МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Вятский государственный университет»
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Допущено к защите
Руководитель проекта
_____ (Исупов К. С.)
«___» _____2021г.

Разработка генератора кроссвордов

Пояснительная записка курсового проекта по дисциплине
«Комплекс знаний бакалавра в области программного и аппаратного обеспечения вычислительной техники»
ТПЖА.09.03.01.331 ПЗ

Разработал студент группы ИВТ-22 _____/Крючков И. С./

Руководитель _____/Исупов К. С./

Консультант _____/Коржавина А. С./

Проект защищен с оценкой «_____» _____
(оценка) *(дата)*

Члены комиссии _____/_____/
(подпись)
_____/_____/
(подпись)
_____/_____/
(подпись)

Киров 2021

Реферат

Крючков И. С. Разработка генератора кроссвордов: ТПЖА.090301.331 ПЗ: Курс. проект / ВятГУ, каф. ЭВМ; рук. Исупов К. С. - Киров, 2021. – ПЗ 93 с, 8 рис., 1 табл., 3 источника.

Объект исследования и разработки – программный продукт генератор кроссвордов

Цель курсового проекта – разработать алгоритм генерации кроссворда и программный инструментарий, реализующий данный алгоритм.

Результатом выполнения курсового проекта является прототип программного обеспечения и проектная документация.

Разработанное программное обеспечение реализует функции, необходимые для предоставления пользователю возможности генерации кроссворда.

Оглавление

Введение

Кроссворд - головоломка, представляющая собой переплетение рядов клеточек, которые заполняются словами по заданным значениям.

Составление кроссворда – занятие достаточно трудоемкое и кропотливое, требующее внимательности и усидчивости. Для составления кроссвордов необходимо иметь доступ к словарям со списками слов и толкований. С появлением персональных компьютеров стали появляться программы для составления кроссвордов. В настоящее время, когда персональные компьютеры имеют оперативную память, измеряемую в гигабайтах, и процессоры, которые имеют в своем составе несколько ядер, а также операционные системы, поддерживающие параллельные вычисления, позволяют реализовывать генераторы кроссвордов, которые заполняют средней плотности и сложности сетки за время, исчисляемое секундами.

Существующие приложения для генерации кроссвордов имеют свои плюсы и минусы. Поэтому было принято решение разработать собственную программу, для решения поставленной задачи.

1. Анализ предметной области

1.1. Актуальность темы

В настоящее время кроссворд — это одно из любимых время препровождения многих людей. Многие издания прессы содержат различные варианты кроссвордов. Процесс разгадывания кроссворда сложное и интересное занятие. В процессе разгадывания человек тренирует логическое мышление и память, узнает новую информацию. Кроме этого, кроссворды могут использоваться в образовательных целях. Например, для тестирования знаний школьников или студентов по конкретному предмету и др.

1.2. Обзор аналогов

1.2.1. Crossword Force – приложение windows для создания собственных кроссвордов.

Плюсы:

- экспорт кроссворда;
- сохранение сетки кроссворда;
- настройка оформления кроссворд.

Минусы:

- отсутствует возможность загрузить словарь;
- ограниченный функционал бесплатной версии.

1.2.2. CrosswordCreator – простой генератор кроссвордов для Windows

Плюсы:

- загрузка словаря;
- настройка оформления кроссворда;
- сохранение сетки кроссворда;
- бесплатная лицензия.

Минусы:

- отсутствует возможность экспортировать кроссворд.

1.2.3. FineCrosser Pro – приложении для генерации кроссвордов на Windows

Плюсы:

– сохранение сетки кроссворда;
– настройка оформления кроссворда;
– экспорт кроссворда.

Минусы:

– отсутствует возможность загрузить словарь;
– ограниченный функционал бесплатной версии.

В таблице 1 представлено сравнение аналогов

Таблица 1 – сравнение аналогов

	Crossword Force	Crossword Creator	FineCrosser Pro
Загрузка словаря	-	+	-
Сохранение сетки кроссворда	+	-	+
Настройка оформления кроссворда	+	+	+
Экспорт кроссворда	+	-	+
Бесплатная лицензия	-	+	-

В результате сравнения аналогов было принято решение разработать программу, включающее преимущества рассмотренных приложений.

2. Постановка задачи

Разработать программу «Генератор кроссвордов» для интерактивного создания кроссвордов с возможностью сохранения результатов генерации для дальнейшей работы с ними.

2.1.1. Требования к функциям

В программе должны быть реализованы следующие функции:

- загрузка словаря слов;
- генерация кроссворда по заданной сетке;
- сохранение сетки;
- экспорт готового кроссворда;
- добавление слов в словарь;
- автосохранение сетки кроссворда.

2.1.2. Требования к интерфейсу

К основным требованиям можно отнести:

- интерактивное поле для формирования сетки кроссворда;
- предпросмотр результата при экспорте;
- простой интуитивно понятный интерфейс.

2.1.3. Требования к входным и выходным данным

Пользователь загружает .txt файл, состоящий из слов с вопросами в формате «слово=вопрос».

Результатом сохранения сетки является текстовый файл формата JSON, который включает в себя минимально необходимую информацию о состоянии поля кроссворда.

Экспорт кроссворда предоставляет возможность сохранить изображение кроссворда с вопросами в формате PNG, а также вывести результат на печать.

2.1.4. Требования к программному обеспечению

Для корректной работы программы необходимы:

- операционная система Windows 7/8/8.1/10/11.

3. Разработка структуры программы

3.1. Функциональное взаимодействие блоков

Обозначенные выше функции можно разбить на следующие блоки.

1) Блок импорта информации.

Данный блок включает в себя следующие функции:

- загрузка словаря;
- открытие сетки;
- добавление слова в словарь.

2) Блок экспорта информации:

Данный блок включает в себя следующие функции:

- сохранение сетки кроссворда;
- экспорт сгенерированного кроссворда.

3) Блок генерации кроссворда:

Данный блок включает в себя следующие функции:

- быстрая генерация кроссворда;
- стандартная генерация кроссворда.

4) Блок вывода на экран:

Данный блок включает в себя следующие функции:

- вывод поля для генерации;
- вывод списка вопросов для сгенерированного кроссворда;
- вывод предпросмотра поля при экспорте кроссворда.

5) Блок настройки поля

Данный блок включает в себя следующую функцию:

- изменение размеров поля кроссворда.

Разработанная схема взаимодействия блоков представлена на рисунке 1.

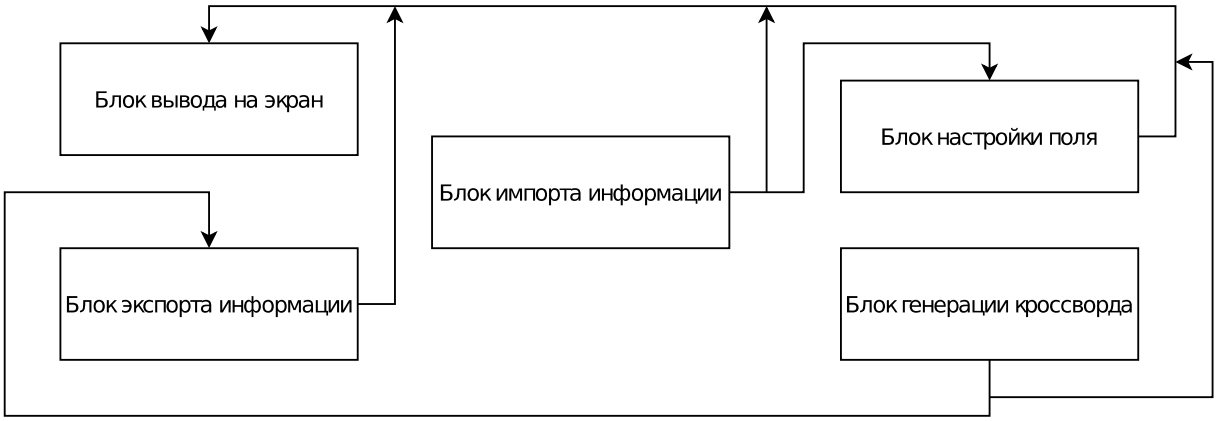

Рисунок 1 – Схема взаимодействия функциональных блоков

3.2. Разработка алгоритмов

Основной функцией программы является генерация кроссворда по заданной сетке. Для решения данной задачи было реализовано два алгоритма различной алгоритмической сложности. В качестве вспомогательных функций перед генерацией происходит поиск шаблонов слов на сетке и определение пересечений найденных шаблонов. Схемы алгоритмов представлены в Приложении А и Б соответственно.

3.2.1. Алгоритм «Стандартная генерация»

В ходе выполнения алгоритма, выполняется перебор слов с возвратом. Плюсом данного метода является то, что он гарантирует составление кроссворда, если точно известно, что из заданного набора слов можно заполнить сетку. Особенность алгоритма в том, что перебор происходит по списку только тех слов из словаря, которые с учетом исключительно своей длины могут быть установлены в какое-либо место на заданной сетке. Перебор шаблонов слов на сетке происходит по убыванию количества пересечений у данного шаблона. Отсюда первым алгоритм пробует подставить слово в шаблон с наибольшим количеством пересечений, так как подобрать слова с наименьшим числом пересечений гораздо эффективнее. Проверка возможности установки слова в данный шаблон выполняется с помощью соответствия текущего слова регулярному выражению для текущего шаблона. Слова, которые уже были установлены на сетку ранее пропускаются. После установки слова, алгоритм продолжает работу, переходя к следующему шаблону. При каждом случае невозможности установить слово в шаблоне после перебора всех возможных вариантов, происходит переход к предыдущему шаблону, в котором установленное ранее слово заменяется на следующее, доступное для установки в данном шаблоне слово. Если после перебора всех слов и шаблонов не удалось составить кроссворд или в случае, если для некоторого шаблона в списке нет слов, возвращается ошибка генерации кроссворда.

Минусом данного алгоритма является его время выполнения. Оценка сложности вычислений выражается по формуле O(n!).

Схема алгоритма представлена на рисунке 2.

