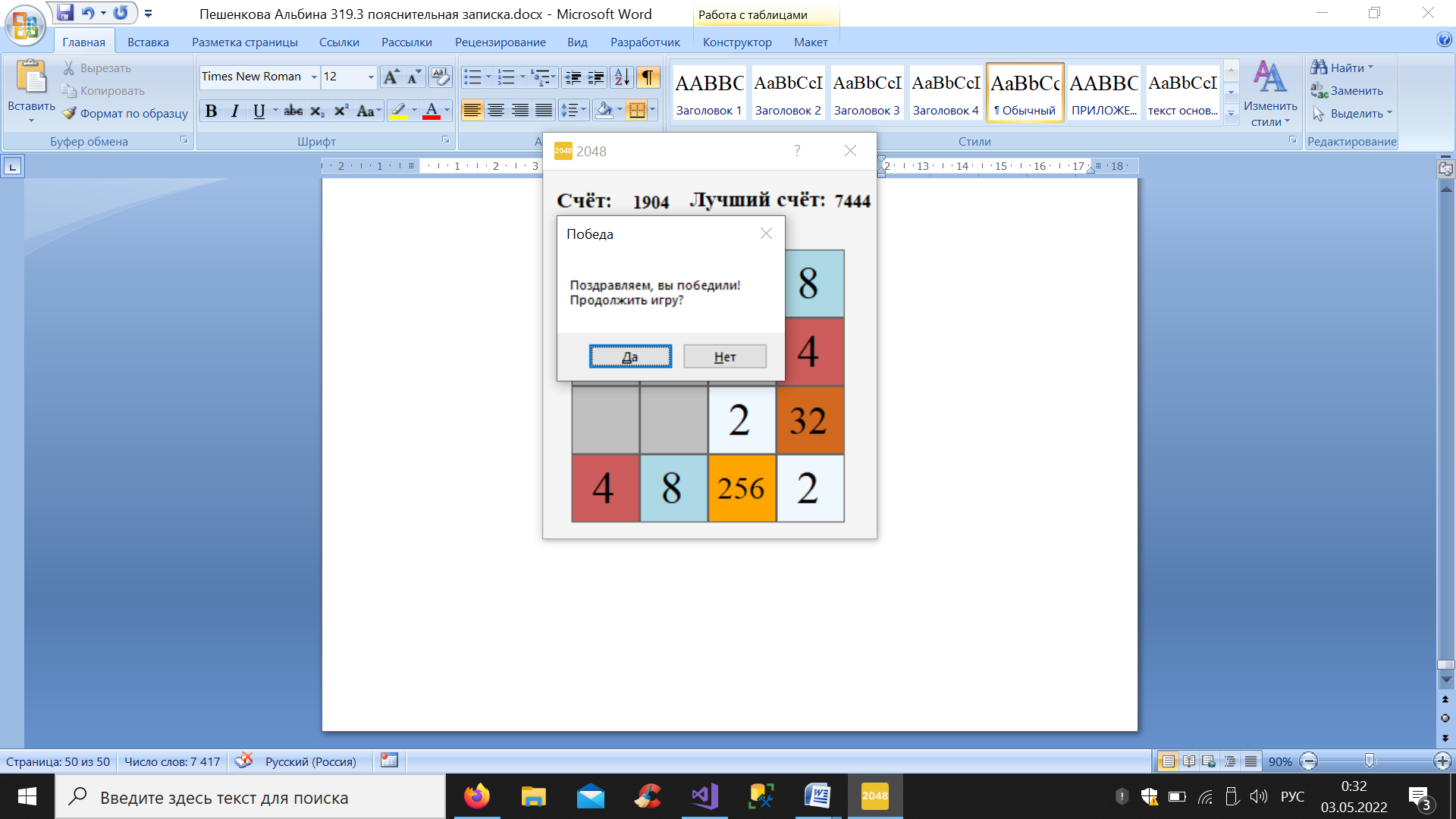


Рисунок 6 – Сообщение о победе

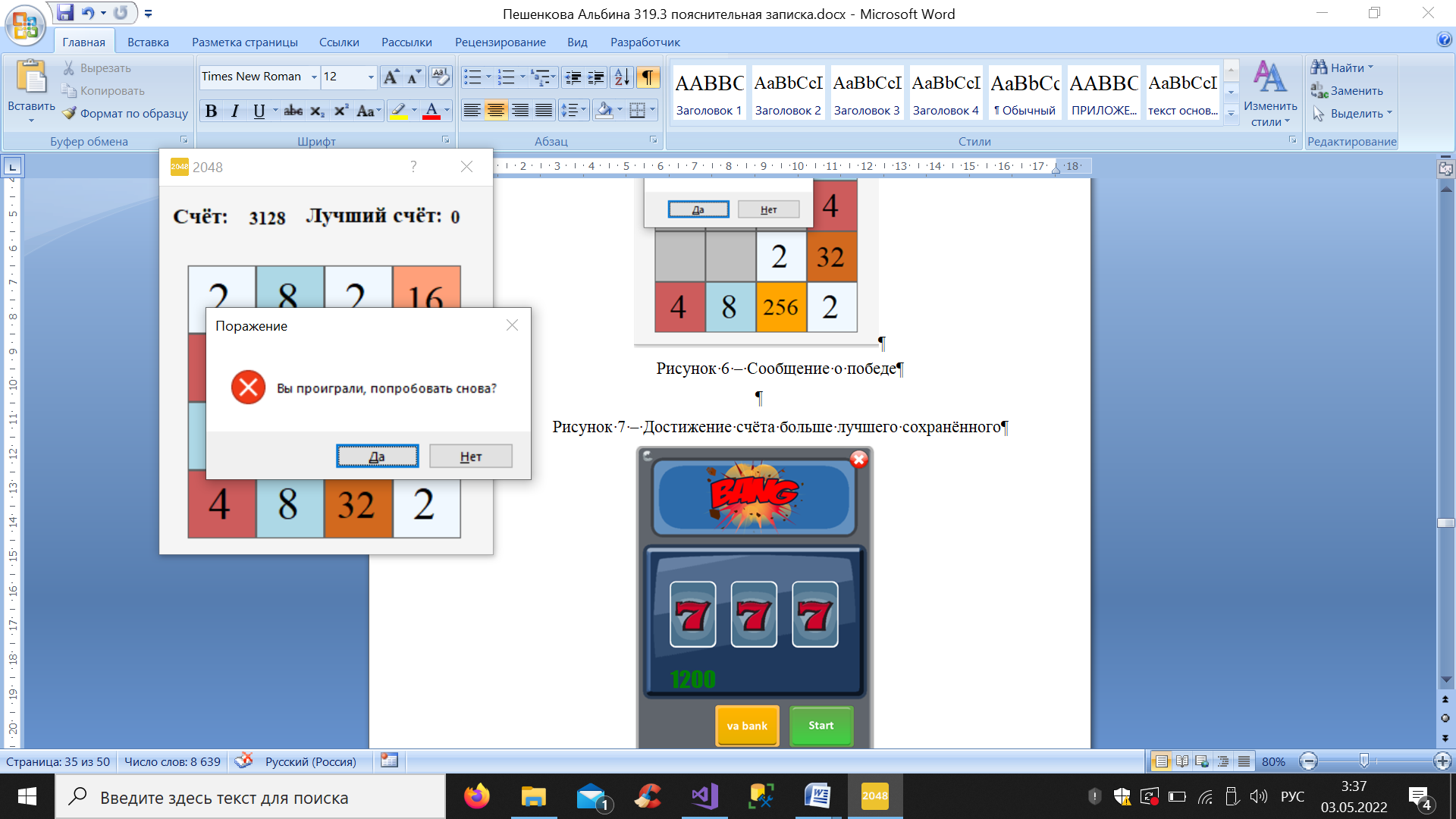


Рисунок 7 – Достижение счёта больше лучшего сохранённого



