ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Научный руководитель, доцент департамента больших данных и информационного поиска факультета компьютерных наук

_____ И. Ю. Самоненко
«_____» _____ 2020 г.

УТВЕРЖДАЮ
Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

ПРИЛОЖЕНИЕ ДЛЯ ГЕНЕРАЦИИ ЗАДАЧ

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.01-01 12 01-1-ЛУ

Исполнитель:
студент группы БПИ 199
_____ Е. В. Голикова
«_____» _____ 2020 г.

Москва 2020

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.01-01 12 01-1				

УТВЕРЖДЕН
RU.17701729.04.01-01 12 01-1-ЛУ

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата
RU.17701729.04.01-01 12 01-1				

ПРИЛОЖЕНИЕ ДЛЯ ГЕНЕРАЦИИ ЗАДАЧ

Текст программы

RU.17701729.04.01-01 12 01-1

Листов 50

Москва 2020

Содержание

1 Библиотека классов Library **3**
1.1 Generator.cs . . . 3
1.2 Problems.cs . . . 5
1.3 Tables.cs . . . 18
1.4 HTMLwriter.cs . . . 25
1.5 PDFwriter.cs . . . 30

2 Проект ProblemsGenerator **34**
2.1 Program.cs . . . 34
2.2 Form1.cs . . . 34
2.3 Form1.Designer.cs . . . 40
2.4 HtmlTemplate.txt . . . 49

1 Библиотека классов Library

1.1 Generator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Text;
using System.Threading.Tasks;

namespace Library
{
    // Класс, запускающий генерацию
    public static class Generator
    {
        static Random rand;

        // Делегат для методов генерации задач
        delegate string[] GenerateType();
        // Ключ генерации
        public static int Seed { get; set; }
        // Тип задачи
        public static int ProblemsType { get; set; }
        // Количество задач
        public static int ProblemsNum { get; set; }

        /// <summary>
        /// Генерирует заданное пользователем количество задач заданного типа
        /// </summary>
        /// <returns>Двумерный массив - на первой строке условия, на второй -
        ответы</returns>
        public static string[,] Generate()
        {
            // Проверяем, был ли указан ключ генерации
            if (Seed == 0) rand = new Random();
            else rand = new Random(Seed);
            Problems.Rand = rand;

            GenerateType GenerateMethod;
            if (ProblemsType == 0) GenerateMethod = Problems.GenOneHeap;
            else if (ProblemsType == 1) GenerateMethod = Problems.GenTwoHeaps;
            else GenerateMethod = Problems.GenTwoWords;

            string[,] problems = new string[3, ProblemsNum];
            string[] result = new string[1];
            for (int i = 0; i < ProblemsNum; i++)
            {
                try
```

```
            {
                result = GenerateMethod();
            }
            catch (Exception e)
            {
                MessageBox.Show("Возникла ошибка при генерации задачи.\n" +
                    "Приложение принудительно завершит работу. " + e.Message);
                Environment.Exit(0);
            }
            problems[0, i] = $"Задача {i + 1}<br>" + result[0];
            problems[1, i] = result[1];
            problems[2, i] = result[2];
        }
        return problems;
    }

    /// <summary>
    /// Генерирует заданное пользователем количество задач разных типов,
    выбирая их рандомно
    /// </summary>
    /// <returns>Двумерный массив - на первой строке условия, на второй -
    ответы</returns>
    public static string[,] RandomGenerate()
    {
        // Проверяем, был ли указан ключ генерации
        if (Seed == 0) rand = new Random();
        else rand = new Random(Seed);

        // Передаем созданный объект рандома в класс генерации задач
        Problems.Rand = rand;

        string[,] problems = new string[3, ProblemsNum];
        string[] result = new string[1];
        // Каждый раз создаем задачу рандомного типа
        for (int i = 0; i < ProblemsNum; i++)
        {
            try
            {
                switch (rand.Next(3))
                {
                    case 0:
                        {
                            result = Problems.GenOneHeap();
                            break;
                        }
                    case 1:
                        {
                            result = Problems.GenTwoHeaps();
                            break;
```