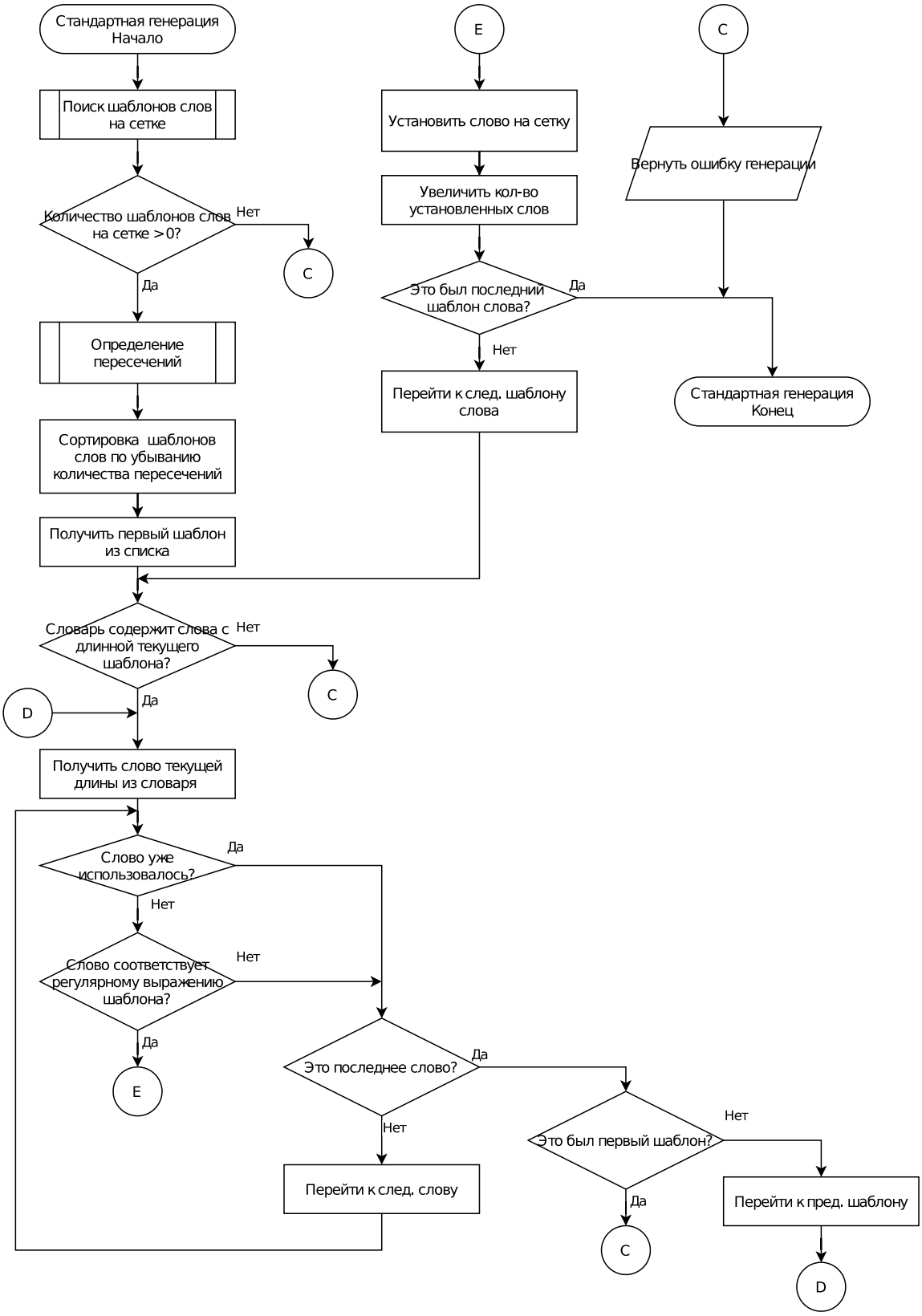

Рисунок 2 – Схема алгоритма «стандартной генерации»

3.2.2. Алгоритм «Быстрая генерация»

В ходе данного алгоритма производиться расстановка слов без возврата. Перебор шаблонов слов на сетке происходит по убыванию количества пересечений у данного шаблона. Расстановка начинается с шаблона с наибольшим числом пересечений. Первое слово выбирается произвольно из списка слов той длины, которые могут быть установлены в данный шаблон. При заполнении последующих шаблонов происходит проверка возможности установки слова с помощью соответствия текущего слова регулярному выражению для текущего шаблона. Слова, которые уже были установлены на сетку ранее пропускаются. После установки слова, а также в случае, если в текущий шаблон не удалось установить слово, алгоритм продолжает работу, переходя к следующему шаблону. Особенность алгоритма заключается в том, что он не гарантирует полное заполнение сетки словами. В результате генерации некоторые шаблоны могут остаться незаполненными. Это компенсируется достаточно большим количеством слов в словаре, а также возможностью быстро сгенерировать новую сетку. Алгоритмическая сложность данного алгоритма O(n).

Схема алгоритма представлена на рисунке 3.

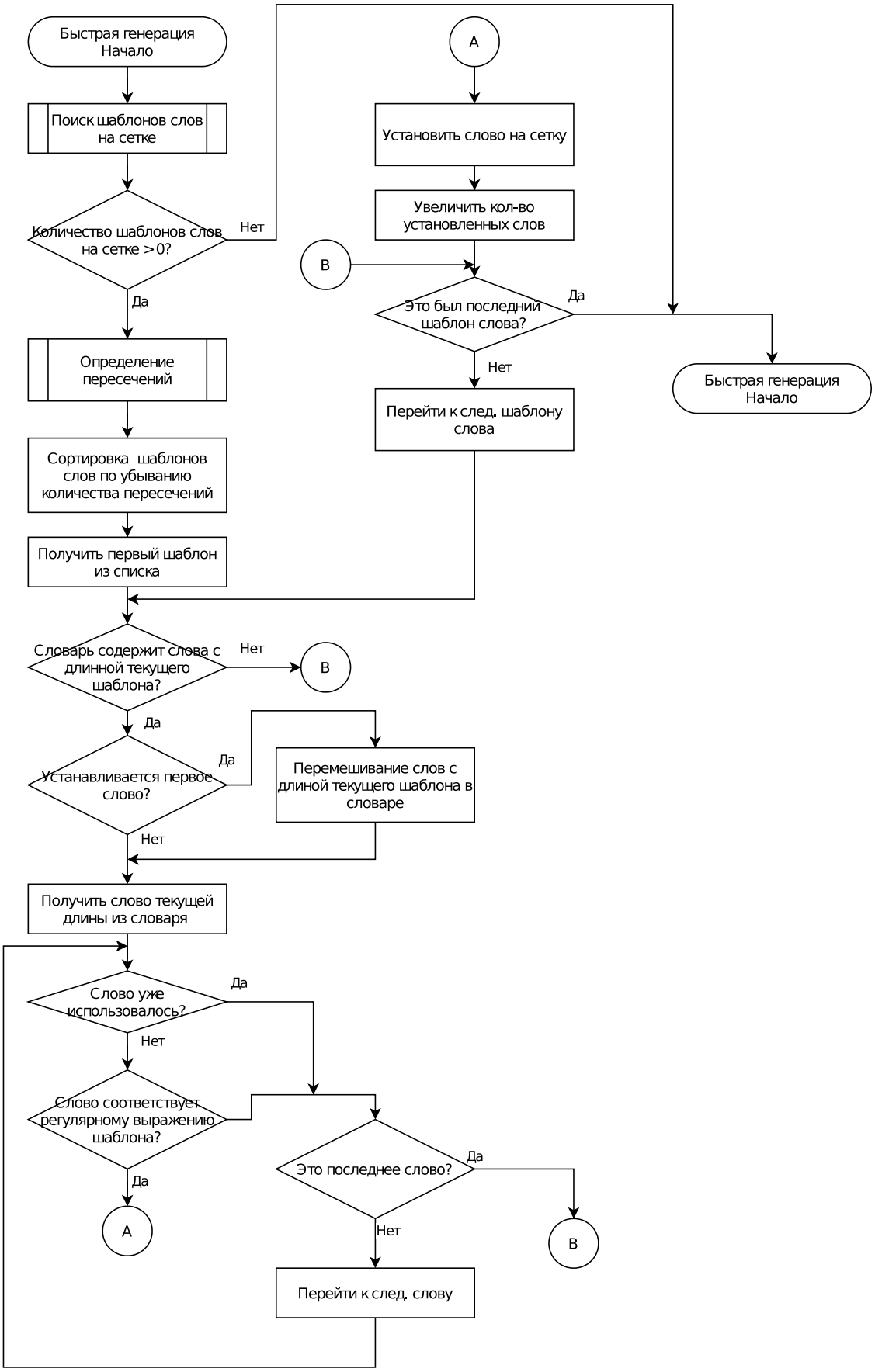

Рисунок 3 – Схема алгоритма «быстрой генерации»

3.3. Разработка интерфейса пользователя.

Интерфейс пользователя должен решать поставленную задачу, быть интуитивно понятным, не перегружен лишними элементами, а процесс взаимодействия пользователя с ним предсказуемым. Пользовательский интерфейс должен предоставлять доступ ко всему перечню функциональных возможностей, предусмотренных приложением.

С учетом данных принципов на главном окне приложения было решено разместить меню программы, интерактивное поле кроссворда, список вопросов сгенерированного кроссворда, настройки параметров и элементы управления.

Меню программы состоит из следующих пунктов:

1. Файл:
   - открыть словарь;
   - добавить слово;
   - открыть макет;
   - сохранить макет;
   - выход.
2. Экспорт – окно экспорта кроссворда.
3. Справка – окно информации о программе и разработчике.

Интерактивное поле кроссворда представляет собой сетку установленного пользователем размера, с которой он может взаимодействовать с помощью мыши. На этапе редактирования пользователь формирует макет кроссворда, который при генерации будет заполнен словами, путем выбора клеток на сетке.

Список вопросов формируется после генерации кроссворда. Состоит из номера слова на сетке и вопроса, для этого слова.

Настройка параметров кроссворда включает в себя настройку ширины и высоты поля кроссворда. Минимальный размер - 5x5 клеток, максимальный – 50x50 клеток.

Элементы управления включают в себя:

– Кнопка «Генерация» - запускает генерацию кроссворда.
– Кнопка «Очистить» - очищает поле кроссворда.
– Флажок выбора режима генерации – выбор алгоритма генерации кроссворда – «Быстрая генерация» и «Стандартная генерация».

Окно экспорта кроссворда состоит из панели параметров экспорта и предпросмотра результата.

В панели параметров экспорта указываются элементы кроссворда, которые будут экспортированы: сетка с вопросами, только сетка либо только вопросы. Выбирается оформление кроссворда: в «стандартном виде», где все клетки, в которых должны стоять буквы – белые, все остальные клетки поля – черные, либо в том виде, в котором он представлен в программе. Определяется вывод ответов на сетке – с включенной опцией на сетке будут отображаться слова, с выключенной опцией, будет выведена только сетка с номерами слов.

Внизу окна располагаются две кнопки:

– Сохранить – экспорт кроссворда в файл изображения .PNG.
– Печать – печать кроссворда.

Экранные формы программы представлены в приложении В.

4. Программная реализация

Для разработки приложения был выбран компилируемый язык программирования C# с использованием платформы WPF – Windows Presentation Foundation.

C# является языком программирования, который разработан для создания множества приложений, работающих в среде .NET Framework. Язык C# прост, типобезопасен и объектно-ориентирован.

Платформа WPF является часть экосистемы платформы .NET и представляет собой подсистему для построения графических интерфейсов. WPF имеет ряд существенных преимуществ по сравнению с WinForms:

- возможность декларативного определения графического интерфейса с помощью специального языка разметки XAML;
- независимость от разрешения экрана;
- аппаратное ускорение графики – использование DirectX для визуализации.

В качестве среды разработки программного обеспечения выбрана Microsoft Visual Studio 2019.

Помимо стандартных ресурсов .NET Framework был использован сторонний пакет компонентов Extended Wpf Toolkit для реализации поля ввода целых чисел типа «счетчик».

Листинг программы приведен в приложении Е.

5. Заключение.

В ходе выполнения курсового проекта было разработано программное обеспечение – «Генератор кроссвордов», выполняющее функции загрузки словаря слов с вопросами, интерактивного построения сетки кроссворда пользователем, генерации кроссворда, экспорта кроссворда.

В качестве направления дальнейшего развития можно выбрать поддержку других типов кроссвордов, механизм ручного редактирования кроссворда.

Библиографический список

1. Кроссворд – Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Кроссворд. Загл. с экрана. – англ.
2. Документация по C# - Microsoft [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/. Загл. с экрана. – англ.
3. Windows Presentation Foundation – Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Windows_Presentation_Foundation. Загл. с экрана – англ.

