МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гродненский государственный университет имени Янки Купалы»

Факультет математики и информатики

Кафедра системного программирования и компьютерной безопасности

ДМИТРИЕВА АНГЕЛИНА ЮРЬЕВНА

**Разработка игрового приложения на языке C#**

Курсовая работа

студента 2 курса специальности

1-260301 "Управление информационными ресурсами"

дневной формы получения образования

|  |  |
| --- | --- |
|  | Научный руководитель  Урбан Ольга Ивановна,  старший преподаватель кафедры современных технологий программирования, магистр физико-математических наук |

Гродно 2019

**Оглавление**

[Введение 4](#_Toc9812843)

[1. Постановка задачи 6](#_Toc9812844)

[1.1 Актуальность решаемой задачи 6](#_Toc9812845)

[1.2 Средства разработки 6](#_Toc9812846)

[2. Разработка проекта 8](#_Toc9812847)

[2.1 Словарь, база слов 8](#_Toc9812848)

[2.2 Размещения без повторений 8](#_Toc9812849)

[2.3 Поля приложения 9](#_Toc9812850)

[2.4 Методы приложения 11](#_Toc9812851)

[3. Руководство пользователя 17](#_Toc9812852)

[Заключение 21](#_Toc9812853)

**РЕЗЮМЕ**

Тема курсовой работы

**«Разработка игрового приложения на языке C#»**

Работа содержит: 22 страницы, 11 рисунков, 1 таблицу, 5 использованных источника литературы.

Ключевые слова: C#, Microsoft .NET Framework, Window Forms, игровое приложение, Linq.

Цель курсовой работы – разработка игрового приложения, в котором пользователь составляет слова из заданных букв, а программа проверяет их на правильность и выдает в конце игры подходящие слова

Объектом исследования выступает разработка приложения на языке C#.

Предметом исследования настоящей работы является программирование приложения с использованием технологии Window Forms.

В работе были использованы следующие методы: сравнительный анализ, моделирование, абстрагирование.

# **Введение**

Индустрия игровых приложений разнообразна. И среди всех жанров и видов выделяются так называемые «офисные игры». Это небольшие, не занимающие много место приложения, в которые люди обычно играют в перерыве на работе. Такие игры по своему определению должны быть просты в использовании, а прохождение одного уровня не должно занимать больше пяти минут.

Приложение «Аммаргана» — небольшая словесная игра, написанная на языке программирования C#. Само ее название отсылает к принципу работы данного приложения. Если в названии «Аммаргана» переставить буквы должным образом, получится слово анаграмма. Анаграмма — литературный приём, состоящий в перестановке букв или звуков определённого слова, что в результате даёт другое слово.

Данное приложение выбирает случайное слово из подготовленного словаря. Размер слова определяется уровнем сложности, которую пользователь выбирает сам. Целью игры является найти максимум русских слов, состоящих из букв базисного слова, а также составить слово, используя все буквы. Время игры ограничено, у пользователя есть 240 секунд на каждом уровне. Для увеличения интереса игрока на экран также выводится количество оставшихся слов.

Затем, по истечении заданного времени, либо по желанию самого игрока приложение выводит список всех возможных слов на форму. В этом списке пользователь сможет увидеть слова, которые он смог найти — они отмечены зеленым цветом. Остальные же отмечены красным. Кроме того, подсчитывается количество найденных слов из всех возможных.

По своей задумке игра «Аммаргана», пусть у нее нет броского интерфейса и особого разнообразия в уровнях, все же должна затягивать игрока. Это интеллектуальная игра проверит вашу память и грамматические навыки.

Для создания графического интерфейса приложения с помощью платформы .NET была использована технология Window Forms или просто формы. Она до сих пор остается наиболее простой и удобной в применении платформой.

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Внутри .NET Framework, Windows Forms реализуется в рамках пространства имён System.Windows.Forms.

Для работы с формами использовался объектно-ориентированный язык программирования — С#. Этот язык был разработан специально для создания приложений на платформе Microsoft .NET Framework. [1]

# **1. Постановка задачи**

## **1.1 Актуальность решаемой задачи**

Приложение «Аммаграна» типичная офисная игра для того, чтобы «убить время». Однако, это и является основой ее актуальности. Подобные мини-игры всегда пользовались популярностью, особенно в период недоступности мощных компьютеров. И даже сейчас, когда моно не беспокоиться о количестве потребляемой операционной памятью, такое приложение сможет найти своего пользователя.

Актуальность решения данной задачи состоит также в способе ее решения. Для разработки приложения использовался объектно-ориентированный язык программирования — С#. Это универсальный язык, подходящий для различных целей. В данном случае он был использован для написания приложения для ПК с графическим пользовательским интерфейсом.

Методы, использованные в приложении «Аммаграна», подходят и работают с любым фонетическим языком. То есть, языком, в котором есть нормальный человеческий алфавит — для китайского или японского не подойдет, конечно, но вот с английским справляется отлично. Для этого достаточно в файлы вместо списка русских слов вставить список слов нужного языка.

## **1.2 Средства разработки**

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio 2017;
* программная платформа .NET Framework 5.0;
* язык программирования C#;
* технология Windows Forms;
* технология Linq.Enumerable
* классы C# (коллекции, IO и др.);

C# — простой, современный объектно-ориентированный и типобезопасный язык программирования. C# относится к широко известному семейству языков C, и покажется хорошо знакомым любому, кто работал с C, C++, Java или JavaScript. Здесь представлен обзор основных компонентов языка. Если вы хотите изучить язык с помощью интерактивных примеров, рекомендуем поработать с нашими вводными руководствами по C#.

C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов — это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такую концепцию работы. Благодаря этому C# отлично подходит для создания и применения программных компонентов.

Вот лишь несколько функций языка C#, обеспечивающих надежность и устойчивость приложений: сборка мусора автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами; обработка исключений предоставляет структурированный и расширяемый способ выявлять и обрабатывать ошибки; строгая типизация языка не позволяет обращаться к неинициализированным переменным, выходить за пределы индексируемых массивов или выполнять неконтролируемое приведение типов.

В C# существует единая система типов. Все типы C#, включая типы-примитивы, такие как int и double, наследуют от одного корневого типа object. Таким образом, все типы используют общий набор операций, и значения любого типа можно хранить, передавать и обрабатывать схожим образом. Кроме того, C# поддерживает пользовательские ссылочные типы и типы значений, позволяя как динамически выделять память для объектов, так и хранить упрощенные структуры в стеке.

Чтобы обеспечить совместимость программ и библиотек C# при дальнейшем развитии, при разработке C# много внимания было уделено управлению версиями. Многие языки программирования обходят вниманием этот вопрос, и в результате программы на этих языках ломаются чаще, чем хотелось бы, при выходе новых версий зависимых библиотек. Вопросы управления версиями существенно повлияли на такие аспекты разработки C#, как раздельные модификаторы virtual и override, правила разрешения перегрузки методов и поддержка явного объявления членов интерфейса. [2]

# **2. Разработка проекта**

## **2.1 Словарь, база слов**

Первостепенной задачей для данного приложение является подготовка базы слов, с помощью которых и будет работать приложение. Эти слова следует где-то хранить и затем использовать. К счастью, в Сети есть уже собранные и готовые толковые словари в виде текстовых файлах.

Для игрового приложения использовался список русских слов с количеством букв не больше девяти на основе следующего словаря:

Ефремова Т. Ф. Толковый словарь. — М.: Рус. яз., 1996.

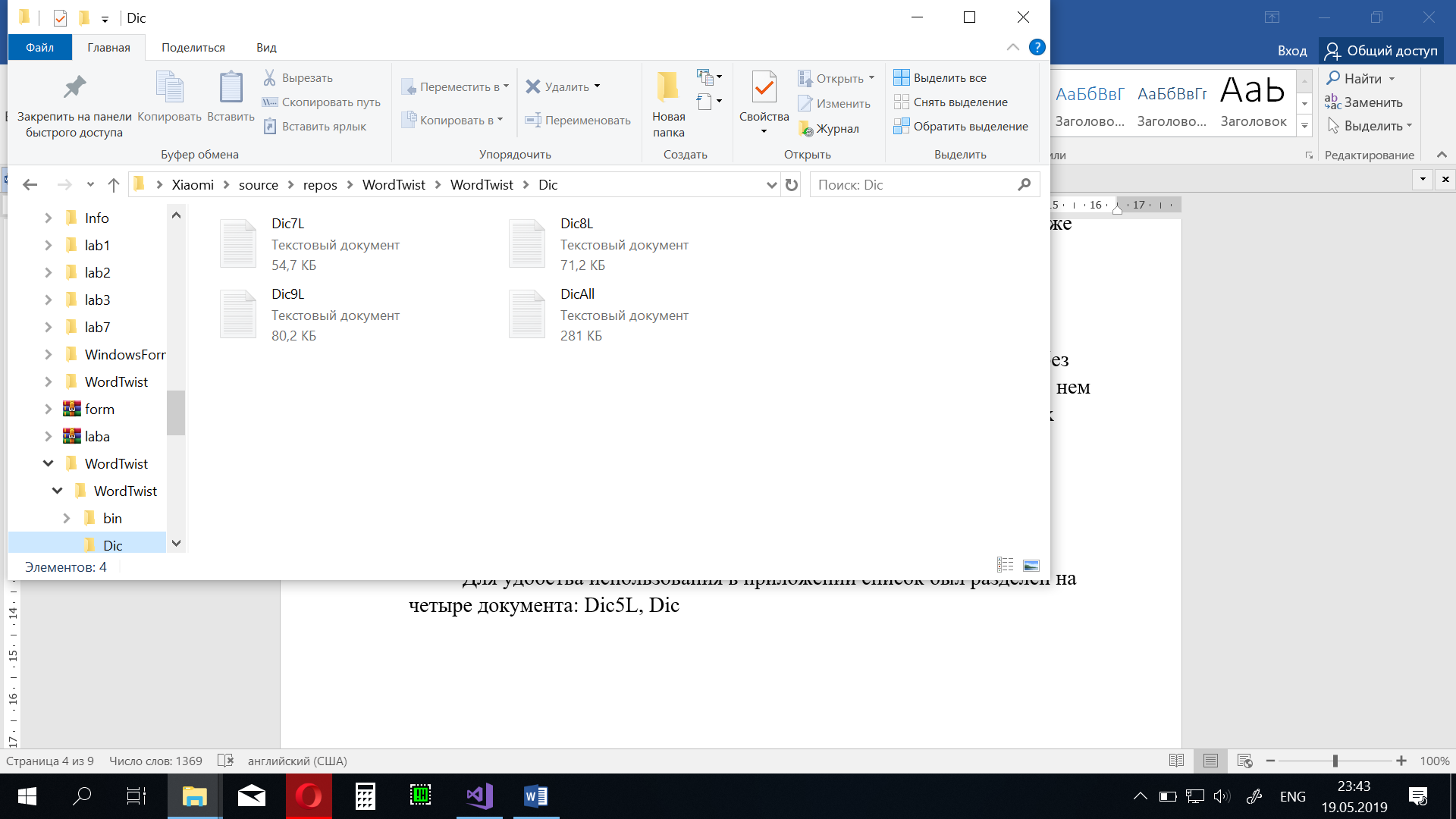
В списке собраны только существительные в начальной форме без имен собственных. Однако список можно считать неполным, так как в нем нет многих заимствованных слов, компьютерной тематики и из других профессиональных областей. [3]