Рисунок 8 – Отображение нового лучшего результата

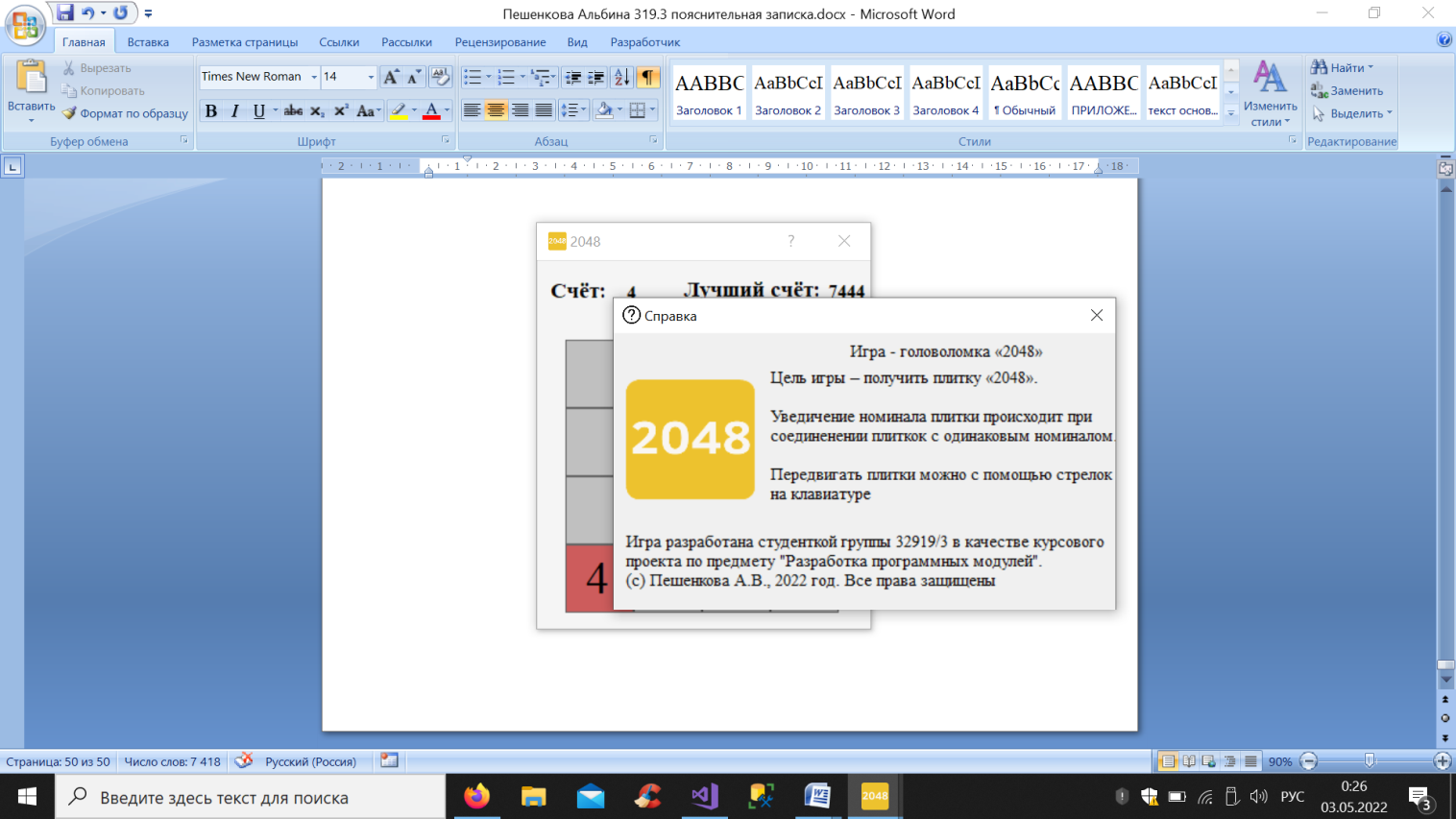


Рисунок 9 – Вызов справки

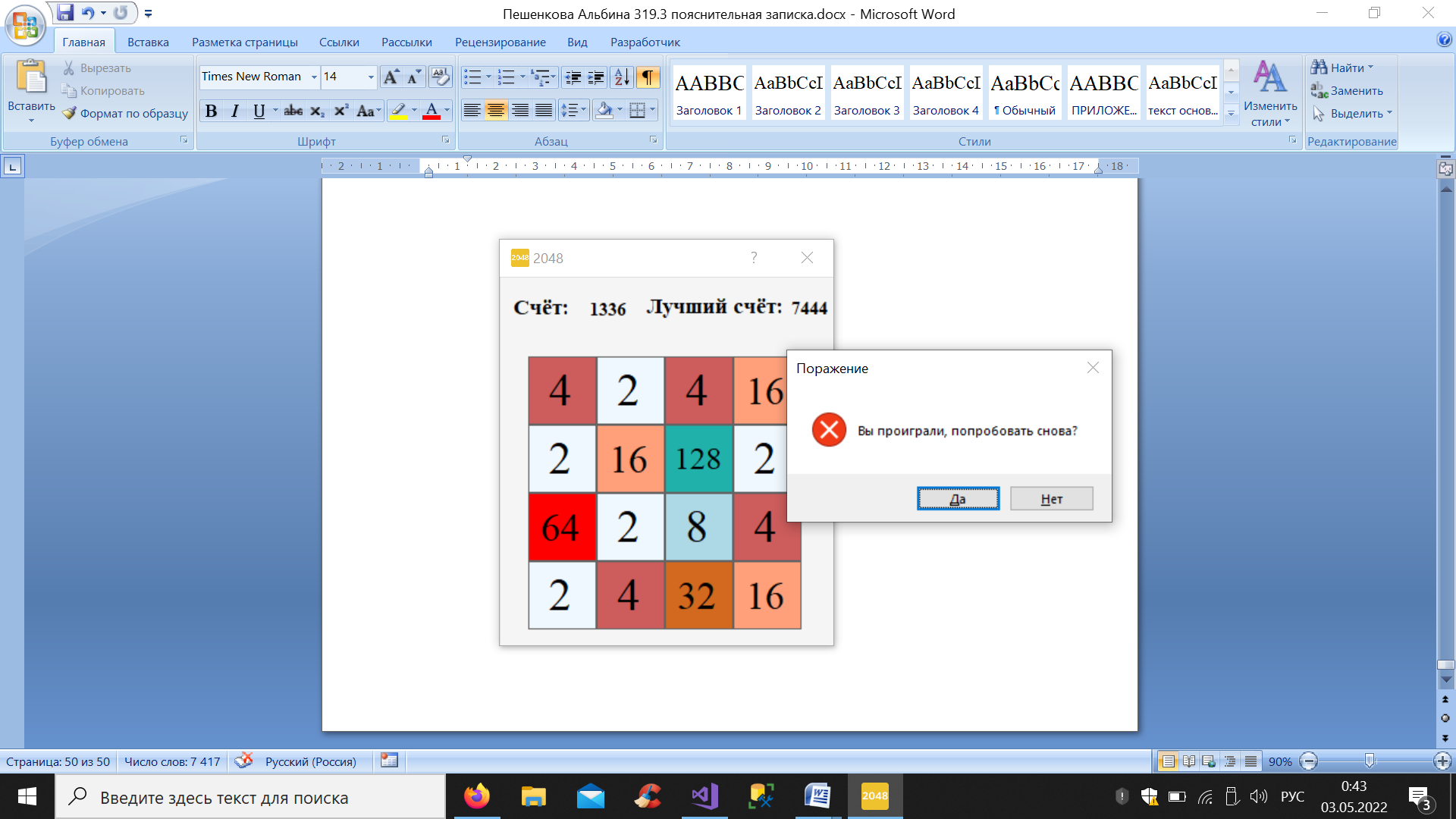