Приложение А

(Обязательное)

Схема алгоритма поиска шаблонов слов на сетке

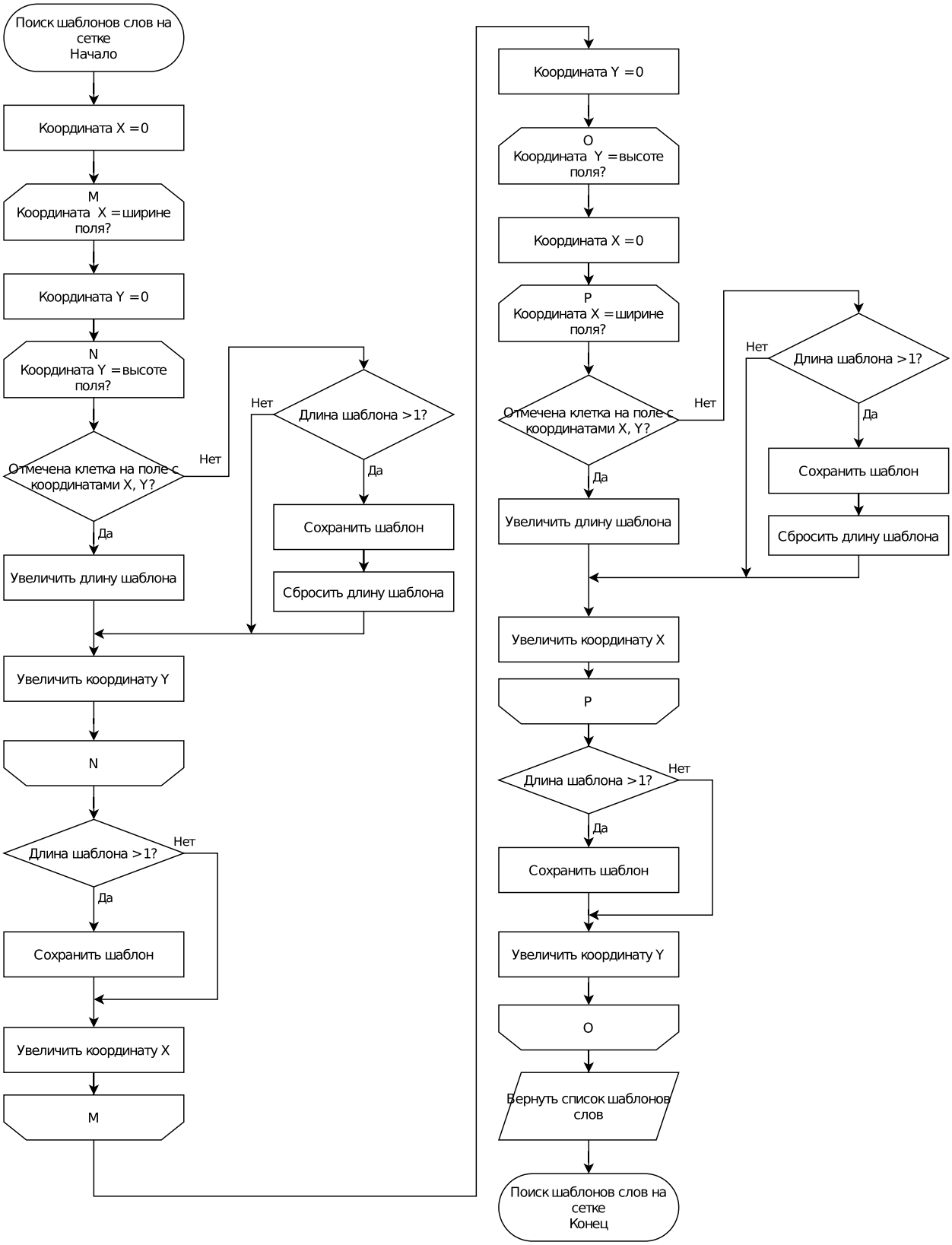

Рисунок 4 - Схема алгоритма поиска шаблонов слов на сетке

Приложение Б

(Обязательное)

Схема алгоритма определения пересечений

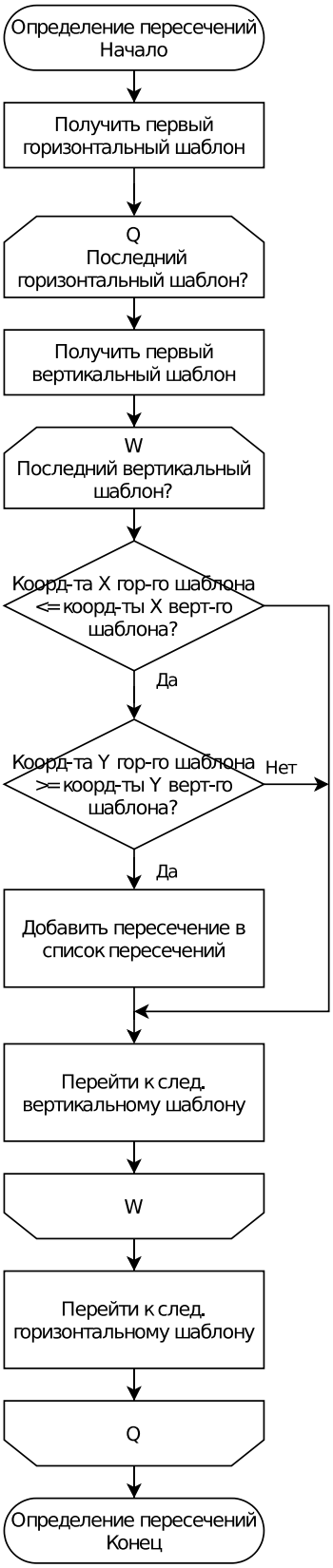

Рисунок 5 - Схема алгоритма определения пересечений.

Приложение В

(Обязательное)