Рисунок 2.1 — Словари приложения

Для удобства использования в приложении список был разделен на четыре документа: Dic7L.txt, Dic8L.txt, Dic9L.txt, DicAll.txt.

В первых трех содержаться слова из 7-ми, 8-ми и 9-ти букв соответственно. Эти словари используются для выбора базового слова, из букв которого потом надо будет составлять другие слова. Такая длина базовых слов была выбрана, так как по статистике средняя длина слова в русском языке 7,2 символа на слово. Нет смысла делать начальное слово короче 7-ми, но и слишком длинное слово не подойдет — тогда будет слишком много возможных вариантов, и пользователь устанет их искать.

Последний файл содержит слова из предыдущих файлов и еще дополнительно слова с меньшим количеством букв (от 3-ех).

## **2.2 Размещения без повторений**

Теперь, когда у нас есть базовый словарь следует подумать об алгоритме поиска слов. Он в свою очередь сводится к тому, что выбирается базовое слово и получаются все возможные варианты сочетания его букв, которые и проверяются на наличие в базе слов.

Кроме того, из слова одной длины можно составить слова, которые будут короче. Поэтому придется обратиться к комбинаторике. Так как каждую букву из начального слова можно использовать только один раз, а также учитывается порядок, то нам подойдет формула для размещения без повторений. Будем искать слова длиной от трех до («длина начального слова» — 1) букв.

Рассчитаем, сколько вариантов нам надо будет перебрать при различной длине исходного слова. Применим следующую формулу:

Число размещений без повторений из десяти элементов по K:

|  |  |
| --- | --- |
| К | Количество размещений |
| 1 | 10 |
| 2 | 90 |
| 3 | 720 |
| 4 | 5040 |
| 5 | 30240 |
| 6 | 151200 |
| 7 | 604800 |
| 8 | 1814400 |
| 9 | 3628800 |
| Всего: | 6235300 |

Таблица 2.2 — Число размещений без повторений из десяти элементов по K

## **2.3 Поля приложения**

Теперь перейдем к исходному коду приложения.

Для работы программы нужны некоторые переменные для хранения данных. Поля с этими переменными можно увидеть на рисунке 2.3.

Рассмотрим эти переменные поближе:

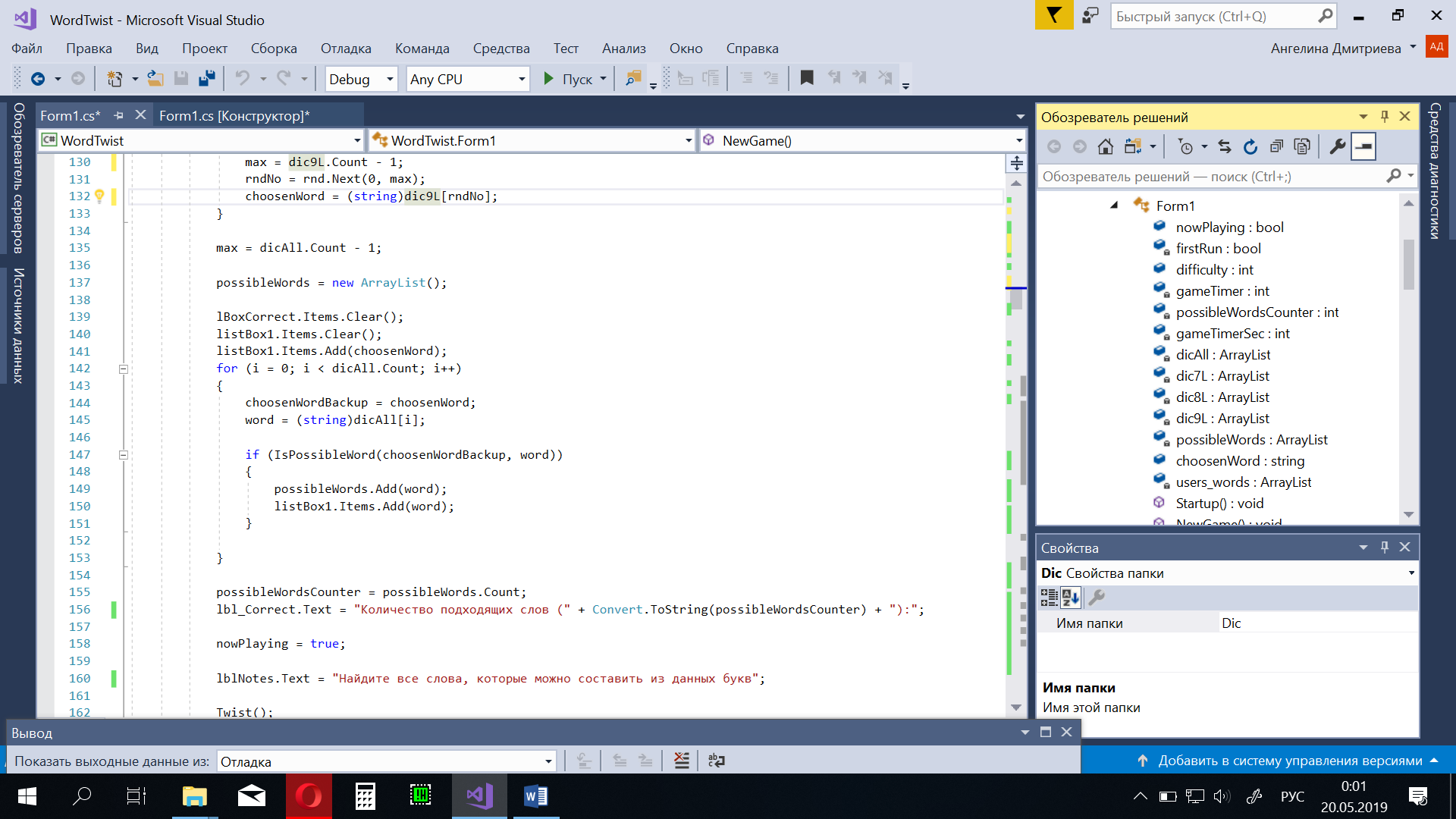
* public bool nowPlaying — логическая переменная, хранящая данные о том, запущена ли в данный момент игра или нет. Нужна для выдачи уведомлений о том, что игра запущена, когда пользователь хочет начать еще одну или поменять уровень. Принимает отрицательное значение при остановке таймера и нажатии на кнопку «Закончить»;
* bool firstRun — логическая переменная, показывающая, первый ли это запуск новой игры или нет. Нужна для запуска метода, загружающего словари в коллекции;
* public int difficulty — выбранная пользователем сложность. Может быть 1, 2 или 3. От сложности зависит длина базового слова и количество показанных на форме букв;
* int gameTimer — таймер приложения. Запускается в начале игры. Останавливается при нажатии кнопки «Закончить»;
* ArrayList possibleWords;
* int possibleWordsCounter — счетчик количества возможных слов, составленных из базового слова. Нужен для вывода информации об количестве оставшихся слов;
* int gameTimerSec — таймер приложения в секундах. Используется для вывода оставшегося времени на форму;
* ArrayList dicAll — расширяющийся массив, содержащий слова из файла DicAll.txt. Используется при поиске подходящих под условие слов, составленных из базового слова;
* ArrayList dic7L — расширяющийся массив содержащий слова из файла Dic7L.txt. Используется для выбора случайного слова из 7-ми букв;
* ArrayList dic8L — расширяющийся массив содержащий слова из файла Dic8L.txt. Используется для выбора случайного слова из 8-ми букв;
* ArrayList dic9L — расширяющийся массив содержащий слова из файла Dic9L.txt. Используется для выбора случайного слова из 9-ми букв;
* public string choosenWord — строковая переменная, содержащая случайное выбранное слово, длиной соответствующее уровню сложности.