Рисунок 10 – Сообщение о поражении

ПРИЛОЖЕНИЕ В

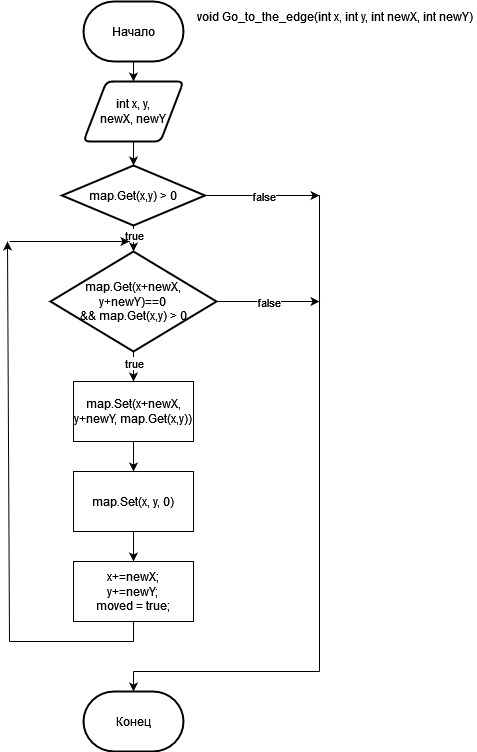


Рисунок 11 – Схема алгоритма метода void Go\_to\_the\_edge(int x, int y, int newX, int newY)

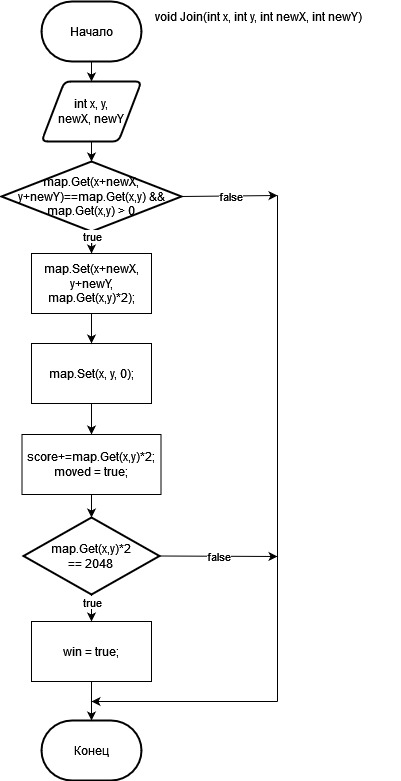


Рисунок 12 – Схема алгоритма метода void Join(string x, int y, int newX, int newY)