Экранные формы

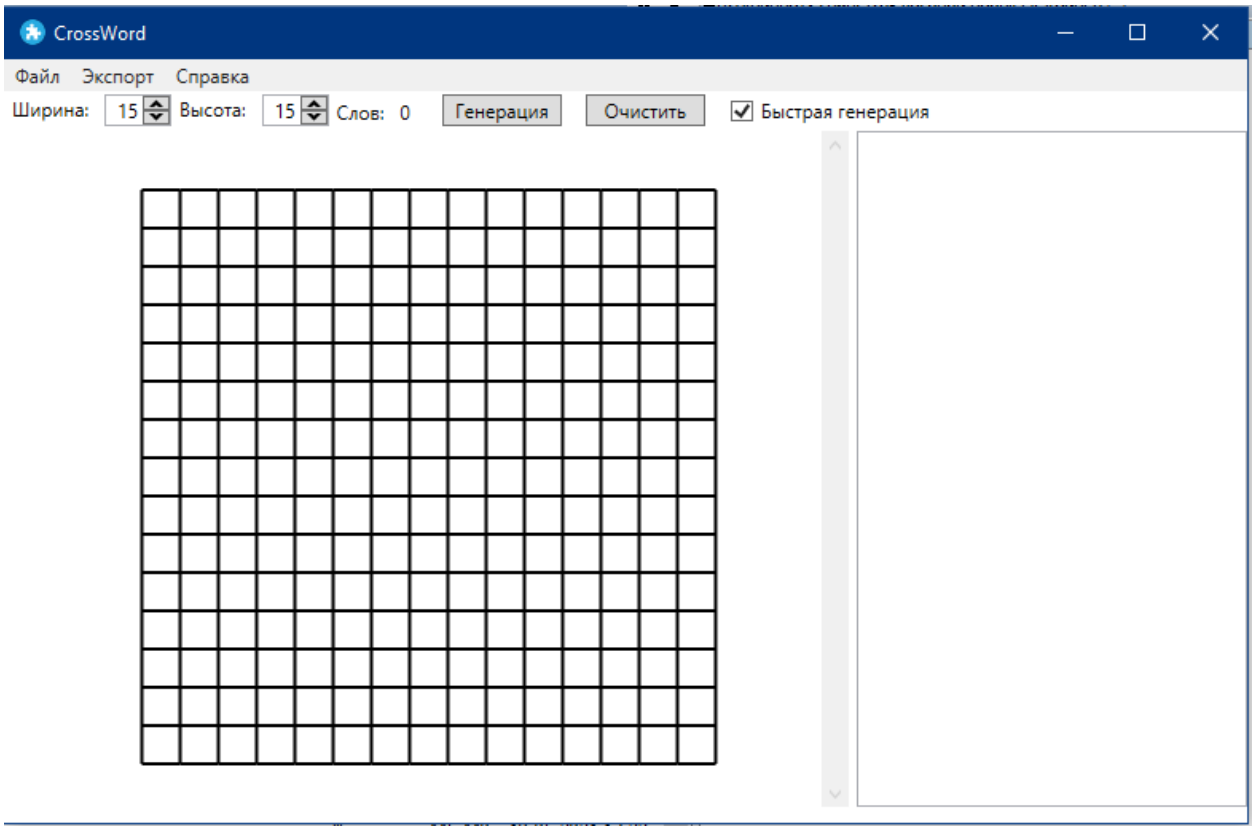

Рисунок 6 – Экранная форма главного окна программы

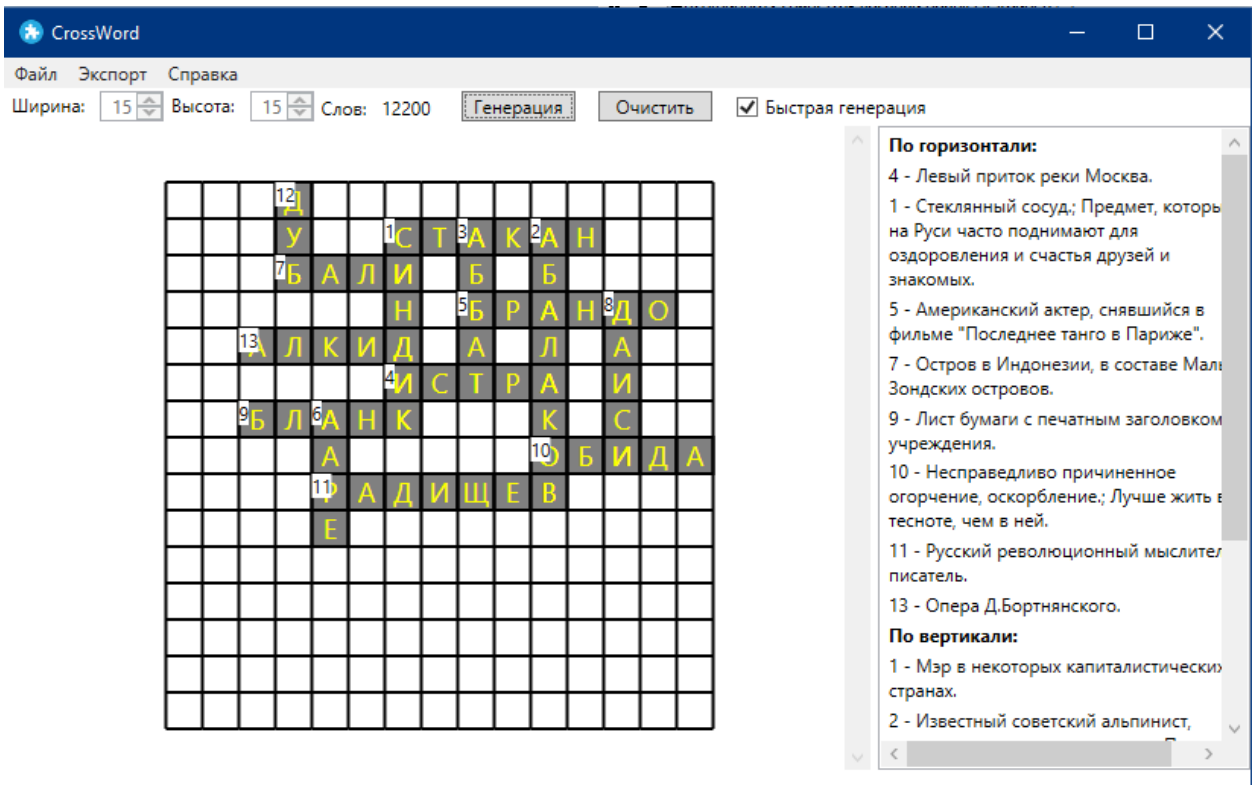

Рисунок 7 – Экранная форма сгенерированного кроссворда

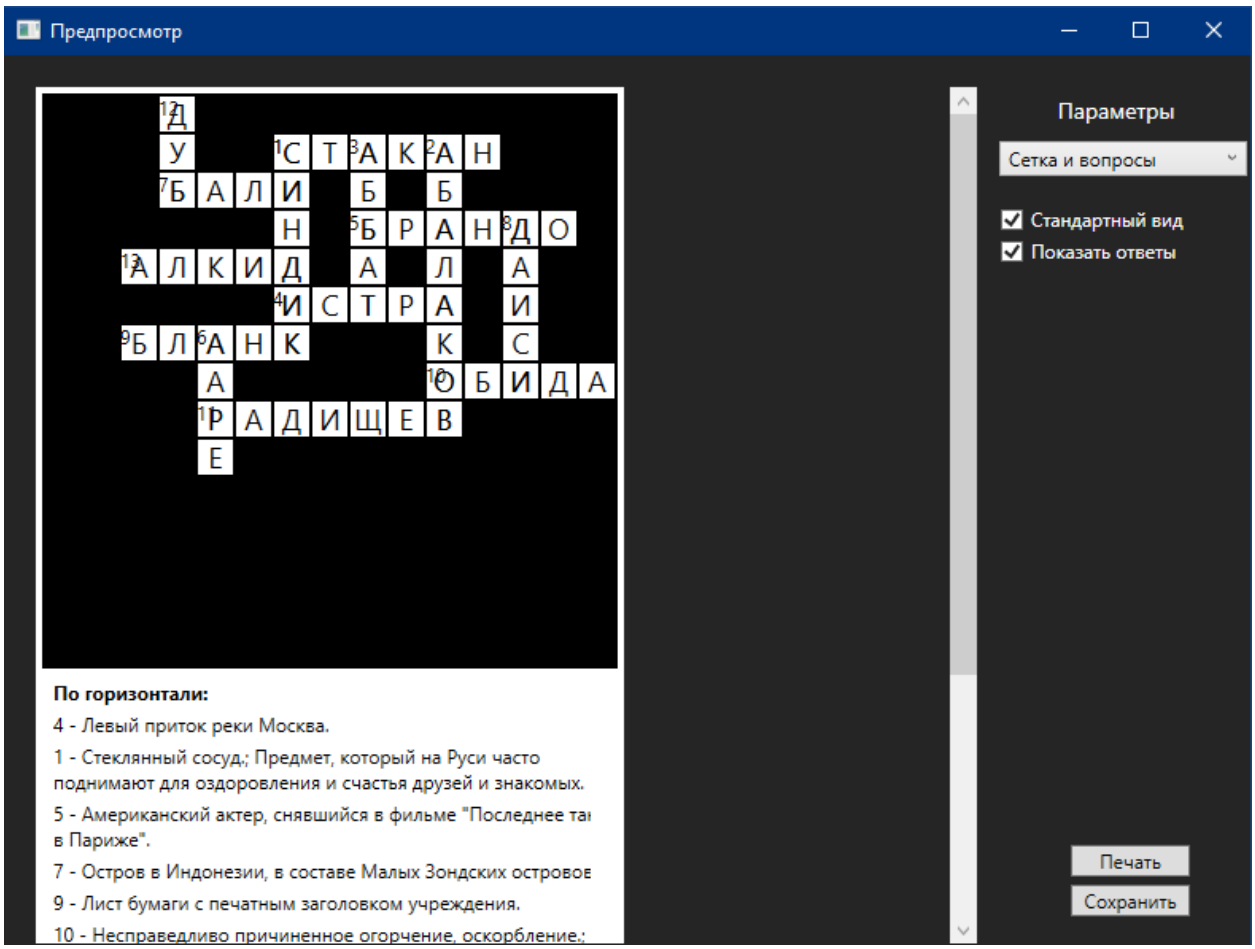

Рисунок 8 – Экранная форма окна экспорта

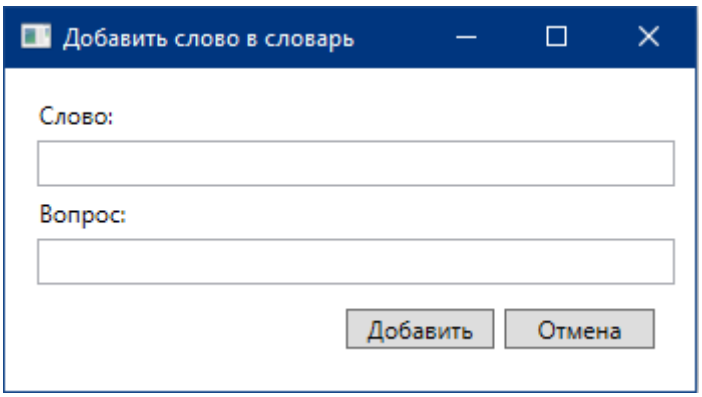

Рисунок 8 – Экранная форма окна добавления слова в словарь

Приложение Г

(Обязательное)

Листинг кода

CW_content.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.IO;
using System.Text.RegularExpressions;
using System.Windows;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Windows.Threading;
namespace Crossword
{
   public class CW_Content
   {
      // словарь слов (длина слова, список слов этой длины)
      public Dictionary<int, List<CW_word_list_item>> items_list = new Dictionary<int,
List<CW_word_list_item>>();
      // сетка из слов
      public Dictionary<int, CW_word_item> words_grid = new Dictionary<int, CW_word_item>();
      List<CW_words_intersections> intersections_list = new List<CW_words_intersections>();
      public int words_num = 0; // кол-во слов в загруженном словаре
      int grid_words_count = 0;  //кол-во слов на сетке
      public int count_seted_words = 0; // кол-во установленных на поле слов
```

```
public bool editingMode = true;
MainWindow mw = null;
// статус процесса генерации
bool generating = false;
public bool dictOpened = false;
public string dictFileName = null;
public CW_Content(MainWindow mawi)
{
    mw = mawi;
}
public string maketSaveFilePath = null;
public bool hasChanges = false;
// чтение файла с key=val (ответ=вопрос)
public void Open()
{
    if (dictOpened == false)
    {
        OpenFileDialog _dialog = new OpenFileDialog();
        _dialog.Filter = "Text files (*.txt)|*.txt";
        _dialog.ShowDialog();
        if (_dialog.FileName.Trim().Length == 0)
            return;
        dictFileName = _dialog.FileName.Trim();
        mw.Dispatcher.Invoke((Action)(() =>
        {
            mw.CW_info_label.Content = "Открытие файла";
        }));
        mw.fileopening = true;
        try
```

```
{
    foreach (string line in File.ReadAllLines(_dialog.FileName.Trim(), Encoding.Default))
    {
        string[] keyvalue = line.Split('=');
        if (keyvalue.Length == 2)
        {
            int lword = keyvalue[0].Length;
            string answ = keyvalue[0].ToUpper();
            string qst = keyvalue[1];
            if (lword >= 2)
            {
                List<CW_word_list_item> _words = null;
                // если в списке есть массив для слов длины lword
                if (items_list.ContainsKey(lword))
                    _words = items_list[lword];
                else
                {
                    _words = new List<CW_word_list_item>();
                    items_list.Add(lword, _words);
                }
                bool _exists = false;
                IEnumerator<CW_word_list_item> _items = _words.GetEnumerator();
                _items.Reset();
                while (!_exists && _items.MoveNext())
                {
                    CW_word_list_item _item = _items.Current;
                    if (_item.answer == answ)
                        _exists = true;
                }
```

```
                    if (!_exists)
                    {
                        _words.Add(new CW_word_list_item(lword, qst, answ));
                        words_num++;
                    }
                }
            }
        }
        catch
        {
            MessageBox.Show("Ошибка открытия словаря");
            mw.fileopening = false;
        }
        mw.Dispatcher.Invoke((Action)(() =>
            {
                mw.onFileOpened(words_num);
            }));
            dictOpened = true;
    }
    else
    {
        MessageBox.Show("Словарь уже загружен");
    }
}
public void words_list_clear()
{
    items_list.Clear();
}
```

```
public void grid_words_list_clear()
{
    words_grid.Clear();
    grid_words_count = 0;
    count_seted_words = 0;
    mw.CW_answers_list.Items.Clear();
}
// формирование регулярного выражения шаблона слова
public string getPattern(CW_word_item item)
{
    string regular = "^";
    int len = item.length;
    int orientation = item.orientation;
    int dx = 0;
    int dy = 0;
    // если горизонтальная ориентация слова
    if(orientation == 0)
        dx = 1;
    else
        dy = 1;
    // если сложность слова 0 (нет пересечений)
    if (item.complexity == 0)
    {
        regular += "[а-я]{" + len + "}";
    }
    else
    {
        int numRep = 0;
```

```
// цикл по каждому символу слова
for (int i = 0; i < len; i++)
{
   bool f = false;
   int idInter = 0;
   // цикл по пересечениям слова
   for (int j = 0; j < item.complexity; j++)
   {
      CW_words_intersections its = intersections_list[item.intersects[j]];
      if (item.x + dx * i == its.x && item.y + dy * i == its.y)
      {
         f = true;
         idInter = j;
      }
   }
   // если не было пересечения
   if (f == false)
   {
      // увеличить кол-во ячеек до пересечения
      numRep++;
   }
   else
   {
      CW_words_intersections its = intersections_list[item.intersects[idInter]];
      bool t = false;
      // если нет символа на пересечении
      if (its.ch == ' ')
      {
         numRep++;
```

```
                t = true;
            }

            // если на пересечениии есть символ
            if (t == false)
            {
                // если есть ячейки до пересечения
                if (numRep > 0)
                {
                    regular += "[а-я]{" + numRep + "}";
                    numRep = 0;
                }
                regular += its.ch;
            }
        }

    }
    if (numRep > 0)
    {
        regular += "[а-я]{" + numRep + "}";
    }
  }
  regular += "$";
  return regular;
}
 // быстрая генерация
public void Generate(int w, int h, List<CwCell> grid)
{
  // если генерация уже не запущена
```

```
if (!generating)
{
    // если есть загруженные слова
    if (words_num > 0)
    {
        mw.CW_info_label.Content = "Генерация";
        // выключаем кнопку "генерировать"
        mw.generate_button.IsEnabled = false;
        mw.cw_grid_width.IsEnabled = false;
        mw.cw_grid_height.IsEnabled = false;
        editingMode = false;
        generating = true;
        grid_words_count = 0;
        words_grid.Clear();
        // поиск вертикальных слов
        // цикл по X
        for (int i = 0; i < w; i++)
        {
            // координаты первый точки
            int x0 = 0;
            int y0 = 0;
            // длина слова
            int ls = 0;
            // цикл по Y
            for (int j = 0; j < h; j++)
            {
                int xy = mw.fromXYtoSingle(i, j);
                if (grid[xy].selected)
                {
```

```
                    if (ls == 0)
                    {
                        x0 = i;
                        y0 = j;
                    }
                    ls++;
                    // если последняя итерация
                    if (j == h - 1)
                    {
                        words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y = y0, orientation = 1, answer = "", question = "", intersects = new List<int>(), isdefined = false, word_number = 0, word_inlist_id = 0 });
                        grid_words_count++;
                        ls = 0;
                    }
                }
                else
                {
                    if (ls > 1)
                    {
                        words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y = y0, orientation = 1, answer = "", question = "", intersects = new List<int>(), isdefined = false, word_number = 0, word_inlist_id = 0 });
                        grid_words_count++;
                    }
                    ls = 0;
                }
            }
        }
        // поиск горизонтальных слов
```

```
// цикл по Y
for (int i = 0; i < h; i++)
{
    // координаты первый точки
    int x0 = 0;
    int y0 = 0;
    // длина слова
    int ls = 0;
    // цикл по X
    for (int j = 0; j < w; j++)
    {
        int xy = mw.fromXYtoSingle(j, i);
        if (grid[xy].selected)
        {
            if (ls == 0)
            {
                x0 = j;
                y0 = i;
            }
            ls++;
            // если последняя итерация
            if (j == w - 1)
            {
                words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y = y0, orientation = 0, answer = "", question = "", intersects = new List<int>(), isdefined = false, word_number = 0, word_inlist_id = 0 });
                grid_words_count++;
                ls = 0;
            }
```

```
                    }
                    else
                    {
                        if (ls > 1)
                        {
                            words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y = y0, orientation = 0, answer = "", question = "", intersects = new List<int>(), isdefined = false, word_number = 0, word_inlist_id = 0 });
                            grid_words_count++;
                        }
                        ls = 0;
                    }
                }
            }
            // если есть шаблоны слов на сетке
            if (grid_words_count > 0)
            {
                // определение пересечений
                intersections();