Рисунок 2.3 — Поля приложения

Данные переменные хранятся в классе формы и используются на протяжении работы приложения. Для работы с этими переменными написаны специальные методы, находящиеся в различных классах программы. [4]

## **2.4 Методы приложения**

**Метод загрузки словарей в массивы Startup().**

Данный метод запускается один раз в начале приложения (if firstRun = true) из метода NewGame(). Он отсылает к статическому методу из класса LoadFile. Этот метод реализует класс StreamReader, который позволяет считывать весь текст или отдельные строки из текстового файла. Для построковой записи в массив используется метод этого класса ReadLine(). Возвращаемым значением является массив, а условием — строковое значение пути к нужному файлу.

class LoadFile

{

public static ArrayList Load(string file)

{

ArrayList array = new ArrayList();

string line;

System.IO.StreamReader MyReader;

MyReader = new StreamReader(file);

line = MyReader.ReadLine();

while (line != null)

{

array.Add(line.ToString());

line = MyReader.ReadLine();

}

MyReader.Close();

return array;

}

}

Данный метод применяется для всех четырех файлов. Данные из этих файлов записываются в отдельные маасивы.

dic7L = LoadFile.Load(@"C:\Users\Xiaomi\source\repos\WordTwist\WordTwist\Dic\Dic7L.txt");

dic8L = LoadFile.Load(@"C:\Users\Xiaomi\source\repos\WordTwist\WordTwist\Dic\Dic8L.txt");

dic9L = LoadFile.Load(@"C:\Users\Xiaomi\source\repos\WordTwist\WordTwist\Dic\Dic9L.txt");

dicAll = LoadFile.Load(@"C:\Users\Xiaomi\source\repos\WordTwist\WordTwist\Dic\DicAll.txt");

При этом логическая переменная nowPlaying принимает значение false, а таймер отключается.

**Метод начала новой игры NewGame().**

Данный метод запускается каждый раз, когда пользователь нажимает кнопку «Новая игра» или меняет уровень сложности. Если метод вызывается впервые, то выполняется метод считывания словарей из метода Startup().

if (firstRun == true)

{

Startup();

firstRun = false;

}

Вызывается метод Clear(), который очищает поле для ввода слов и включает кнопки с буквами.

Затем начинается все самое интересное — выбирается случайное базовое слово, соответствующее сложности.

Random rnd = new Random();

if (difficulty == 1)

{

max = dic7L.Count - 1;

rndNo = rnd.Next(0, max);

choosenWord = (string)dic5L[rndNo];

}

else if (difficulty == 2)

{

max = dic8L.Count - 1;

rndNo = rnd.Next(0, max);

choosenWord = (string)dic6L[rndNo];

}

else

{

max = dic9L.Count - 1;

rndNo = rnd.Next(0, max);

choosenWord = (string)dic7L[rndNo];

}

Это действие выполняется с помощью класса Random(). Целочисленной переменно max присваивается значение длины массива с базой слов (файлы со словами, состоящими из 7-ми, 8-ми или 9-ти букв). Затем выбирается случайное целое число от нуля до (длины массива – 1). Это число и становится индексом выбранного слова.

Далее надо найти все слова, составленные из букв данного слова. Для этого с помощью цикла проходимся по словарю со всеми словами dicAll и каждый его элемент сравниваем с выбранным словом. Для этого был написан специальный метод IsPossibleWord(string baseWord, string compareWord) (рисунок 2.4). В качестве переменных он принимает два строковых значение — одно для базового слова и другое для слова, которое нужно проверить.

for (i = 0; i < dicAll.Count - 1; i++)

{

choosenWordBackup = choosenWord;

word = (string)dicAll[i];

if (IsPossibleWord(choosenWordBackup, word))

{

possibleWords.Add(word);

}

}

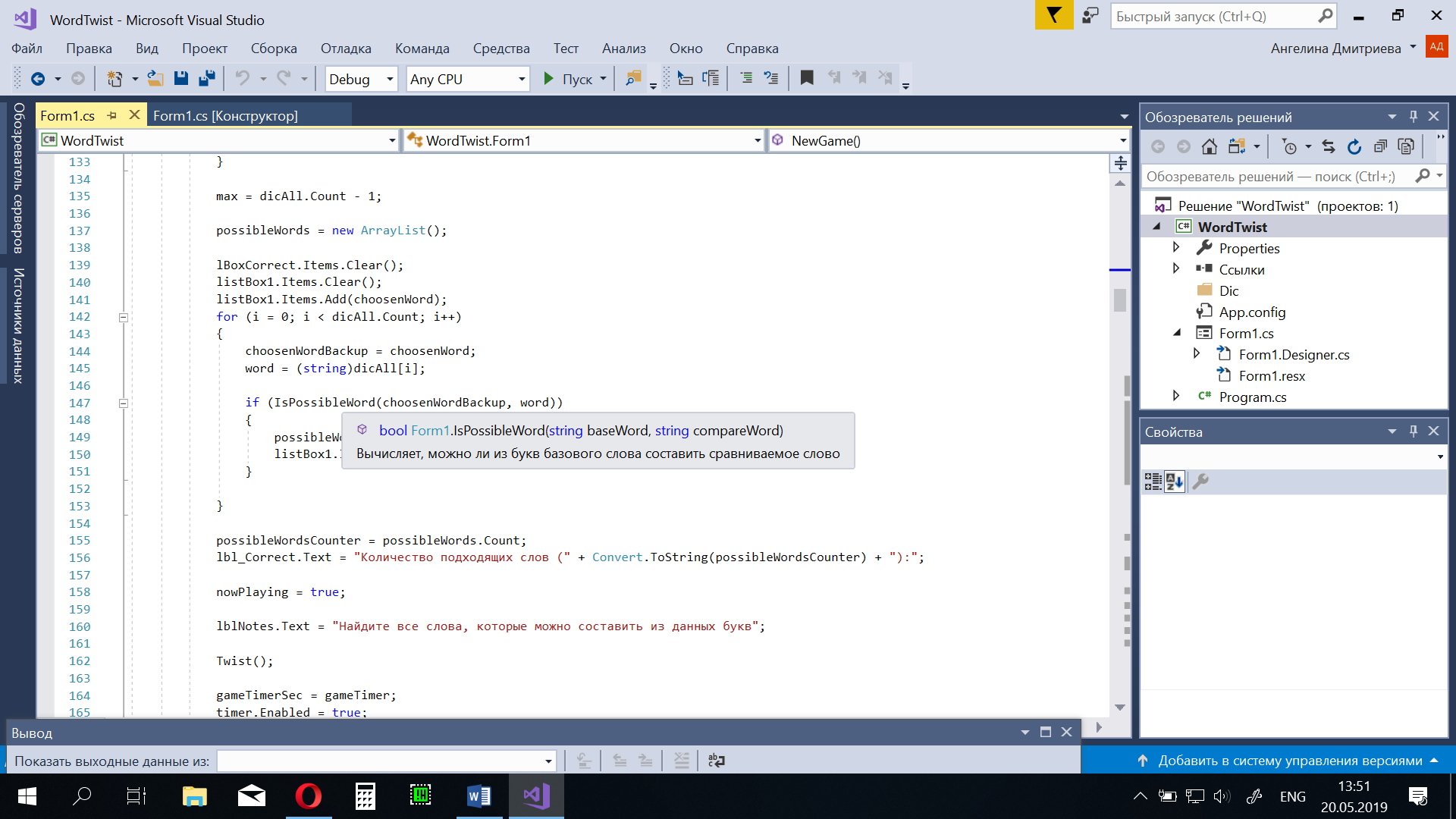


Рисунок 2.4.1 — метод IsPossibleWord()

**Метод поиска возможных слов IsPossibleWord()**

Данный метод является самым интересным в этом приложении, хотя он и достаточно прост в исполнении. Чтобы узнать, можно ли из некоторых символов составить существующее слово можно поступить следующим образом.

public bool IsPossibleWord(string baseWord, string compareWord)

{

bool found = true;

while (found && compareWord.Length > 0)

{

char currChar = compareWord[0];

compareWord = compareWord.Remove(0, 1);

int index = baseWord.IndexOf(currChar);

if (index >= 0)

{

baseWord = baseWord.Remove(index, 1);

}

else

{

found = false;

break;

}

}

return found;

}

В цикле мы с помощью метода IndexOf ищем индекс символа currChar проверяемого слова compareWord в базовом слове baseWord. В случае, если индекс равен -1 (то есть его не существует), цикл завершается и выдает логическое значение false. Если же такой символ присутствует, то логическая переменная остается равной true, этот символ удаляется из обоих слов (метод Remove) и цикл переходит к следующему символу compareWord.

Таким образом прорабатываются все символы compareWord и если длина этого слова становится равна нулю, то оно подходит под наше условие. Тогда метод возвращает значение true в метод NewGame() и данное слово записывается в массив possibleWords.

Далее в методе NewGame() подсчитывается количество таких возможных слов (possibleWordsCounter = possibleWords.Count) и выводится на экран формы. Переменная nowPlaying принимает значение true.

Также в методе новой игры запускается метод перемешивания букв в выбранном слове.

**Метод перестановки букв в базовом слове Twist()**

Данный метод вызывается при начале новой игры из метода NewGame(), а также при нажатии кнопки «Перемешать».

Принцип его работы заключается в следующем. Создается целочисленный массив pickedSeq длиной равный количеству букв в выбранном слове (один из трех уровней + 4). Он заполняется последовательно идущими числами от 0 до (длина массива – 1).