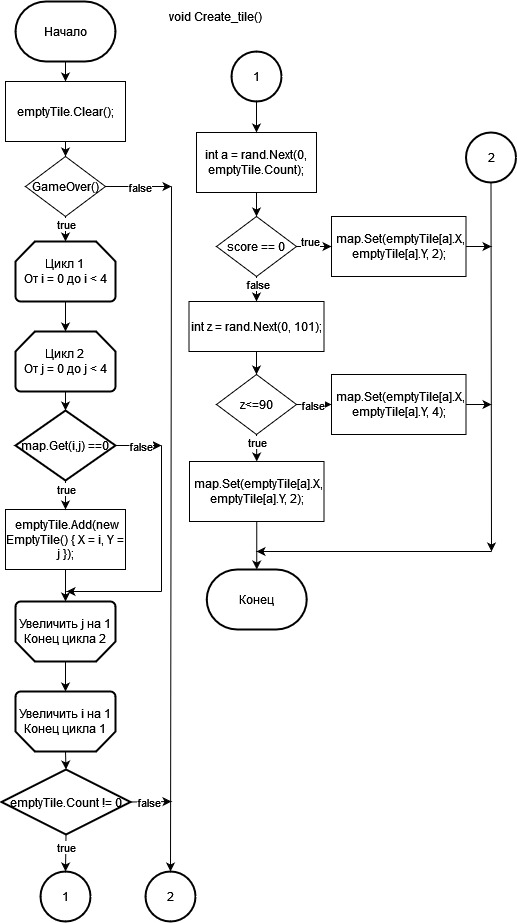


Рисунок 13 – Схема алгоритма метода void Creat\_tile()

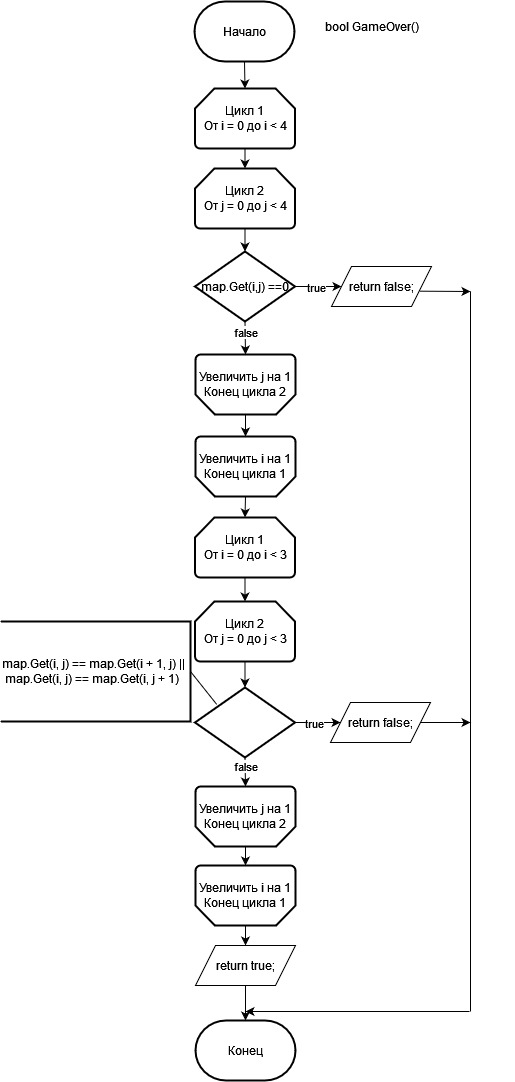


Рисунок 14 – Схема алгоритма метода bool GameOver()

ПРИЛОЖЕНИЕ Г

**Модуль Form1.cs**

using System;

using System.Drawing;

using System.Windows.Forms;

namespace \_2048

{

public partial class Form1 : Form

{

Model model; //экземпляр класса Model, в котором хранятся основные алгоритмы игры

Help\_window help; //экземпляр формы справки

bool win = false; //флаг, обозначающий было ли сообщение о победе для предотвращения повторного вывода сообщения при продолжении игры после победы

public bool helpWindow\_closed; //флаг, отражающий открыта ли справка

Label[,] tile = new Label[4, 4]; //плитки с числами

public Form1()