                //System.Diagnostics.Debug.WriteLine(grid_words_count);
                // сортировка сетки слов по сложности, длине
                var items = from pair in words_grid
                            orderby pair.Value.complexity descending, pair.Value.length
                            select pair;
                words_grid = items.ToDictionary(x => x.Key, x => x.Value);
                bool first_word = true;
                foreach (var word_data in words_grid)
                {
```

```
CW_word_item word = word_data.Value;
int len = word.length;

// если в списке есть слова данной длины
if (items_list.ContainsKey(len))
{
   // если устанавливаем первое слово
   if (first_word)
   {
      // перемешиваем массив, чтобы слово каждый раз было случайным
      var rand = new Random();
      var randomList = items_list[len].OrderBy(x => rand.Next()).ToList();
      items_list[len] = randomList;
      first_word = false;
   }
   foreach (CW_word_list_item item in items_list[len])
   {
      // если слово еще не использовалось
      if (!item.isUsed)
      {
         string pattern = getPattern(word);
         string text = item.answer;
         string qst_text = item.question;
         // если строка совпадает с регуляркой
         if (Regex.IsMatch(text, pattern, RegexOptions.IgnoreCase))
         {
            word.updateWord(text, qst_text, intersections_list);
            // увеличить кол-во установленных слов
            count_seted_words++;
```

```
                        item.isUsed = true;
                        break;
                    }
                }
            }
        }
        setNumerating();
        mw.draw_words(ref mw.cw_field, true);
        generating = false;
    }
    else
    {
        MessageBox.Show("Необходимо заполнить сетку");
        // включаем кнопку "генерировать"
        mw.generate_button.IsEnabled = true;
        mw.cw_grid_width.IsEnabled = true;
        mw.cw_grid_height.IsEnabled = true;
        editingMode = true;
        generating = false;
    }
}
else
{
    MessageBox.Show("Слова не найдены");
}
}
}
public void Generate_slow(int w, int h, List<CwCell> grid)
```

```
{
    // если генерация уже не запущена
    if (!generating)
    {
        // если есть загруженные слова
        if (words_num > 0)
        {
            mw.CW_info_label.Content = "Генерация";
            mw.CW_info_label.UpdateLayout();
            // выключаем кнопку "генерировать"
            mw.generate_button.IsEnabled = false;
            mw.cw_grid_width.IsEnabled = false;
            mw.cw_grid_height.IsEnabled = false;
            editingMode = false;
            generating = true;
            grid_words_count = 0;
            words_grid.Clear();
            // поиск вертикальных слов
            // цикл по X
            for (int i = 0; i < w; i++)
            {
                // координаты первый точки
                int x0 = 0;
                int y0 = 0;
                // длина слова
                int ls = 0;
                // цикл по Y
                for (int j = 0; j < h; j++)
                {
```

```
                int xy = mw.fromXYtoSingle(i, j);
                if (grid[xy].selected)
                {
                    if (ls == 0)
                    {
                        x0 = i;
                        y0 = j;
                    }
                    ls++;
                    // если последняя итерация
                    if (j == h - 1)
                    {
                        words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y =
y0, orientation = 1, answer = "", question = "", intersects = new List<int>(), isdefined = false,
word_number = 0, word_inlist_id = 0 });
                        grid_words_count++;
                        ls = 0;
                    }
                }
                else
                {
                    if (ls > 1)
                    {
                        words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y =
y0, orientation = 1, answer = "", question = "", intersects = new List<int>(), isdefined = false,
word_number = 0, word_inlist_id = 0 });
                        grid_words_count++;
                    }
                    ls = 0;
                }
```

```
            }
        }
        // поиск горизонтальных слов
        // цикл по Y
        for (int i = 0; i < h; i++)
        {
            // координаты первый точки
            int x0 = 0;
            int y0 = 0;
            // длина слова
            int ls = 0;
            // цикл по X
            for (int j = 0; j < w; j++)
            {
                int xy = mw.fromXYtoSingle(j, i);
                if (grid[xy].selected)
                {
                    if (ls == 0)
                    {
                        x0 = j;
                        y0 = i;
                    }
                    ls++;
                    // если последняя итерация
                    if (j == w - 1)
                    {
                        words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y =
y0, orientation = 0, answer = "", question = "", intersects = new List<int>(), isdefined = false,
word_number = 0, word_inlist_id = 0 });
```

```
                    grid_words_count++;
                    ls = 0;
                }
            }
            else
            {
                if (ls > 1)
                {
                    words_grid.Add(grid_words_count, new CW_word_item { length = ls, x = x0, y = y0, orientation = 0, answer = "", question = "", intersects = new List<int>(), isdefined = false, word_number = 0, word_inlist_id = 0 });
                    grid_words_count++;
                }
                ls = 0;
            }
        }
    }
    // если есть шаблоны слов на сетке
    if (grid_words_count > 0)
    {
        // определение пересечений
        intersections();
        // сортировка сетки слов по сложности, длине
        var items = from pair in words_grid
                    orderby pair.Value.complexity descending, pair.Value.length
                    select pair;
        words_grid = items.ToDictionary(x => x.Key, x => x.Value);
        int word_gid = 0;
        int word_glen = grid_words_count;
```

```
bool end_gen = false;
bool process = true;
while(word_gid < word_glen && end_gen == false && process == true)
{
   CW_word_item word = words_grid[word_gid];
   int len = word.length;
   // если в списке есть слова данной длины
   if (items_list.ContainsKey(len))
   {
      int word_lid = word.word_inlist_id;
      int word_llen = items_list[len].Count;
      bool st2 = false;
      while (word_lid < word_llen && st2 == false)
      {
         CW_word_list_item item = items_list[len][word_lid];
         // если слово еще не использовалось
         if (!item.isUsed)
         {
            string pattern = getPattern(word);
            string text = item.answer;
            string qst_text = item.question;
            // если строка совпадает с регуляркой
            if (Regex.IsMatch(text, pattern, RegexOptions.IgnoreCase))
            {
               word.updateWord(text, qst_text, intersections_list);
               item.isUsed = true;
               word.word_inlist_id = word_lid;
               st2 = true;
               word_gid++;
```

```
                count_seted_words++;
                // если это было последнее слово на сетке
                if (word_gid == word_glen)
                {
                    end_gen = true;
                }
                else
                {
                    words_grid[word_gid].word_inlist_id = 0;
                }
            }
            else
            {
                // перейти к след. слову по списку
                word_lid++;
            }
        }
        else
        {
            // перейти к след. слову по списку
            word_lid++;
        }
    }
    // если слово не удалось установить
    if(st2 != true)
    {
        // если слово не удалось установить
        // если это первое слово в сетке
        if (word_gid == 0)
```

```
            {
                MessageBox.Show("Не удалось сгенерировать кроссворд!!!");
                // включаем кнопку "генерировать"
                mw.generate_button.IsEnabled = true;
                mw.cw_grid_width.IsEnabled = true;
                mw.cw_grid_height.IsEnabled = true;
                editingMode = true;
                generating = false;
                process = false;
            }
            else
            {
                // перейти к пред. слову в сетке
                word_gid--;
                count_seted_words--;
                CW_word_item word_prev = words_grid[word_gid];
                items_list[word_prev.length][word_prev.word_inlist_id].isUsed = false;
                word_prev.resetWord(intersections_list);
                word_prev.word_inlist_id++;
            }
        }
    }
    else
    {
        MessageBox.Show("Не удалось сгенерировать кроссворд!!");
        // включаем кнопку "генерировать"
        mw.generate_button.IsEnabled = true;
        mw.cw_grid_width.IsEnabled = true;
        mw.cw_grid_height.IsEnabled = true;
```

```
                editingMode = true;
                generating = false;
                process = false;
            }
        }
        // если полностью сгенерировали
        if (end_gen == true) {
            setNumerating();
            mw.draw_words(ref mw.cw_field, true);
            generating = false;
        }
        else
        {
            MessageBox.Show("Не удалось сгенерировать кроссворд!");
            // включаем кнопку "генерировать"
            mw.generate_button.IsEnabled = true;
            mw.cw_grid_width.IsEnabled = true;
            mw.cw_grid_height.IsEnabled = true;
            editingMode = true;
            generating = false;
            process = false;
        }
    }
    else
    {
        MessageBox.Show("Необходимо заполнить сетку");
        // включаем кнопку "генерировать"
        mw.generate_button.IsEnabled = true;
        mw.cw_grid_width.IsEnabled = true;
```

```
                mw.cw_grid_height.IsEnabled = true;
                editingMode = true;
                generating = false;
            }
        }
        else
        {
            MessageBox.Show("Слова не найдены");
        }
    }
}
// определение нуемрации слов
private void setNumerating()
{
    int idd = 1;
    foreach (var word_data in words_grid)
    {
        CW_word_item word = word_data.Value;
        int key = word_data.Key;

        // если не определен номер слова
        if (word.word_number == 0)
        {
            // если есть пересечения
            if (word.complexity > 0)
            {
                // цикл по пересечениям слова
                for (int j = 0; j < word.complexity; j++)
                {
```

```
                CW_words_intersections its = intersections_list[word.intersects[j]];
                // id второго слова в пересечении
                int wid = its.words_ids[0];
                if (its.words_ids[0] == key)
                {
                   wid = its.words_ids[1];
                }
                // если пересечение находится в первой клетке слова (в которой указывается
номер слова)
                if (its.x == word.x && its.y == word.y && its.x == words_grid[wid].x && its.y ==
words_grid[wid].y)
                {
                   // если у второго слова в пересеченииии не определен номер
                   if (words_grid[wid].word_number == 0)
                   {
                      // устанавливает новый номер обоим словам
                      word.word_number = idd;
                      words_grid[wid].word_number = idd;
                      // увеличить номер слова
                      idd++;
                      break;
                   }
                   else
                   {
                      word.word_number = words_grid[wid].word_number;
                      // увеличить номер слова
                      idd++;
                   }
                }
```

```
                }
                if (word.word_number == 0)
                {
                    word.word_number = idd;
                    idd++;
                }
            }
            else
            {
                word.word_number = idd;
                // увеличить номер слова
                idd++;
            }
        }
    }
}
// определение пересечений
private void intersections()
{
    foreach (var wordX_data in words_grid)
    {
        CW_word_item wordX = wordX_data.Value;
        int keyX = wordX_data.Key;
        foreach (var wordY_data in words_grid)
        {
            CW_word_item wordY = wordY_data.Value;
            int keyY = wordY_data.Key;
            if (wordX.orientation == 0 && wordY.orientation == 1)
            {
```

```
                if (wordX.x <= wordY.x && wordX.x + wordX.length-1 >= wordY.x)
                {
                    if (wordX.y >= wordY.y && wordY.y + wordY.length-1 >= wordX.y)
                    {
                        intersections_list.Add(new CW_words_intersections(wordY.x, wordX.y, ' ', new int[]
{ keyX, keyY }));
                        int intr_id = intersections_list.Count()-1; // id Добавленного пересечения
                        wordX.intersects.Add(intr_id);
                        wordY.intersects.Add(intr_id);
                        wordX.complexity++; // увеличить сложность слова
                        wordY.complexity++; // увеличить сложность слова
                    }
                }
            }
        }
    }
    // сохранить макет
    public void saveMaket()
    {
        CW_Maket mk = new CW_Maket { grid_wrds = words_grid, cells = mw.cw_cells, cw = mw.w,
ch = mw.h, csw = count_seted_words, gwc = grid_words_count, edm = editingMode };
        string json = JsonSerializer.Serialize(mk);
        SaveFileDialog saveJsonDialog = new SaveFileDialog();
        saveJsonDialog.Filter = "json files (*.json)|*.json";
        if (saveJsonDialog.ShowDialog() == true)
        {
            File.WriteAllText(saveJsonDialog.FileName, json);
            if(maketSaveFilePath == null)
```