Затем с помощью цикла эти элементы перебираются, при этом каждый элемент меняется с другим случайным элементом с меньшим индексом.

letterCount = difficulty + 4;

pickedSeq = new int[letterCount];

for (j = 0; j <= letterCount - 1; j++)

pickedSeq[j] = j;

Random rnd = new Random();

for (i = letterCount - 1; i >= 1; i--)

{

j = rnd.Next(i + 1);

int temp = pickedSeq[j];

pickedSeq[j] = pickedSeq[i];

pickedSeq[i] = temp;

}

После этого каждой букве присваиваются новые индексы с помощью метода Substring(int начальный индекс, int длина).

button1.Text = choosenWord.Substring(pickedSeq[0], 1);

button2.Text = choosenWord.Substring(pickedSeq[1], 1);

button3.Text = choosenWord.Substring(pickedSeq[2], 1);

button4.Text = choosenWord.Substring(pickedSeq[3], 1);

button5.Text = choosenWord.Substring(pickedSeq[4], 1);

if (difficulty == 2)

button6.Text = choosenWord.Substring(pickedSeq[5], 1);

else if (difficulty == 3)

{

button6.Text = choosenWord.Substring(pickedSeq[5], 1);

button7.Text = choosenWord.Substring(pickedSeq[6], 1);

}

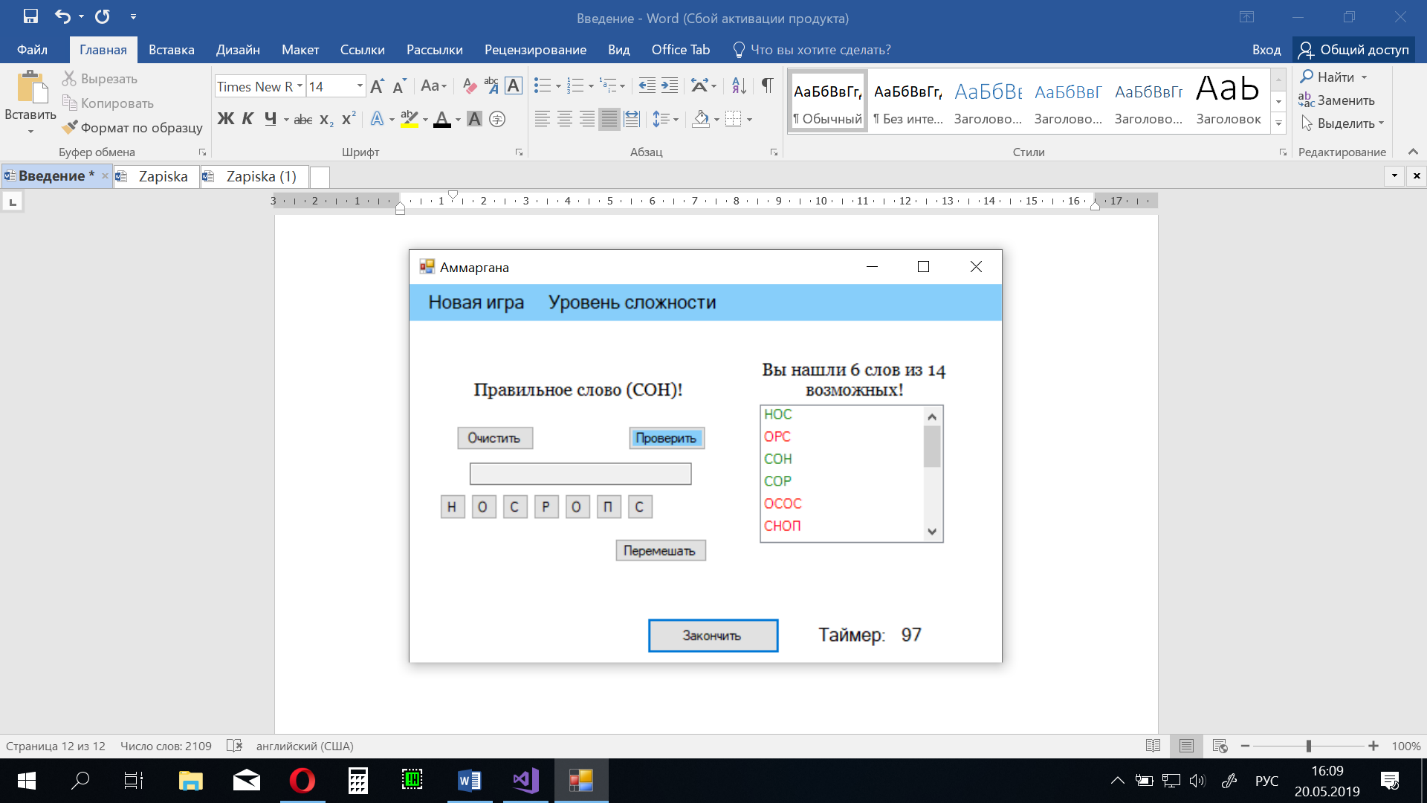
**Метод lBoxCorrect\_DrawItem(object sender, DrawItemEventArgs e)**

Рисунок 2.4.2 — Работа метода lBoxCorrect\_DrawItem

В конце игры после нажатия кнопки «Закончить» либо после окончания работы таймера пользователь может увидеть статистику — то, сколько он слов смог найти. При этом в listBox выводятся все возможные слова: те, которые пользователь нашел выводятся зеленым цветом, остальные — красным.

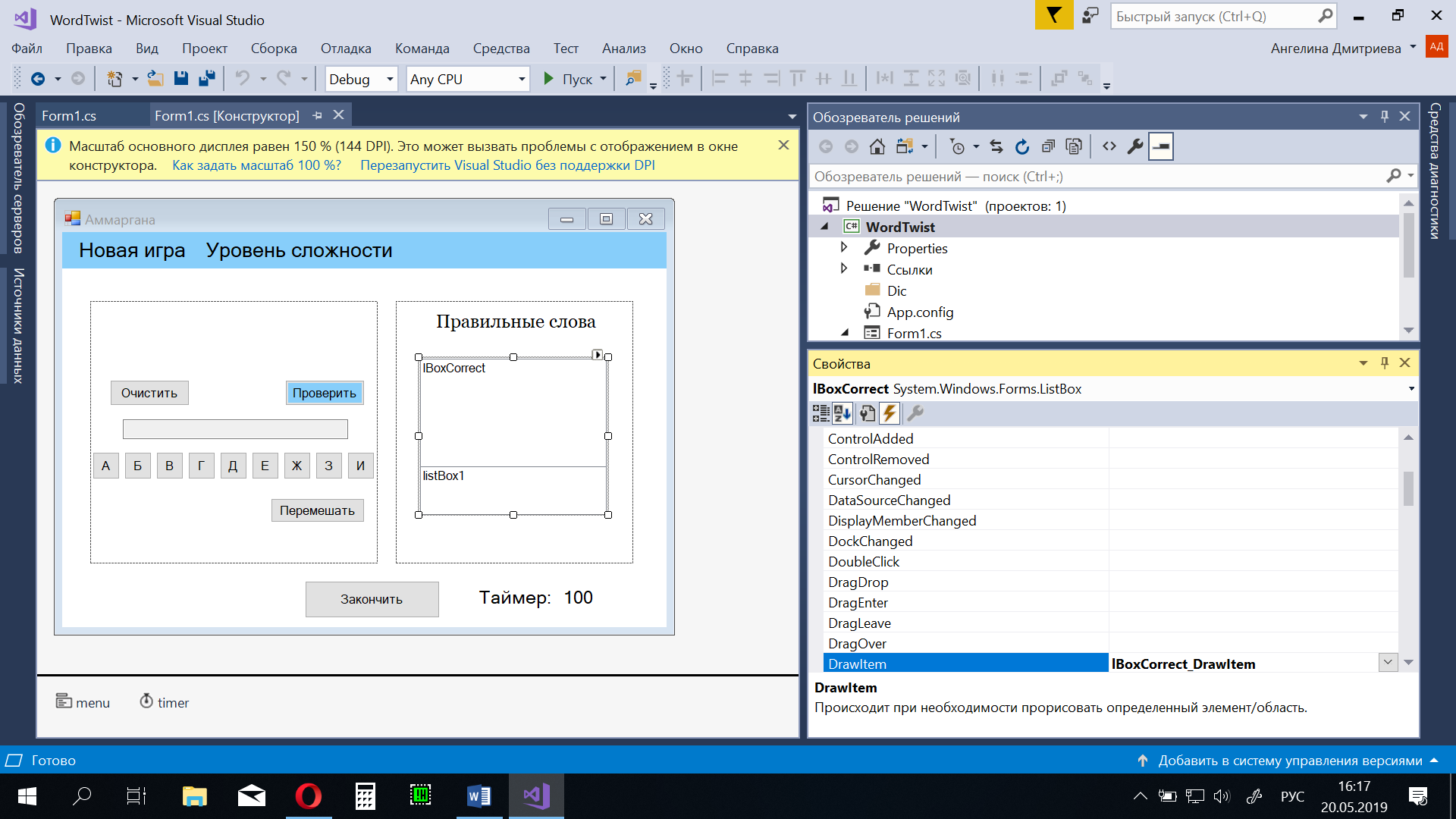
Для того, чтобы сделать подобную функцию возможной пришлось обратиться к обработчику событий для нашего listBox.