{

InitializeComponent();

Start();

this.KeyUp += new KeyEventHandler(pressing\_arrows); //добавление обработчика события нажатия клавиш

}

private void Form1\_Load(object sender, EventArgs e)

{

helpWindow\_closed = true;

score\_now.Text = "0";

bestScore.Text = Convert.ToString(Properties.Settings.Default.best\_score);

}

/// <summary>

/// Метод, который подготавливает форму к началу игры

/// </summary>

private void Start()

{

model = new Model();

model.Start();

//создание поля, на котором будут двигаться плитки

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

{

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

{

PictureBox pic = new PictureBox();

pic.Location = new Point(25 + 60 \* j, 70 + 60 \* i);

pic.BorderStyle = BorderStyle.FixedSingle;

pic.Size = new Size(60, 60);

pic.BackColor = Color.Silver;

this.Controls.Add(pic);

}

}

//отображение плиток

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

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

{

tile[i, j] = new Label();

if (model.GetMap(i, j) > 0)

tile[i, j].Text = Convert.ToString(model.GetMap(i, j));

else

tile[i, j].Text = "";

tile[i, j].MinimumSize = new Size(60, 60);

tile[i, j].BorderStyle = BorderStyle.FixedSingle;

tile[i, j].Size = new Size(60, 60);

tile[i, j].Location = new Point(25 + 60 \* j, 70 + 60 \* i);

tile[i, j].TextAlign = ContentAlignment.MiddleCenter;

tile[i, j].Font = new Font("Times New Roman", 30);

this.Controls.Add(tile[i, j]);

tile[i, j].BringToFront();

new\_color\_tile(i, j);

}

}

/// <summary>

/// Настройка цвета плиток в зависимости от числа на ней

/// </summary>

/// <param name="i">Номер строки, в которой находится плитка</param>

/// <param name="j">Номер столбца, в котором находится плитка</param>

private void new\_color\_tile(int i, int j)

{

switch (model.GetMap(i, j))

{

case 0:

tile[i, j].Visible = false;

tile[i, j].Font = new Font("Times New Roman", 30); break;

case 2:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.AliceBlue; tile[i, j].Font = new Font("Times New Roman", 30); break;

case 4:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.IndianRed; tile[i, j].Font = new Font("Times New Roman", 30); break;

case 8:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.LightBlue; tile[i, j].Font = new Font("Times New Roman", 30); break;

case 16:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.LightSalmon; tile[i, j].Font = new Font("Times New Roman", 25); break;

case 32:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Chocolate; tile[i, j].Font = new Font("Times New Roman", 25); break;

case 64:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Red; tile[i, j].Font = new Font("Times New Roman", 25); break;

case 128:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.LightSeaGreen; tile[i, j].Font = new Font("Times New Roman", 20); break;

case 256:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Orange; tile[i, j].Font = new Font("Times New Roman", 20); break;

case 512:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Aquamarine; tile[i, j].Font = new Font("Times New Roman", 20); break;

case 1024:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Coral; tile[i, j].Font = new Font("Times New Roman", 15); break;

case 2048:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Gold; tile[i, j].Font = new Font("Times New Roman", 15); break;

case 4096:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.Beige; tile[i, j].Font = new Font("Times New Roman", 15); break;

default:

tile[i, j].Visible = true;

tile[i, j].BackColor = Color.DarkViolet; tile[i, j].Font = new Font("Times New Roman", 15); break;

}

}

/// <summary>

/// Метод, который срабатывает при нажатии клавиш

/// </summary>

private void pressing\_arrows(object sender, KeyEventArgs e)

{

switch (e.KeyCode.ToString())

{

case "Right":

model.Right();

break;

case "Left":

model.Left();

break;

case "Up":

model.Up();

break;

case "Down":

model.Down();

break;

}

score\_now.Text = Convert.ToString(model.score); //вывод счёта

//изменение чисел на плитках

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

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

{

if (model.GetMap(i, j) != 0)

tile[i, j].Text = Convert.ToString(model.GetMap(i, j));

else

tile[i, j].Text = "";

new\_color\_tile(i,j);

}

//проверка на окончание игры

if (model.GameOver() || model.win)

{

if (model.GameOver())

{

if (MessageBox.Show("Вы проиграли, попробовать снова?", "Поражение", MessageBoxButtons.YesNo, MessageBoxIcon.Stop) == DialogResult.Yes)