```
            {
                DispatcherTimer autoSaveTimer = new DispatcherTimer();
                autoSaveTimer.Interval = new TimeSpan(0, 0, 30); //in Hour, Minutes, Second.
                autoSaveTimer.Tick += autoSaveMaket;
                autoSaveTimer.Start();
            }
            maketSaveFilePath = saveJsonDialog.FileName;
        }
    }
    // автосохранение макета
    public void autoSaveMaket(object sender, EventArgs e)
    {
        if(maketSaveFilePath != null)
        {
            if (hasChanges == true)
            {
                CW_Maket mk = new CW_Maket { grid_wrds = words_grid, cells = mw.cw_cells, cw =
mw.w, ch = mw.h, csw = count_seted_words, gwc = grid_words_count, edm = editingMode };
                string json = JsonSerializer.Serialize(mk);
                File.WriteAllText(maketSaveFilePath, json);
                hasChanges = false;
            }
        }
    }
    // открыть сетку
    public void openMaket()
    {
        OpenFileDialog openJsonDialog = new OpenFileDialog();
        if (openJsonDialog.ShowDialog() == true)
```

```
{
    try
    {
        string jsonString = File.ReadAllText(openJsonDialog.FileName);
        CW_Maket mk = JsonSerializer.Deserialize<CW_Maket>(jsonString);
        words_grid = mk.grid_wrds;
        mw.cw_cells = mk.cells;
        mw.w = mk.cw;
        mw.h = mk.ch;
        count_seted_words = mk.csw;
        grid_words_count = mk.gwc;
        editingMode = mk.edm;
        mw.draw_words(ref mw.cw_field, true);
        // перерисовка
        mw.Cw_Draw(ref mw.cw_field);
        mw.pushQuestions2List();
        if (maketSaveFilePath == null)
        {
            DispatcherTimer autoSaveTimer = new DispatcherTimer();
            autoSaveTimer.Interval = new TimeSpan(0, 0, 30); //in Hour, Minutes, Second.
            autoSaveTimer.Tick += autoSaveMaket;
            autoSaveTimer.Start();
        }
        maketSaveFilePath = openJsonDialog.FileName;
    }
        catch
    {
        MessageBox.Show("Ошибка файла");
    }
```

```
        }
    }
}
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public Person(string n, int a)
    {
        Name = n;
        Age = a;
    }
}
class Person_two
{
    public string Name { get; set; }
    public int Age { get; set; }
}
public class CW_word_list_item
{
    public int length; // длина слова
    public string answer; // само слово
    public string question; // вопрос
    public bool isUsed; // ипользовано слово?
    public CW_word_list_item(int l, string quest, string ans)
    {
        length = l;
        question = quest;
        answer = ans;
```

```
            isUsed = false;
        }
    }
    public class CW_word_item
    {
        public int length { get; set; } // длина слова
        public string answer { get; set; } // само слово
        public string question { get; set; } // вопрос для слова
        public int x { get; set; }
        public int y { get; set; }
        public int orientation { get; set; } // ориенатция (0 - горизионталь, 1 - вертикаль)
        public int complexity { get; set; } // сложность слова
        public List<int> intersects { get; set; }  //id пересечений
        public bool isdefined { get; set; }  // установлено ли слово
        public int word_number { get; set; } // номер слова в сетке
        public int word_inlist_id { get; set; }
        public char getCharByXY(int x0, int y0)
        {
            int dl;
            // если слово горизонтальное
            if(orientation == 0)
            {
                dl = x0 - x;
            }
            else
            {
                dl = y0 - y;
            }
            return answer[dl];
```

```
    }
    public void updateWord(string answ, string qst, List<CW_words_intersections> ilist)
    {
        answer = answ;
        question = qst;
        isdefined = true;
        if (intersects.Count > 0)
        {
            foreach(int id in intersects)
            {
                ilist[id].ch = getCharByXY(ilist[id].x, ilist[id].y);
            }
        }
    }
    public void resetWord(List<CW_words_intersections> ilist)
    {
        answer = "";
        question = "";
        isdefined = false;
        if (intersects.Count > 0)
        {
            foreach (int id in intersects)
            {
                ilist[id].ch = ' ';
            }
        }
    }
}
// пересечения слов
```

```
public class CW_words_intersections
{
    public int x; // координата X пересечения
    public int y; // координата Y пересечения
    public char ch; // буква на пересечении (может быть пустая строка)
    public int[] words_ids; // id слов
    public CW_words_intersections(int ox, int oy, char c, int[] wids)
    {
        x = ox;
        y = oy;
        ch = c;
        words_ids = wids;
    }
}

public class CW_Maket
{
    public List<CwCell> cells { get; set; }
    public Dictionary<int, CW_word_item> grid_wrds { get; set; }
    public int cw { get; set; }
    public int ch { get; set; }
    public int csw { get; set; }
    public int gwc { get; set; }
    public bool edm { get; set; }
}
}
```

MainWindow.xaml.cs

```
using System.Windows;
using System.Collections.Generic;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Shapes;
using System;
using System.Windows.Controls;
using System.Threading.Tasks;
using System.Windows.Media.Imaging;
namespace Crossword
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public List<CwCell> cw_cells = null;
        private List<CwCell> cw_cells_tmp = null;
        // размер клетки
        private int block_size = 24;
        public int w;
        public int h;
        // отступы на канве
        public int canvas_paddingT = 5;
        public int canvas_paddingL = 5;
        public struct PointXY
        {
```

```
    public int x;
    public int y;
}
private PointXY selectedCell;
private CW_Content content = null;
public bool fileopening = false;
public MainWindow()
{
    InitializeComponent();
    content = new CW_Content(this);
}
public int fromXYtoSingle(int x, int y)
{
    return x * h + y;
}
public int[] fromSingleToXY(int c)
{
    return new int[2] { c / h, c % h };
}
// закрыть
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
    System.Windows.Application.Current.Shutdown();
}
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    selectedCell.x = -1;
    selectedCell.y = -1;
    LoadCells();
```

```
    Cw_Draw(ref cw_field);
}
private void LoadCells()
{
    if (cw_grid_width == null || cw_grid_height == null)
    {
        return;
    }
    w = (int)cw_grid_width.Value;
    h = (int)cw_grid_height.Value;
    cw_cells_tmp = cw_cells;
    cw_cells = new List<CwCell>();
    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {
            int[] oxy = new int[2] { -1, -1 };
            if (cw_cells_tmp != null)
            {
                oxy = fromSingleToXY(cw_cells.Count - 1);
            }

            if (cw_cells_tmp != null && oxy[0] >= i && oxy[1] >= j)
            {
                cw_cells = cw_cells_tmp;
            }
            else
            {
```

```
                cw_cells.Add(new CwCell { x = i * block_size, y = j * block_size, grid_x = i, grid_y = j, selected = false });
            }
        }
    }
    public void Cw_Draw(ref Canvas cvs, bool place_words = true)
    {
        // если канва не определена
        if (cw_cells == null || cvs == null)
        {
            return;
        }
        // очистка
        cvs.Children.Clear();
        cvs.Width = block_size * w;
        cvs.Height = block_size * h;
        // фон
        cvs.Background = Brushes.White;
        for (int i = 0; i < w; i++)
        {
            for (int j = 0; j < h; j++)
            {
                int xy = fromXYtoSingle(i, j);
                if (cw_cells[xy].selected) {
                    Point cellPoint1 = new Point(canvas_paddingL + cw_cells[xy].x, canvas_paddingT + cw_cells[xy].y);
                    Point cellPoint2 = new Point(canvas_paddingL + cw_cells[xy].x + block_size, canvas_paddingT + cw_cells[xy].y);
```

```
                Point cellPoint3 = new Point(canvas_paddingL + cw_cells[xy].x + block_size,
canvas_paddingT + cw_cells[xy].y + block_size);

                Point cellPoint4 = new Point(canvas_paddingL + cw_cells[xy].x, canvas_paddingT +
cw_cells[xy].y + block_size);

                PointCollection cellPointCollection = new PointCollection();

                cellPointCollection.Add(cellPoint1);

                cellPointCollection.Add(cellPoint2);

                cellPointCollection.Add(cellPoint3);

                cellPointCollection.Add(cellPoint4);

                Polygon cellEl = new Polygon

                {

                   Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0)),

                   Fill = Brushes.Gray,

                   StrokeThickness = 2,

                   Points = cellPointCollection,

                };

                cvs.Children.Add(cellEl);

            }

            //вертикаль

            Line vline = new Line

            {

               X1 = canvas_paddingL + cw_cells[xy].x,

               X2 = canvas_paddingL +  cw_cells[xy].x,

               Y1 = canvas_paddingT + cw_cells[xy].y,

               Y2 = canvas_paddingT + cw_cells[xy].y + block_size,

               StrokeThickness = 2

            };

            //System.Diagnostics.Debug.WriteLine(cw_cells.Length);

            vline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
```

```
cvs.Children.Add(vline);
//горизонталь
Line gline = new Line
{
   X1 = canvas_paddingL +  cw_cells[xy].x,
   X2 = canvas_paddingL +  cw_cells[xy].x + block_size,
   Y1 = canvas_paddingT + cw_cells[xy].y,
   Y2 = canvas_paddingT + cw_cells[xy].y,
   StrokeThickness = 2
};
gline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
cvs.Children.Add(gline);
// отрисовать крайнюю правую границу у блока
if (i == w - 1)
{
   Line vLastline = new Line
   {
      X1 = canvas_paddingL + cw_cells[xy].x + block_size,
      X2 = canvas_paddingL + cw_cells[xy].x + block_size,
      Y1 = canvas_paddingT + cw_cells[xy].y,
      Y2 = canvas_paddingT + cw_cells[xy].y + block_size,
      StrokeThickness = 2
   };
   vLastline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
   cvs.Children.Add(vLastline);
}
// отрисовать крайнюю нижнюю границу у блока
if (j == h - 1)
{
```

```
                Line vLastline = new Line
                {
                   X1 = canvas_paddingL +  cw_cells[xy].x,
                   X2 = canvas_paddingL +  cw_cells[xy].x + block_size,
                   Y1 = canvas_paddingT + cw_cells[xy].y + block_size,
                   Y2 = canvas_paddingT + cw_cells[xy].y + block_size,
                   StrokeThickness = 2
                };
                vLastline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
                cvs.Children.Add(vLastline);
            }
        }
    }
    draw_words(ref cvs, place_words);
    content.hasChanges = true;
}
public void draw_words(ref Canvas cvs, bool place_word)
{
    // если на поле есть установленные слова
    if (content.count_seted_words > 0)
    {
        // цикл по каждому слову в сетке
        foreach (var word_data in content.words_grid)
        {
            CW_word_item word = word_data.Value;
                // если слово установлено
            if (word.isdefined)
            {
                int word_len = word.length;
```

```
int dx = 0;
int dy = 0;
if (word.orientation == 0)
{
   dx = 1;
}
else
{
   dy = 1;
}
if (place_word == true)
{
   // установка букв
   for (int i = 0; i < word_len; i++)
   {
      int x_pos = (word.x + dx * i) * block_size;
      int y_pos = (word.y + dy * i) * block_size;
      Label charBlock = new Label();
      charBlock.FontSize = 18;
      charBlock.Width = block_size;
      charBlock.Height = block_size;
      charBlock.Foreground = new SolidColorBrush(Color.FromArgb(255, 255, 255, 0));
      charBlock.Content = word.answer[i].ToString();
      charBlock.VerticalContentAlignment = VerticalAlignment.Center;
      charBlock.HorizontalContentAlignment = HorizontalAlignment.Center;
      charBlock.Padding = new Thickness(0);
      Canvas.SetLeft(charBlock, canvas_paddingL + x_pos);
      Canvas.SetTop(charBlock, canvas_paddingT + y_pos);
      cvs.Children.Add(charBlock);
```

```
                }
            }
            // установка номера слова
            TextBlock word_num = new TextBlock();
            word_num.Background = Brushes.White;
            word_num.TextAlignment = TextAlignment.Center;
            word_num.Text = word.word_number.ToString();
            Canvas.SetLeft(word_num, canvas_paddingL +  word.x * block_size);
            Canvas.SetTop(word_num, canvas_paddingT + word.y * block_size);
            cvs.Children.Add(word_num);
        }
      }
   }
}
public void Cw_Draw_preview(ref Canvas cvs, bool place_words = true)
{
   // если канва не определена
   if (cw_cells == null || cvs == null)
   {
      return;
   }
   // очистка
   cvs.Children.Clear();
   cvs.Width = block_size * w;
   cvs.Height = block_size * h;
   // фон
   cvs.Background = Brushes.White;
   for (int i = 0; i < w; i++)
   {
```