Рисунок 2.4.3 — событие DrawItem

Для работы этого метода в вызывающем его методе следует заполнить список нужными значениями и прописать следующий код. Здесь задается способ рисования элементов управления.

lBoxCorrect.DrawMode = DrawMode.OwnerDrawFixed;

lBoxCorrect.DrawItem += new DrawItemEventHandler(lBoxCorrect\_DrawItem);

panel2.Controls.Add(lBoxCorrect);

Далее в отдельный метод под названием lBoxCorrect\_DrawItem записываем способ рисования элементов в списке. Сначала проверяется, не пустой ли массив найденных пользователем слов users\_words. Если так, то все элементы закрашиваются в красный (Brushes.Red).

if (users\_words == null)

{

myBrush = Brushes.Red;

e.Graphics.DrawString(lBoxCorrect.Items[e.Index].ToString(),

e.Font, myBrush, e.Bounds, StringFormat.GenericDefault);

}

Если массив не пустой, то ищется индекс значения элемента DrawItemEventArgs e в этом массиве со словами пользователя. Если индекс больше -1, значит элемент существует и в списке на форме он окрашивается в зеленый. Иначе — в красный.

int pos = Array.IndexOf(users\_words1, lBoxCorrect.Items[e.Index]);

if (pos > -1)

{

myBrush = Brushes.Green;

}

else

{

myBrush = Brushes.Red;

}

e.Graphics.DrawString(lBoxCorrect.Items[e.Index].ToString(), e.Font, myBrush, e.Bounds, StringFormat.GenericDefault);

# **3. Руководство пользователя**

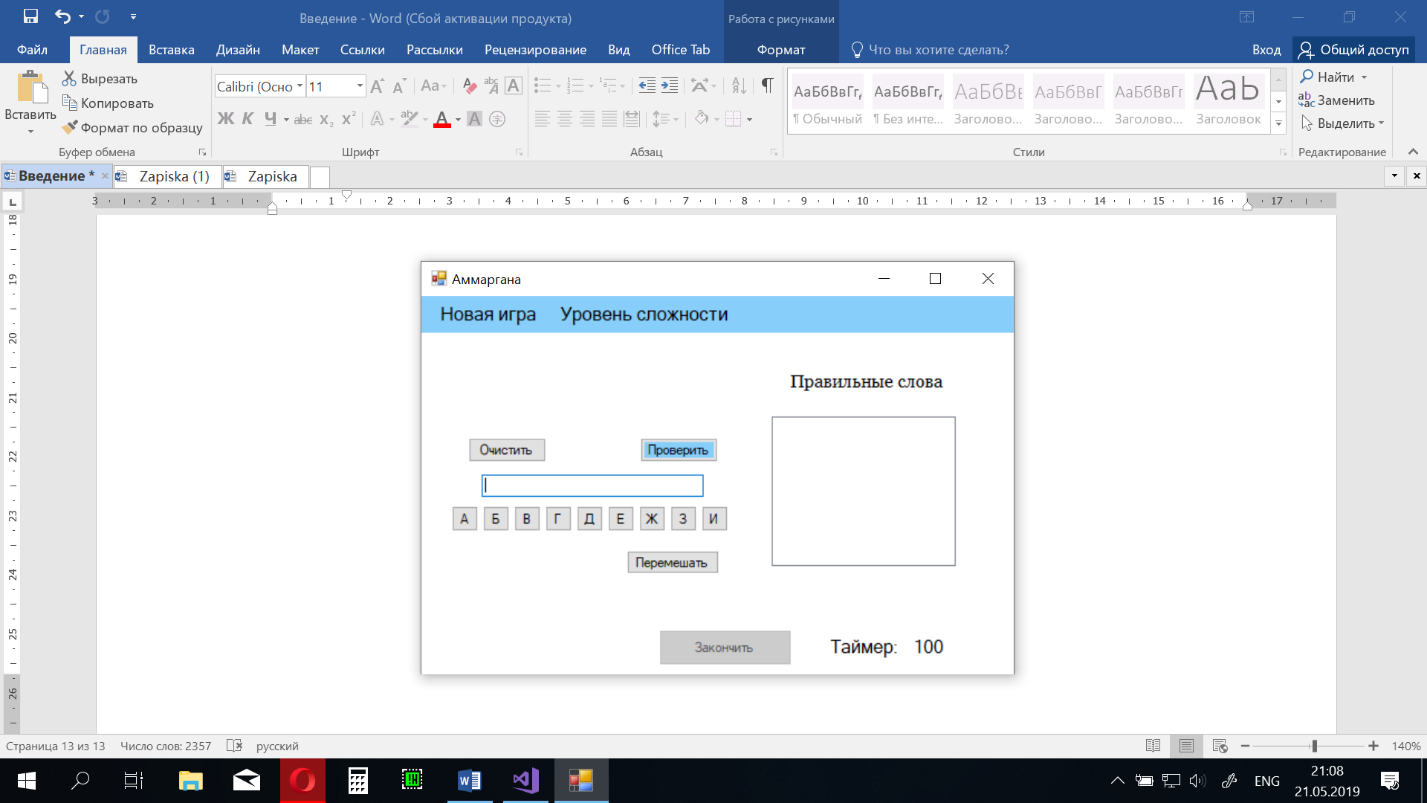
После запуска приложения открывается следующее окно формы. На меню можно увидеть кнопки «Новая игра» и «Уровень сложности». Остальные панели не активны до начала игры.

Рисунок 3.1 — Стартовое окно игрового приложения

Нажмите кнопку «Новая игра». Приложение выберет случайное слово из встроенного словаря. Длина слова зависит от выбранного уровня сложности и может составлять от 7-ми до 9-ти букв.

В результате вы увидите форму, похожую на ту, что на рисунке 3.2. Буквы выбранного слова поместятся на кнопки. При этом само слово будет перемешено для увеличения интереса.

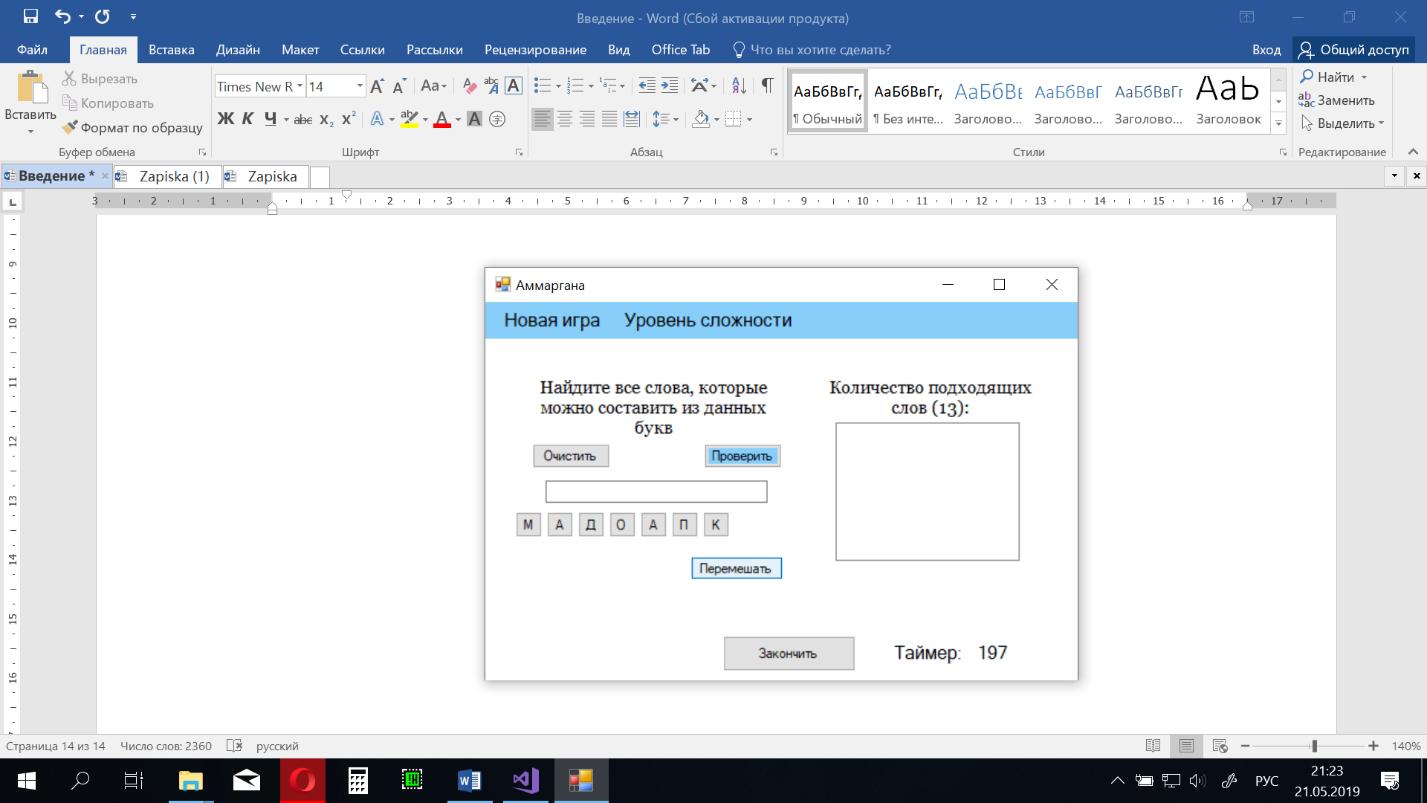


Рисунок 3.2 — Окно приложения после начала новой игры

При нажатии на кнопку «Перемешать» буквы в слове будут случайным образом перемешаны при помощи встроенного алгоритма. Результат можно увидеть на рисунке 3.3.