Application.Restart();

else

this.Close();

}

else if(model.win && win == false)

{

if (MessageBox.Show("Поздравляем, вы победили!\nПродолжить игру?", "Победа", MessageBoxButtons.YesNo, MessageBoxIcon.None) == DialogResult.No)

this.Close();

win = true;

}

}

}

/// <summary>

/// Событие при закрытии формы

/// </summary>

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

//сохранение лучшего результата

if (Convert.ToInt32(score\_now.Text) > Properties.Settings.Default.best\_score)

{

Properties.Settings.Default.best\_score = Convert.ToInt32(score\_now.Text);

Properties.Settings.Default.Save();

}

}

/// <summary>

/// Вызов справки

/// </summary>

private void Form1\_HelpButtonClicked(object sender, System.ComponentModel.CancelEventArgs e)

{

if (helpWindow\_closed)

{

help = new Help\_window();

help.Owner = this;

helpWindow\_closed = false;

help.ShowDialog();

}

}

}

}

**Модуль Model.cs**

using System;

using System.Collections.Generic;

namespace \_2048

{

class Model

{

Map\_of\_tile map; //создание объекта класса Map\_of\_tile для работы с числами на плитках

Random rand = new Random(); //объект для получения рандомных значений

public bool win; //флаг, который отслеживает победу, т.е. достижение числа на плитке = 2048

bool moved; //флаг, отслеживающий произошло ли передвижение плиток на поле

List<EmptyTile> emptyTile = new List<EmptyTile>(); //список пустых плиток

public int score; //счёт игры

/// <summary>

/// Конструктор без параметров, выделяющий память под объект класса Map\_of\_tile

/// </summary>

public Model()

{ map = new Map\_of\_tile(); }

/// <summary>

/// Метод, устанавливающий значения для начала игры

/// </summary>

public void Start()

{

win = false;

score = 0;

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

{

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

map.Set(i, j, 0);

}

Create\_tile();

Create\_tile();

}

/// <summary>

/// Метод создания новых плиток, который устанавливает значения 2 (с вероятностью 90%) или 4 (с вероятностью 10%) на пустую плитку

/// </summary>

private void Create\_tile()

{

emptyTile.Clear();

if (GameOver()) return;

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

{

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

{

if (map.Get(i, j) <= 0)

{

emptyTile.Add(new EmptyTile() { X = i, Y = j });

}

}

}

if (emptyTile.Count != 0)

{

int a = rand.Next(0, emptyTile.Count);

if (score ==0)

map.Set(emptyTile[a].X, emptyTile[a].Y, 2);

else

{

int z = rand.Next(0, 101);

if (z <= 90)

map.Set(emptyTile[a].X, emptyTile[a].Y, 2);

else

map.Set(emptyTile[a].X, emptyTile[a].Y, 4);

}

}

}

/// <summary>

/// Метод, осуществляющий сдвиг плиток на поле в выбранную сторону

/// </summary>

/// <param name="x">Номер строки, в которой находится плитка</param>

/// <param name="y">Номер столбца, в котором находится плитка</param>

/// <param name="newX">Сдвиг плитки по вертикали</param>

/// <param name="newY">Сдвиг плитки по горизонтали</param>

void Go\_to\_the\_edge(int x, int y, int newX, int newY)

{

if (map.Get(x, y) > 0)

while (map.Get(x + newX, y + newY) == 0 && map.Get(x, y) > 0)

{

map.Set(x + newX, y + newY, map.Get(x, y));

map.Set(x, y, 0);

x += newX; y += newY;

moved = true;

}

}

/// <summary>

/// Метод, осуществляющий соединение плиток с одинаковыми числами

/// </summary>

/// <param name="x">Номер строки, в которой находится плитка</param>

/// <param name="y">Номер столбца, в котором находится плитка</param>

/// <param name="newX">Сдвиг плитки по вертикали</param>

/// <param name="newY">Сдвиг плитки по горизонтали</param>

void Join(int x, int y, int newX, int newY)

{

if (map.Get(x + newX, y + newY) == map.Get(x, y) && map.Get(x, y) > 0)

{

map.Set(x + newX, y + newY, map.Get(x, y) \* 2);

score += map.Get(x, y) \* 2;

if (map.Get(x, y) \* 2 == 2048) //проверка победы

win = true;

map.Set(x, y, 0);

moved = true;

}

}

/// <summary>

/// Метод движения плиток вправо

/// </summary>

public void Right()

{

moved = false;

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

for (int j = 2; j >= 0; j--)