```
            for (int j = 0; j < h; j++)
            {
                int xy = fromXYtoSingle(i, j);
                if (cw_cells[xy].selected)
                {
                    Point cellPoint1 = new Point(canvas_paddingL + cw_cells[xy].x, canvas_paddingT +
cw_cells[xy].y);
                    Point cellPoint2 = new Point(canvas_paddingL + cw_cells[xy].x + block_size,
canvas_paddingT + cw_cells[xy].y);
                    Point cellPoint3 = new Point(canvas_paddingL + cw_cells[xy].x + block_size,
canvas_paddingT + cw_cells[xy].y + block_size);
                    Point cellPoint4 = new Point(canvas_paddingL + cw_cells[xy].x, canvas_paddingT +
cw_cells[xy].y + block_size);
                    PointCollection cellPointCollection = new PointCollection();
                    cellPointCollection.Add(cellPoint1);
                    cellPointCollection.Add(cellPoint2);
                    cellPointCollection.Add(cellPoint3);
                    cellPointCollection.Add(cellPoint4);
                    Polygon cellEl = new Polygon
                    {
                        Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0)),
                        Fill = Brushes.White,
                        StrokeThickness = 2,
                        Points = cellPointCollection,
                    };
                    cvs.Children.Add(cellEl);
                }
                else
                {
                    Point cellPoint1 = new Point(canvas_paddingL + cw_cells[xy].x, canvas_paddingT +
cw_cells[xy].y);
```

```
                Point cellPoint2 = new Point(canvas_paddingL + cw_cells[xy].x + block_size,
canvas_paddingT + cw_cells[xy].y);

                Point cellPoint3 = new Point(canvas_paddingL + cw_cells[xy].x + block_size,
canvas_paddingT + cw_cells[xy].y + block_size);

                Point cellPoint4 = new Point(canvas_paddingL + cw_cells[xy].x, canvas_paddingT +
cw_cells[xy].y + block_size);

                PointCollection cellPointCollection = new PointCollection();

                cellPointCollection.Add(cellPoint1);

                cellPointCollection.Add(cellPoint2);

                cellPointCollection.Add(cellPoint3);

                cellPointCollection.Add(cellPoint4);

                Polygon cellEl = new Polygon

                {

                   Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0)),

                   Fill = Brushes.Black,

                   StrokeThickness = 2,

                   Points = cellPointCollection,

                };

                cvs.Children.Add(cellEl);

            }

            //вертикаль

            Line vline = new Line

            {

               X1 = canvas_paddingL + cw_cells[xy].x,

               X2 = canvas_paddingL + cw_cells[xy].x,

               Y1 = canvas_paddingT + cw_cells[xy].y,

               Y2 = canvas_paddingT + cw_cells[xy].y + block_size,

               StrokeThickness = 2

            };

            vline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
```

```
cvs.Children.Add(vline);
//горизонталь
Line gline = new Line
{
   X1 = canvas_paddingL + cw_cells[xy].x,
   X2 = canvas_paddingL + cw_cells[xy].x + block_size,
   Y1 = canvas_paddingT + cw_cells[xy].y,
   Y2 = canvas_paddingT + cw_cells[xy].y,
   StrokeThickness = 2
};
gline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
cvs.Children.Add(gline);
// отрисовать крайнюю правую границу у блока
if (i == w - 1)
{
   Line vLastline = new Line
   {
      X1 = canvas_paddingL + cw_cells[xy].x + block_size,
      X2 = canvas_paddingL + cw_cells[xy].x + block_size,
      Y1 = canvas_paddingT + cw_cells[xy].y,
      Y2 = canvas_paddingT + cw_cells[xy].y + block_size,
      StrokeThickness = 2
   };
   vLastline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
   cvs.Children.Add(vLastline);
}
// отрисовать крайнюю нижнюю границу у блока
if (j == h - 1)
{
```

```
                Line vLastline = new Line
                {
                    X1 = canvas_paddingL + cw_cells[xy].x,
                    X2 = canvas_paddingL + cw_cells[xy].x + block_size,
                    Y1 = canvas_paddingT + cw_cells[xy].y + block_size,
                    Y2 = canvas_paddingT + cw_cells[xy].y + block_size,
                    StrokeThickness = 2
                };
                vLastline.Stroke = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
                cvs.Children.Add(vLastline);
            }
        }
    }
    draw_words_preview(ref cvs, place_words);

}
public void draw_words_preview(ref Canvas cvs, bool place_word)
{
    // если на поле есть установленные слова
    if (content.count_seted_words > 0)
    {
        // цикл по каждому слову в сетке
        foreach (var word_data in content.words_grid)
        {
            CW_word_item word = word_data.Value;
            // если слово установлено
            if (word.isdefined)
            {
                int word_len = word.length;
```

```
int dx = 0;
int dy = 0;
if (word.orientation == 0)
{
   dx = 1;
}
else
{
   dy = 1;
}
if (place_word == true)
{
   // установка букв
   for (int i = 0; i < word_len; i++)
   {
      int x_pos = (word.x + dx * i) * block_size;
      int y_pos = (word.y + dy * i) * block_size;
      Label charBlock = new Label();
      charBlock.FontSize = 18;
      charBlock.Width = block_size;
      charBlock.Height = block_size;
      charBlock.Foreground = new SolidColorBrush(Color.FromArgb(255, 0, 0, 0));
      charBlock.Content = word.answer[i].ToString();
      charBlock.VerticalContentAlignment = VerticalAlignment.Center;
      charBlock.HorizontalContentAlignment = HorizontalAlignment.Center;
      charBlock.Padding = new Thickness(0);
      Canvas.SetLeft(charBlock, canvas_paddingL + x_pos);
      Canvas.SetTop(charBlock, canvas_paddingT + y_pos);
      cvs.Children.Add(charBlock);
```

```
                }
            }
            // установка номера слова
            TextBlock word_num = new TextBlock();
            word_num.Background = Brushes.Transparent;
            word_num.TextAlignment = TextAlignment.Center;
            word_num.Text = word.word_number.ToString();
            Canvas.SetLeft(word_num, canvas_paddingL + word.x * block_size);
            Canvas.SetTop(word_num, canvas_paddingT + word.y * block_size);
            cvs.Children.Add(word_num);
        }
    }
  }
}
public void pushQuestions2List()
{
  // если на поле есть установленные слова
  if (content.count_seted_words > 0)
  {
    bool setG = false;
    bool setV = false;
    // цикл по каждому горизонтальному слову в сетке
    foreach (var word_data in content.words_grid)
    {
      CW_word_item word = word_data.Value;
      if (word.isdefined)
      {
        if (word.orientation == 0)
        {
```

```
            // если заголовок не установлен
            if (setG == false)
            {
               TextBlock g_answer_block = new TextBlock();
               g_answer_block.Text = "По горизонтали:";
               g_answer_block.FontWeight = FontWeights.Bold;
               CW_answers_list.Items.Add(g_answer_block);
               setG = true;
            }
            // если слово установлено
            if (word.isdefined)
            {
               TextBlock answer_block = new TextBlock();
               answer_block.TextWrapping = TextWrapping.Wrap;
               answer_block.Width = 235;
               answer_block.Text = word.word_number + " - " + word.question;
               CW_answers_list.Items.Add(answer_block);
            }
         }
      }
   }
   // цикл по каждому вертикальному слову в сетке
   foreach (var word_data in content.words_grid)
   {
      CW_word_item word = word_data.Value;
      if (word.isdefined)
      {
         if (word.orientation == 1)
         {
```

```
                // если заголовок не установлен
                if (setV == false)
                {
                    TextBlock v_answer_block = new TextBlock();
                    v_answer_block.Text = "По вертикали:";
                    v_answer_block.FontWeight = FontWeights.Bold;
                    CW_answers_list.Items.Add(v_answer_block);
                    setV = true;
                }
                // если слово установлено
                if (word.isdefined)
                {
                    TextBlock answer_block = new TextBlock();
                    answer_block.Text = word.word_number + " - " + word.question;
                    answer_block.Width = 235;
                    answer_block.TextWrapping = TextWrapping.Wrap;
                    CW_answers_list.Items.Add(answer_block);
                }
            }
        }
    }
}
public void clear_cells()
{
    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {
```

```
            int xy = fromXYtoSingle(i, j);
            cw_cells[xy].selected = false;
        }
    }
}
private void Window_SizeChanged(object sender, SizeChangedEventArgs e)
{
    // перерисовка при изменении размера формы
    Cw_Draw(ref cw_field);
}
// клик по полю
private void Cw_field_MouseUp(object sender, MouseButtonEventArgs e)
{
    if (content.editingMode == true)
    {
        double mouseX = Math.Round(e.GetPosition(cw_field).X - canvas_paddingL);
        double mouseY = Math.Round(e.GetPosition(cw_field).Y - canvas_paddingT);
        if (mouseX >= 0 && mouseY >= 0)
        {
            selectedCell.x = (int)mouseX / block_size;
            selectedCell.y = (int)mouseY / block_size;
            // toggle bool
            try
            {
                int xy = fromXYtoSingle(selectedCell.x, selectedCell.y);
                cw_cells[xy].selected = !cw_cells[xy].selected;
            }
            catch (Exception)
            {
```

```
                }
                // перерисовка
                Cw_Draw(ref cw_field);
            }
        }
        else
        {
            MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Перейти в режим редактирования?", "Ok", System.Windows.MessageBoxButton.YesNo);
            if (messageBoxResult == MessageBoxResult.Yes)
            {
                content.editingMode = true;
                generate_button.IsEnabled = true;
                cw_grid_width.IsEnabled = true;
                cw_grid_height.IsEnabled = true;
                content.grid_words_list_clear();
                Cw_Draw(ref cw_field);
            }
        }
    }
    private void Cw_grid_width_ValueChanged(object sender, RoutedPropertyChangedEventArgs<object> e)
    {
        if(cw_grid_width.Value != null) {
            if (content != null)
            {
                if (content.editingMode == false)
                {
                    MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Перейти в режим редактирования?", "Ok", System.Windows.MessageBoxButton.YesNo);
```

```
                if (messageBoxResult == MessageBoxResult.Yes)
                {
                    content.editingMode = true;
                    generate_button.IsEnabled = true;
                    cw_grid_width.IsEnabled = true;
                    cw_grid_height.IsEnabled = true;
                    LoadCells();
                    Cw_Draw(ref cw_field);
                }
            }
            else
            {
                LoadCells();
                Cw_Draw(ref cw_field);
            }
        }
    }

    private void Cw_grid_height_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<object> e)
    {
        if (cw_grid_width.Value != null)
        {
            if (content != null)
            {
                if (content.editingMode == false)
                {
```

```
                MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Перейти в режим редактирования?", "Ok", System.Windows.MessageBoxButton.YesNo);
                if (messageBoxResult == MessageBoxResult.Yes)
                {
                    content.editingMode = true;
                    generate_button.IsEnabled = true;
                    cw_grid_width.IsEnabled = true;
                    cw_grid_height.IsEnabled = true;
                    LoadCells();
                    Cw_Draw(ref cw_field);
                }
            }
            else
            {
                LoadCells();
                Cw_Draw(ref cw_field);
            }
        }
    }
}
public void onFileOpened(int wn)
{
    fileopening = false;
    cw_words_number.Content = wn;
    CW_info_label.Content = "";
    addWord_menu_item.IsEnabled = true;
    openDictBut.IsEnabled = false;
}
// открыть
```