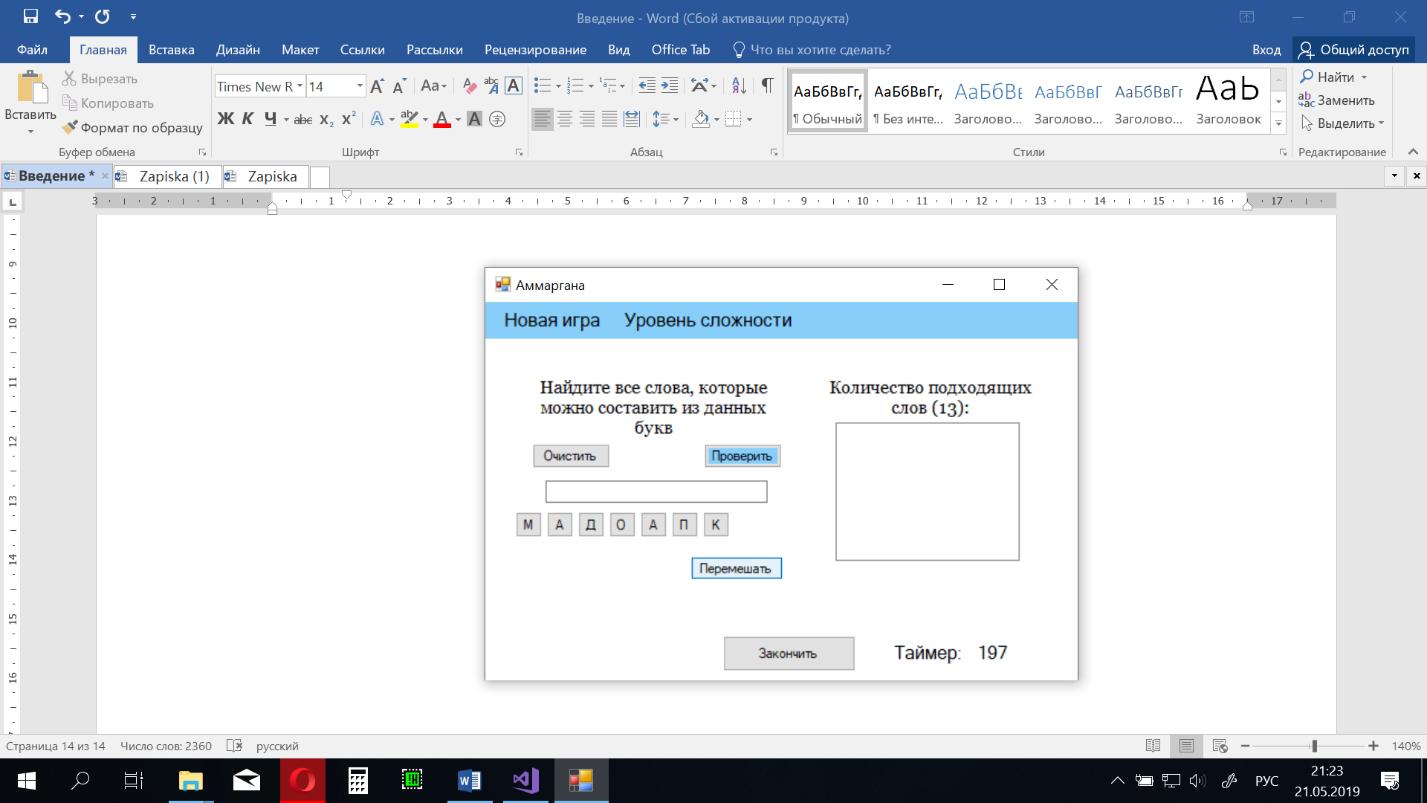
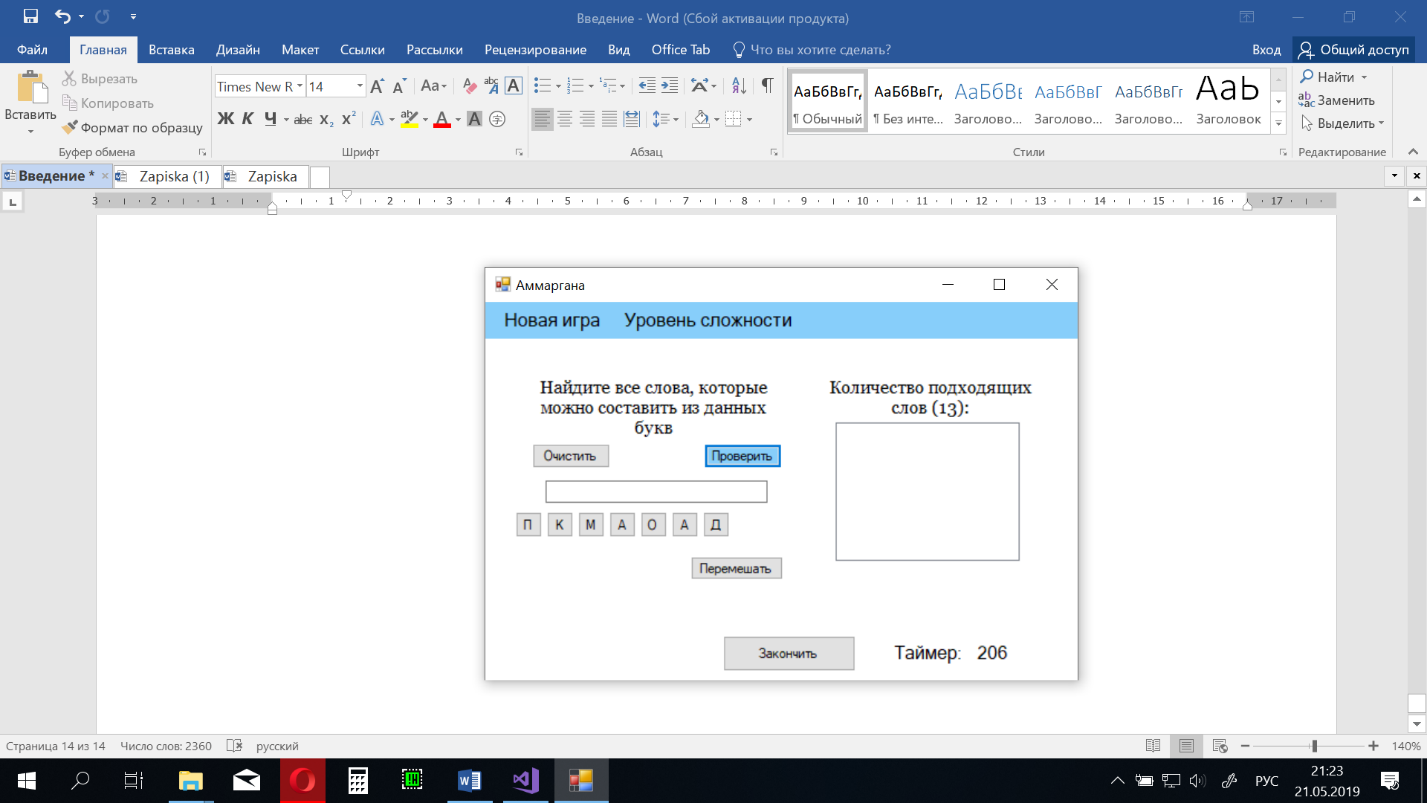


Рисунок 3.3 — Демонстрация работы кнопки «Перемешать»

Теперь игра началась. Это можно увидеть по таймеру справа внизу на форме. Попробуйте ввести слово из заданных букв. Для этого по очереди нажимайте на буквы — те, которые вы уже использовали станут неактивными.

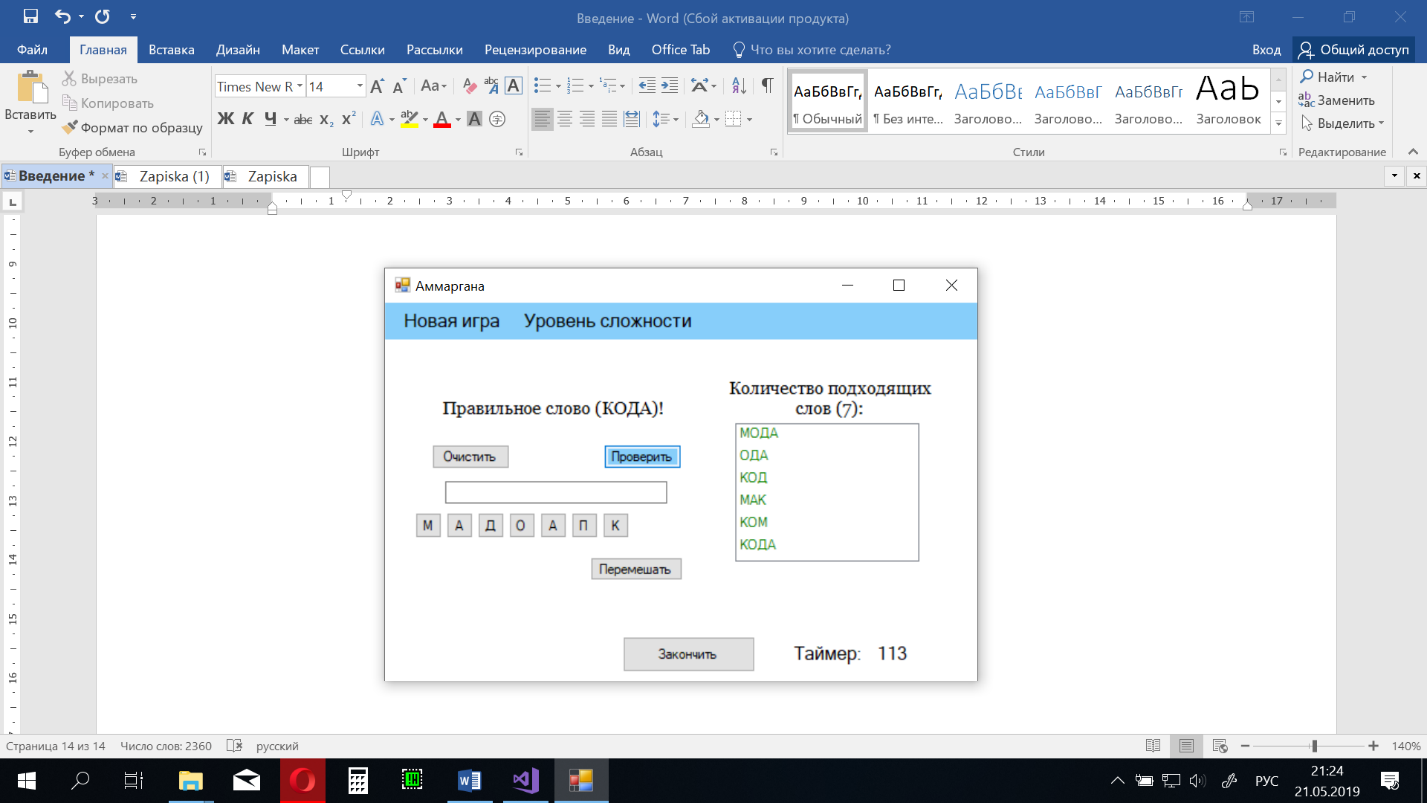
Постарайтесь найти как можно больше слов до окончания таймера. Ваши слова записываются в окошко справа. Над этим окошком мы можете увидеть количество слов, которые все еще можно составить.