Go\_to\_the\_edge(i, j, 0, 1);

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

for (int j = 2; j >= 0; j--)

Join(i, j, 0, 1);

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

for (int j = 2; j >= 0; j--)

Go\_to\_the\_edge(i, j, 0, 1);

if (moved) //создание новой плитки в случае, если плитки были сдвинуты

Create\_tile();

}

/// <summary>

/// Метод движения плиток влево

/// </summary>

public void Left()

{

moved = false;

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

for (int j = 1; j < 4; j++)

Go\_to\_the\_edge(i, j, 0, -1);

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

for (int j = 1; j < 4; j++)

Join(i, j, 0, -1);

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

for (int j = 1; j < 4; j++)

Go\_to\_the\_edge(i, j, 0, -1);

if (moved) //создание новой плитки в случае, если плитки были сдвинуты

Create\_tile();

}

/// <summary>

/// Метод движения плиток вверх

/// </summary>

public void Up()

{

moved = false;

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

for (int i = 1; i < 4; i++)

Go\_to\_the\_edge(i, j, -1, 0);

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

for (int i = 1; i < 4; i++)

Join(i, j, -1, 0);

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

for (int i = 1; i < 4; i++)

Go\_to\_the\_edge(i, j, -1, 0);

if (moved) //создание новой плитки в случае, если плитки были сдвинуты

Create\_tile();

}

/// <summary>

/// Метод движения плиток вниз

/// </summary>

public void Down()

{

moved = false;

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

for (int i = 2; i >= 0; i--)

Go\_to\_the\_edge(i, j, 1, 0);

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

for (int i = 2; i >= 0; i--)

Join(i, j, 1, 0);

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

for (int i = 2; i >= 0; i--)

Go\_to\_the\_edge(i, j, 1, 0);

if (moved) //создание новой плитки в случае, если плитки были сдвинуты

Create\_tile();

}

/// <summary>

/// Функция проверки окончания игры из-за проигрыша

/// </summary>

/// <returns>Окончание игры</returns>

public bool GameOver()

{

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

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

if (map.Get(i, j) == 0)

return false;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

if (map.Get(i, j) == map.Get(i + 1, j) || map.Get(i, j) == map.Get(i, j + 1))

return false;

return true;

}

/// <summary>

/// Функция для получения числа с плитки

/// </summary>

/// <param name="i">Номер строки, в которой находится плитка</param>

/// <param name="j">Номер столбца, в котором находится плитка</param>

/// <returns>Число на плитке</returns>

public int GetMap(int i, int j)

{

return map.Get(i, j);

}

}

/// <summary>

/// Класс для записи координат пустых плиток

/// </summary>

public class EmptyTile

{

public int X { get; set; }

public int Y { get; set; }

}

}

**Модуль Map\_of\_tile.cs**

using System;

namespace \_2048

{

class Map\_of\_tile

{

int[,] map\_tile; //двумерный массив для хранения чисел на плитах

/// <summary>

/// Конструктор без параметров, в котором выделяется память под масив, хранящий значения на плитках

/// </summary>

public Map\_of\_tile()

{

map\_tile = new int[4, 4];

}

/// <summary>

/// Функция получения числа на плитке

/// </summary>

/// <param name="x">Номер строки, в которой находится плитка</param>

/// <param name="y">Номер столбца, в котором находится плитка</param>

/// <returns></returns>

public int Get(int x, int y)

{

if (x >= 0 && x < 4 && y >= 0 && y < 4)

return map\_tile[x, y];

return -1; //возврат -1, если запрошено значение с плитки вне поля

}

/// <summary>

/// Метод, устанавливающий число на плитке

/// </summary>

/// <param name="x">Номер строки, в которой находится плитка</param>

/// <param name="y">Номер столбца, в котором находится плитка</param>

/// <param name="num">Устанавливаемое число</param>

public void Set(int x, int y, int num)

{

if (x >= 0 && x < 4 && y >= 0 && y < 4)

map\_tile[x, y] = num;

}

}

}

**Модуль Help\_window.cs**

using System;

using System.Windows.Forms;

namespace \_2048

{

public partial class Help\_window : Form

{

Form1 form;

public Help\_window()

{

InitializeComponent();

}

private void Help\_window\_Load(object sender, EventArgs e)

{

form = (Form1)this.Owner; //задание владельца формы

}

private void Help\_window\_FormClosed(object sender, FormClosedEventArgs e)

{

form.helpWindow\_closed = true; //передача в основную форму информации, что справка закрыта

}

}

}