```
public void MenuItem_Click_1(object sender, RoutedEventArgs e)
{
    Task task = new Task(content.Open);
    task.Start();
}
// генерировать
private void Button_Click(object sender, RoutedEventArgs e)
{
    if (fileopening == false)
    {
        content.grid_words_list_clear();
        if (fast_generation.IsChecked == true)
        {
            content.Generate(w, h, cw_cells);
        }
        else
        {
            content.Generate_slow(w, h, cw_cells);
        }
        pushQuestions2List();
        CW_info_label.Content = "";
        generate_button.IsEnabled = true;
        Cw_Draw(ref cw_field);
    }
    else
    {
        MessageBox.Show("Дождитесь окончания открытия файла");
    }
}
```

```
// очистить
private void Cw_clear_button_Click(object sender, RoutedEventArgs e)
{
    if(content.editingMode == false)
    {
        MessageBoxResult messageBoxResult = System.Windows.MessageBox.Show("Перейти в режим редактирования?", "Ok", System.Windows.MessageBoxButton.YesNo);
        if (messageBoxResult == MessageBoxResult.Yes)
        {
            content.editingMode = true;
            generate_button.IsEnabled = true;
            cw_grid_width.IsEnabled = true;
            cw_grid_height.IsEnabled = true;
            content.grid_words_list_clear();
            clear_cells();
            Cw_Draw(ref cw_field);
        }
    }
    else
    {
        content.grid_words_list_clear();
        clear_cells();
        Cw_Draw(ref cw_field);
    }
}
// справка
private void MenuItem_Click_3(object sender, RoutedEventArgs e)
{
```

```
            string helpText = "CrossWord - генератор кроссвордов. " + Environment.NewLine +
"Разработчик: Крючков И. С.";

            MessageBox.Show(helpText, "Справка");

        }

        // сохранить

        private void MenuItem_Click_4(object sender, RoutedEventArgs e)

        {

            SaveToFileWindow StF = new SaveToFileWindow(this, content);

            StF.Owner = this;

            StF.ShowDialog();

        }

        // сохранить сетку

        private void MenuItem_Click_2(object sender, RoutedEventArgs e)

        {

            content.saveMaket();

        }

        // открыть сетку

        private void MenuItem_Click_5(object sender, RoutedEventArgs e)

        {

            content.openMaket();

        }

        // Добавить слово

        private void MenuItem_Click_6(object sender, RoutedEventArgs e)

        {

            if (content.dictOpened == true)

            {

                addWordWindow AddWord = new addWordWindow(this, content);

                AddWord.Owner = this;

                AddWord.ShowDialog();
```

```
            }
            else
            {
                MessageBox.Show("Необходимо открыть словарь");
            }
        }
    }
    public class CwCell
    {
        // координаты на канве
        public int x { get; set; }
        public int y { get; set; }
        // координаты клеток
        public int grid_x { get; set; }
        public int grid_y { get; set; }
        public bool selected { get; set; }
    }
}
```

SaveToFileWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using Microsoft.Win32;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
namespace Crossword
{
    /// <summary>
    /// Логика взаимодействия для Window1.xaml
    /// </summary>
    public partial class SaveToFileWindow : Window
    {
        MainWindow mw = null;
        CW_Content content = null;
        public SaveToFileWindow(MainWindow mnw, CW_Content cnt)
        {
            InitializeComponent();
            mw = mnw;
```

```
        content = cnt;
    }
    // Сохранить
    private void Export_button_Click(object sender, RoutedEventArgs e)
    {
        preview_scroll.ScrollToTop();
        preview_scroll.UpdateLayout();
        RenderTargetBitmap rtb = new RenderTargetBitmap((int)preview_canvas.ActualWidth,
        (int)preview_canvas.ActualHeight, 96d, 96d, System.Windows.Media.PixelFormats.Default);
        rtb.Render(preview_canvas);
        BitmapEncoder pngEncoder = new PngBitmapEncoder();
        pngEncoder.Frames.Add(BitmapFrame.Create(rtb));
        SaveFileDialog saveFileDialog = new SaveFileDialog();
        saveFileDialog.Filter = "PNG (*.png)|*.png";
        saveFileDialog.DefaultExt = "png";
        if (saveFileDialog.ShowDialog() == true)
        {
            using (var fs = System.IO.File.OpenWrite(saveFileDialog.FileName))
            {
                pngEncoder.Save(fs);
            }
        }
        this.Hide();
    }
    // добавление на канву списка вопросов
    public void drawQuestions(CW_Content content)
    {
        ListBox previewlistBox = new ListBox();
        // если на поле есть установленные слова
```

```
if (content.count_seted_words > 0)
{
   bool setG = false;
   bool setV = false;
   // цикл по каждому горизонтальному слову в сетке
   foreach (var word_data in content.words_grid)
   {
      CW_word_item word = word_data.Value;
      if (word.isdefined)
      {
         if (word.orientation == 0)
         {
            // если заголовок не установлен
            if (setG == false)
            {
               TextBlock g_answer_block = new TextBlock();
               g_answer_block.Text = "По горизонтали:";
               g_answer_block.FontWeight = FontWeights.Bold;
               previewlistBox.Items.Add(g_answer_block);
               setG = true;
            }
            // если слово установлено
            if (word.isdefined)
            {
               TextBlock answer_block = new TextBlock();
               answer_block.TextWrapping = TextWrapping.Wrap;
               answer_block.Width = preview_canvas.Width;
               answer_block.Text = word.word_number + " - " + word.question;
               previewlistBox.Items.Add(answer_block);
```

```
            }
        }
    }
}
// цикл по каждому вертикальному слову в сетке
foreach (var word_data in content.words_grid)
{
    CW_word_item word = word_data.Value;
    if (word.isdefined)
    {
        if (word.orientation == 1)
        {
            // если заголовок не установлен
            if (setV == false)
            {
                TextBlock v_answer_block = new TextBlock();
                v_answer_block.Text = "По вертикали:";
                v_answer_block.FontWeight = FontWeights.Bold;
                previewlistBox.Items.Add(v_answer_block);
                setV = true;
            }
            // если слово установлено
            if (word.isdefined)
            {
                TextBlock answer_block = new TextBlock();
                answer_block.Text = word.word_number + " - " + word.question;
                answer_block.Width = preview_canvas.Width;
                answer_block.TextWrapping = TextWrapping.Wrap;
                previewlistBox.Items.Add(answer_block);
```

```
                }
            }
        }
    }
}
previewlistBox.SetValue(
ScrollViewer.HorizontalScrollBarVisibilityProperty,
ScrollBarVisibility.Disabled);
previewlistBox.BorderThickness = new Thickness(0, 0, 0, 0);
previewlistBox.IsEnabled = false;
previewlistBox.Width = preview_canvas.Width - 2 * mw.canvas_paddingL;
previewlistBox.InvalidateArrange();
previewlistBox.UpdateLayout();
preview_canvas.Height = preview_canvas.Height;
Canvas.SetLeft(previewlistBox, mw.canvas_paddingL);
Canvas.SetTop(previewlistBox, preview_canvas.Height+10);
preview_canvas.Children.Add(previewlistBox);
preview_canvas.UpdateLayout();
preview_canvas.Height = preview_canvas.Height + previewlistBox.ActualHeight;
}
public void setCanvasPadding()
{
preview_canvas.Width = preview_canvas.Width + 2 * mw.canvas_paddingL;
preview_canvas.Height = preview_canvas.Height + 2 * mw.canvas_paddingT;
}
private void Window_Activated(object sender, EventArgs e)
{
mw.Cw_Draw(ref preview_canvas);
drawQuestions(content);
```

```
    setCanvasPadding();
}
public void clear_preview_canvas()
{
    preview_canvas.Children.Clear();
    preview_canvas.Height = 0;
}

private void redraw_preview()
{
    if (mw != null)
    {
        bool place_words = true;
        if (words_status.IsChecked == true)
        {
            place_words = true;
        }
        else
        {
            place_words = false;
        }
        // сетка + вопросы
        if (export_data_choose.SelectedIndex == 0)
        {
            clear_preview_canvas();
            if (standart_view.IsChecked == true)
            {
                mw.Cw_Draw_preview(ref preview_canvas, place_words);
            }
```

```
            else
            {
                mw.Cw_Draw(ref preview_canvas, place_words);
            }
            drawQuestions(content);
            setCanvasPadding();
        }
        else if (export_data_choose.SelectedIndex == 1)
        {
            clear_preview_canvas();
            if (standart_view.IsChecked == true)
            {
                mw.Cw_Draw_preview(ref preview_canvas, place_words);
            }
            else
            {
                mw.Cw_Draw(ref preview_canvas, place_words);
            }
            setCanvasPadding();
        }
        else
        {
            clear_preview_canvas();
            drawQuestions(content);
            setCanvasPadding();
        }
    }
}
// выбор данных для экспорта
```

```
        private void ComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
        {
            redraw_preview();
        }
        // стандартный вид (вкл/выкл)
        private void Standart_view_Checked(object sender, RoutedEventArgs e)
        {
            redraw_preview();
        }
        // слова (вкл/выкл)
        private void Words_status_Checked(object sender, RoutedEventArgs e)
        {
            redraw_preview();
        }
        private void Preview_print_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                PrintDialog dialog = new PrintDialog();
                if (dialog.ShowDialog() != true)
                    return;
                dialog.PrintVisual(preview_canvas, "Print");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Print Screen", MessageBoxButton.OK,
MessageBoxImage.Error);
            }
        }
```

```
    }
}
```

addWordWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.IO;
namespace Crossword
{
    /// <summary>
    /// Логика взаимодействия для Window1.xaml
    /// </summary>
    public partial class addWordWindow : Window
    {
        MainWindow mw = null;
        CW_Content content = null;
        public addWordWindow(MainWindow m, CW_Content cnt)
        {
```

```
    InitializeComponent();
    content = cnt;
    mw = m;
}
private void AddWord_Click(object sender, RoutedEventArgs e)
{
    string word_val = addWordAnswer.Text;
    string quest_val = AddWordQuestion.Text;
    int lword = word_val.Trim().Length;
    int qlen = quest_val.Trim().Length;
    string answ = word_val.ToUpper();
    string qst = quest_val;
    if (lword >= 2)
    {
        if(qlen >= 2) {
            List<CW_word_list_item> _words = null;
            // если в списке есть массив для слов длины lword
            if (content.items_list.ContainsKey(lword))
                _words = content.items_list[lword];
            else
            {
                _words = new List<CW_word_list_item>();
                content.items_list.Add(lword, _words);
            }
            bool _exists = false;
            IEnumerator<CW_word_list_item> _items = _words.GetEnumerator();
            _items.Reset();
            while (!_exists && _items.MoveNext())
            {
```

```
            CW_word_list_item _item = _items.Current;
            if (_item.answer == answ)
                _exists = true;
        }
        if (!_exists)
        {
            if (File.Exists(content.dictFileName))
            {
                _words.Add(new CW_word_list_item(lword, qst, answ));
                content.words_num++;
                mw.cw_words_number.Content = content.words_num;
                addWordAnswer.Text = "";
                AddWordQuestion.Text = "";
                string appendText = answ + "=" + qst + Environment.NewLine;
                using (StreamWriter w = new StreamWriter(content.dictFileName, true,
Encoding.GetEncoding(1251)))
                {
                    w.Write(appendText);
                }
                MessageBox.Show("Слово добавлено");
            }
            else
            {
                MessageBox.Show("Не удалось записать в файл");
            }
        }
        else
        {
            MessageBox.Show("Слово уже есть в словаре");
```

```
            }
        }
        else
        {
            MessageBox.Show("Минимальная длина вопроса - 2 символа");
        }
    }
    else
    {
        MessageBox.Show("Минимальная длина слова - 2 символа");
    }
}
private void CloseAddWord_Click(object sender, RoutedEventArgs e)
{
    this.Hide();
}
private void AddWordAnswer_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    if (!System.Text.RegularExpressions.Regex.IsMatch(e.Text, "^[а-яА-Я]"))
    {
        e.Handled = true;
    }
}
private void AddWordQuestion_PreviewTextInput(object sender, TextCompositionEventArgs e)
{
    if (!System.Text.RegularExpressions.Regex.IsMatch(e.Text, "^[а-яА-Я]"))
    {
        e.Handled = true;
    }
```

```
        }
    }
}
```