Рисунок 3.4 — Демонстрация записи правильного слова

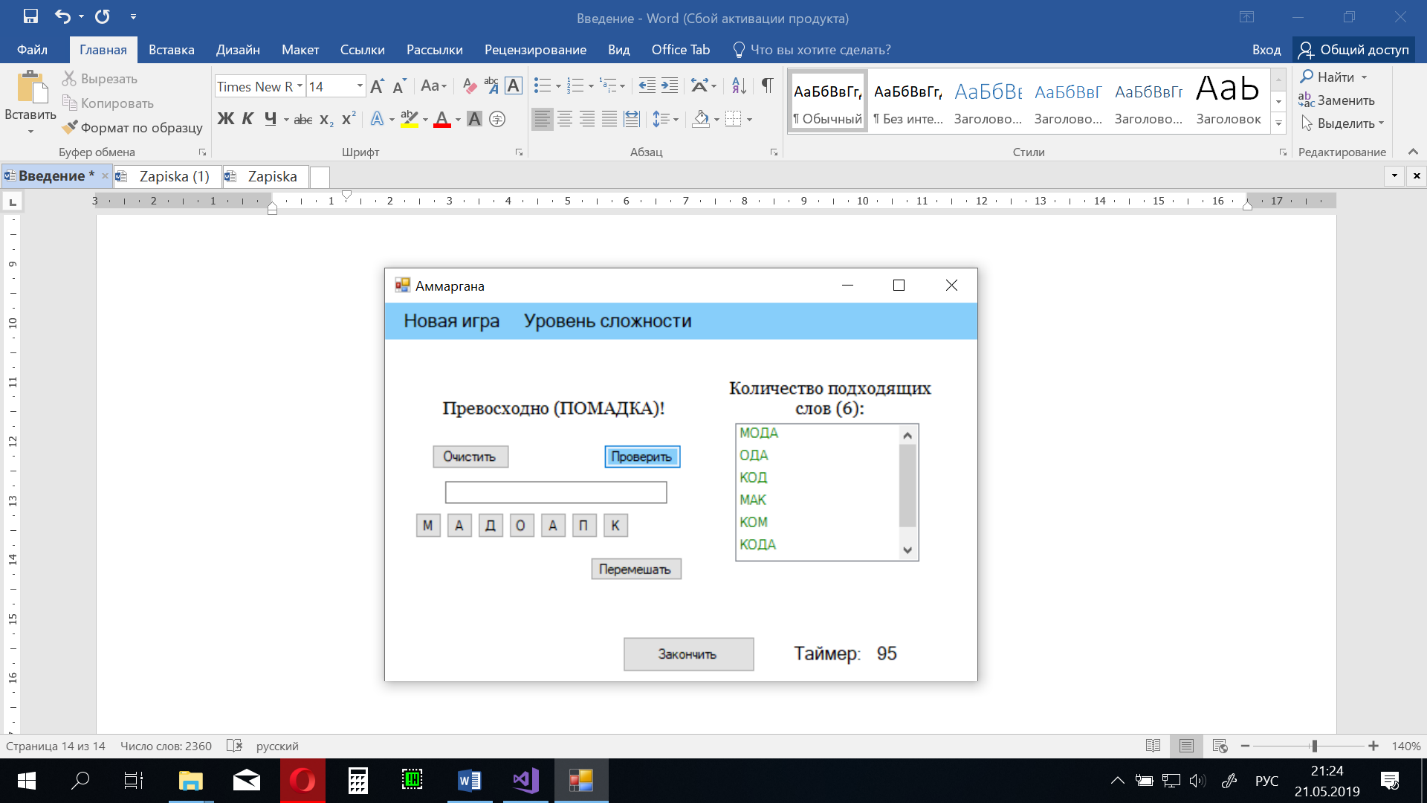
Когда вы вводите правильное слово форма выдает надпись, сообщающую, что слово возможное. Когда это слово записывается в список справа, вы можете увидеть, что количество возможных слов уменьшается — стремитесь найти все слова из возможных.

Рисунок 3.5 — Демонстрация записи искомого слова

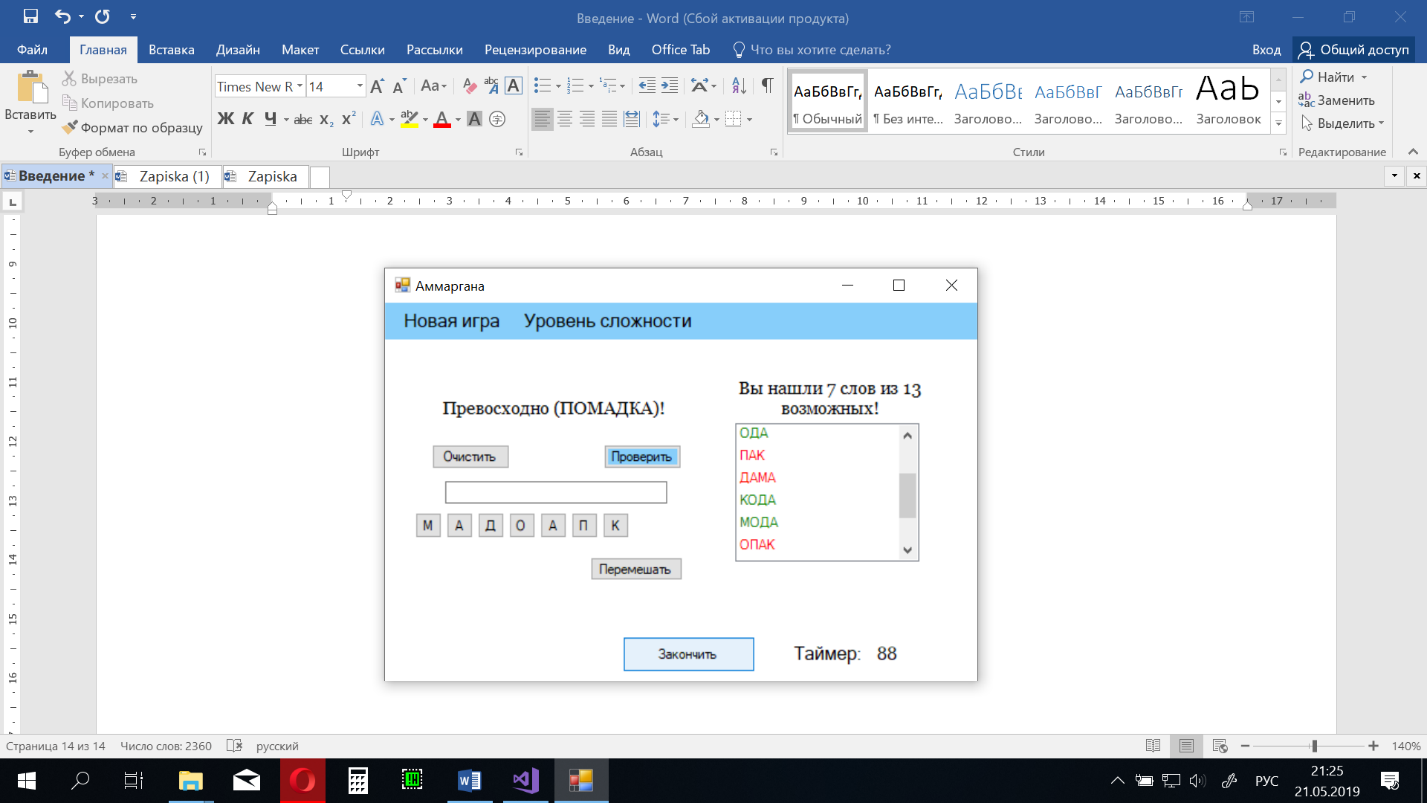
Если при составлении слова вы сможете использовать все буквы и тем самым найдете изначальное слово, форма выдаст соответствующее сообщение (рисунок 3.5).

Рисунок 3.6 — Демонстрация работы кнопки «Закончить»

При нажатии на кнопку «Закончить» игровой таймер будет остановлен, а в окошке справа будут выведены все возможные слова. Те слова, которые вы смогли найти и вписали в список будут выделены зеленым цветом — остальные красным.

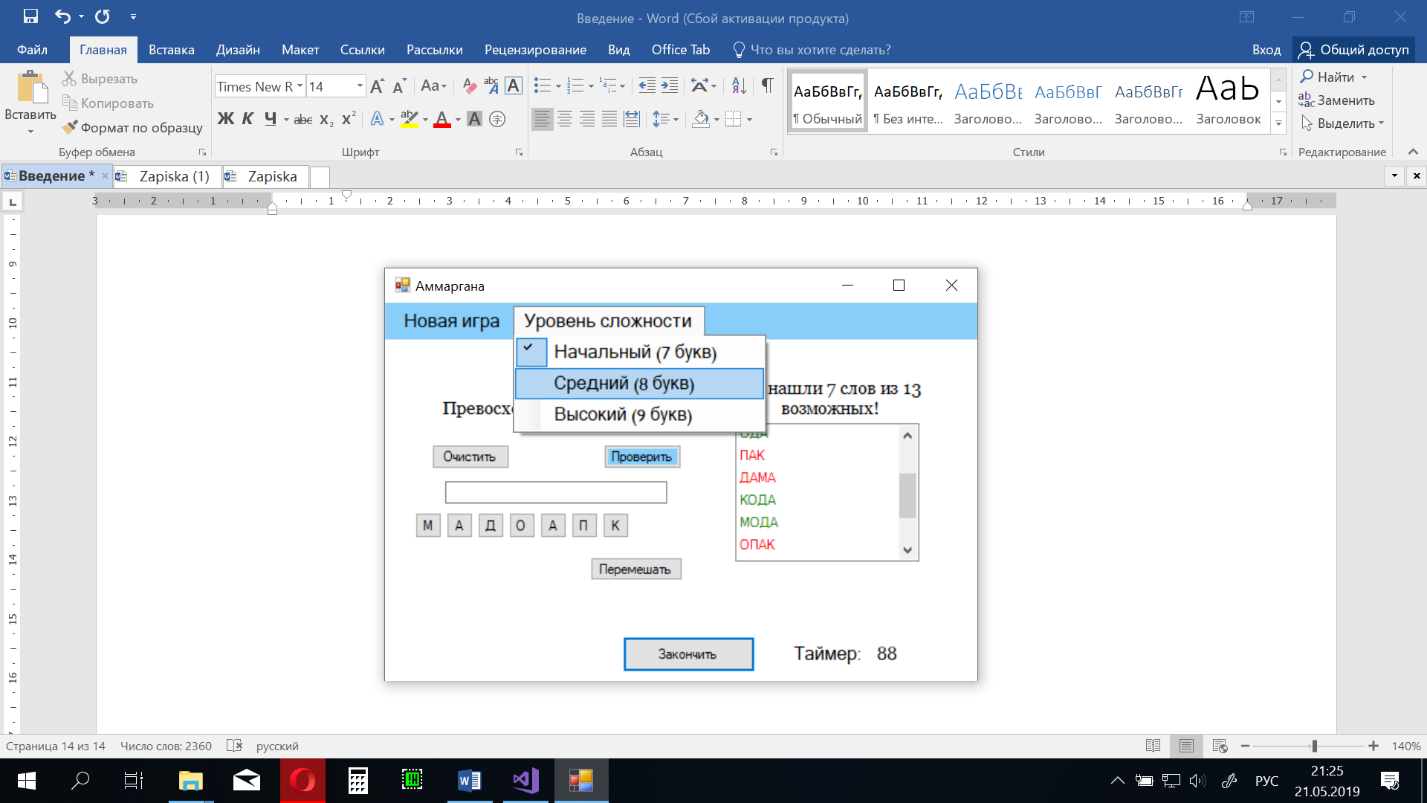
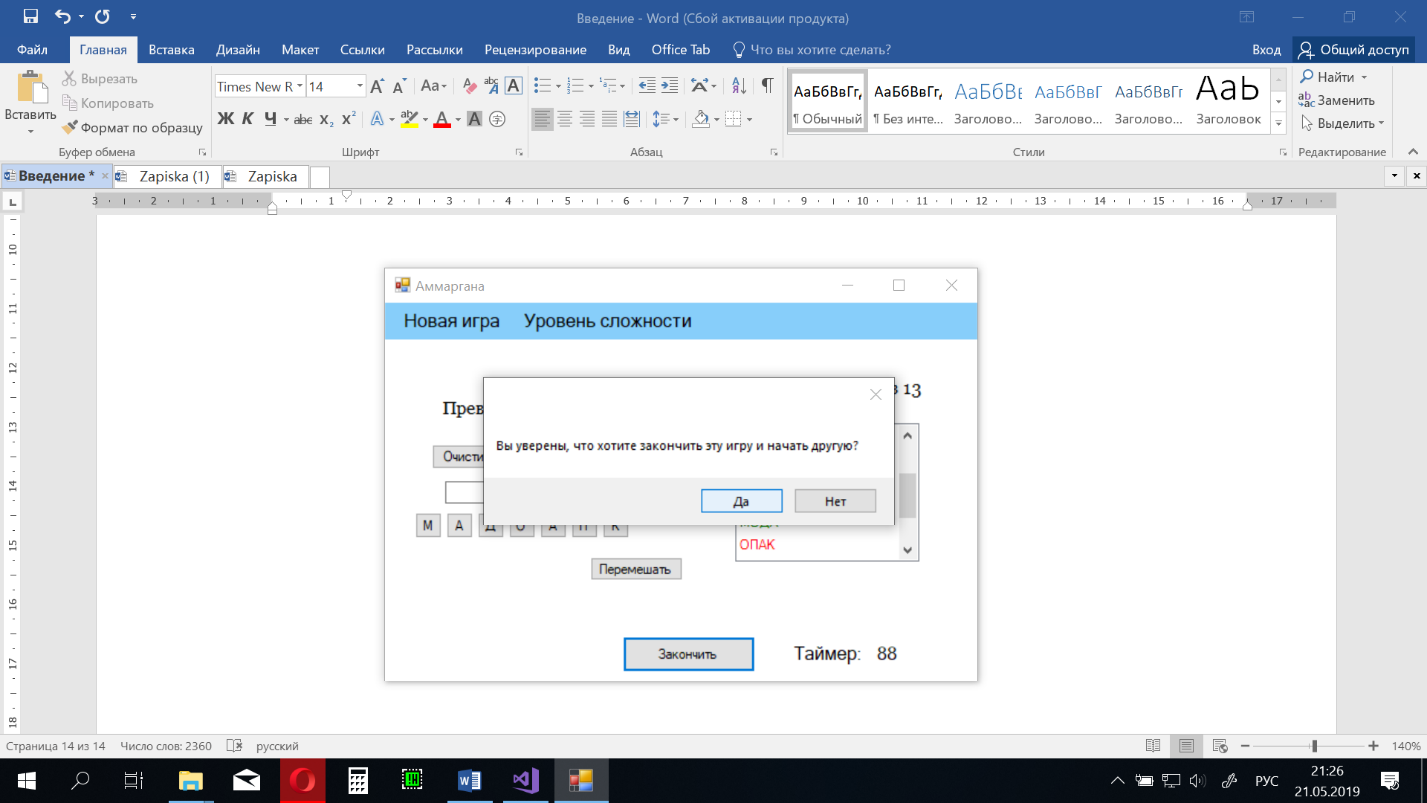
Над данным списком будет выведена информация о том, сколько слов из возможных вы нашли. Пример можно увидеть на рисунке 3.6.

Рисунок 3.7 — Демонстрация работы кнопки «Уровень сложности»

Вы можете изменить уровень сложности, нажав на кнопку «Уровень сложности» (рисунок 3.7). В игре существуют три уровня сложности. На начальном выбранное слово состоит из 7-ми букв, на среднем — из 8-ми и на высоком — из 9-ти букв.

Вы можете начать новую игру, нажав на нужный вам уровень сложности, либо нажав на кнопку «Новая игра».

Тогда приложение выдаст информационное окно с сообщением, о том, что вы хотите начать новую игру. Данное сообщение можно увидеть на рисунке 3.8.

Рисунок 3.8 — Информационное окно при начале новой игры

# **Заключение**

В ходе выполнения курсовой работы было разработано игровое приложение «Аммаргана», работающее по принципу анаграмм. В этой игре пользователь составляет слова из заданных букв, а программа проверяет их на правильность и выдает в конце игры подходящие слова. Во время выполнения данной курсовой работы было изучено немало теоретического материала, а также просмотрено и разобрано большое количество уже готовых решений тех или иных задач.

В ходе выполнения курсовой работы было выполнено следующее:

- поиск базы слов для приложения;

- разработка архитектуры приложения;

- разработка функциональных части приложения;

- разработка пользовательского интерфейса;

- написание исходного кода приложения;

- тестирование приложения.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

**Список использованных источников**

1. ProfessorWeb .NET & Web Programming [Электронный ресурс] / Режим доступа: https://professorweb.ru – Дата доступа: 20.05.2019.

2. Microsoft Руководство по языку C# [Электронный ресурс]/ msdn.microsoft.com – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> Дата доступа 20.11.2018.

3. Список существительных русского языка в txt [Электронный ресурс] / Режим доступа: <http://blog.harrix.org/article/3334> – Дата доступа: 10.05.2018.

4. METANIT.COM Сайт о программировании [Электронный ресурс] / Режим доступа: https://metanit.com. Дата доступа: 20.05.2019

5. Пацей, Н.В. Курс лекций по языку программирования С# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.