```
                        }
                    case 2:
                        {
                            result = Problems.GenTwoWords();
                            break;
                        }
                    default:
                        {
                            result = new string[1];
                            break;
                        }
                }
            }
            catch (Exception e)
            {
                MessageBox.Show("Возникла ошибка при генерации задачи.\n" +
                    "Приложение принудительно завершит работу. " + e.Message);
                Environment.Exit(0);
            }
            problems[0, i] = $"Задача {i + 1}<br>" + result[0];
            problems[1, i] = result[1];
            problems[2, i] = result[2];
        }
        return problems;
    }

}
}
```

1.2 Problems.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Text;
using System.Threading.Tasks;

namespace Library
{
    /// <summary>
    /// Класс, содержащий методы для генерации условий и ответов для разных видов
    задач
    /// </summary>
    public static class Problems
    {
        // Экземпляр рандома, созданный в классе Generator
        public static Random Rand { get; set; }
```

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.04.01-01 12 01-1				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

```
// Делегат для методов, являющихся арифметическими операциями
public delegate int Adder(int x);

/// <summary>
/// Делает строку квадратной таблицей
/// </summary>
/// <param name="str">Строка</param>
/// <returns>Таблица</returns>
static string[,] StrToArray(string str)
{
    string[] arrOfRows = str.Split('\n');
    int len = arrOfRows.Length;
    var res = new string[len, len];
    string[] row;
    for (int i = 0; i < len; i++)
    {
        row = arrOfRows[i].Split(' ');
        for (int j = 0; j < len; j++)
        {
            res[i, j] = row[j];
        }
    }

    return res;
}

/// <summary>
/// Генерирует задачу и ответ для типа "одна куча камней"
/// </summary>
/// <returns>Массив из двух строк - условие и ответ</returns>
public static string[] GenOneHeap()
{
    // Количество действий
    int numOfActions = Rand.Next(2, 5);
    // Действия сложения
    int[] toAdd = new int[numOfActions - 1];
    int num;
    // Добавляем действия сложения (все действия - 1), так, чтобы не
    повторялось
    for (int i = 0; i < numOfActions - 1; i++)
    {
        do
        {
            num = Rand.Next(1, 4);
        } while (toAdd.Contains(num));
        toAdd[i] = num;
    }
    Array.Sort(toAdd);
```

```
// Действие умножения (пусть будет одно)
int toMult = Rand.Next(2, 4);

// Количество камней для выигрыша
int toWin = Rand.Next(25, 66);

string tableStr = string.Empty;
// Верхнее ограничение для выигрыша
int upperBound = (toWin - toAdd.Max()) * toMult - Rand.Next(8, 14);
// Создаем таблицу для данных
try
{
    tableStr = Tables.OneHeap(toAdd, toMult, toWin, upperBound);
}
catch (Exception e)
{
    MessageBox.Show("Возникла ошибка при рассчитывании решения.\n" +
    e.Message);
    Environment.Exit(0);
}

string[] table = tableStr.Split(' ');

/// Вопрос 1б
// Список клеток, из которых можно выиграть первым ходом.
List<int> oneMoveWins = new List<int>();

// Добавляем те, которые выигрышные из-за сложения
foreach (var term in toAdd)
{
    for (int i = toWin - 1; i >= toWin - term; i--)
    {
        if (!oneMoveWins.Contains(i))
        {
            oneMoveWins.Add(i);
        }
    }
}
oneMoveWins.Sort();

// Сразу создадим строку для первого ответа
string ans1a = $"{(int)Math.Ceiling((double)toWin /
toMult)}-{upperBound / toMult}, " +
    string.Join(", ", oneMoveWins);

// И список для вопроса 1б
List<int> quest1bInts = new List<int>();
// Добавляем в него все, что выигрывается не первым ходом из-за
ограничения сверху
```

```
for (int i = upperBound / toMult + 1; i < toWin - toAdd[toAdd.Length -
1]; i++)
{
    quest1bInts.Add(i);
}

// Нужно оставить всего 3-4 штуки, остальные удаляем
int toDelete = quest1bInts.Count - Rand.Next(3, 5);
for (int i = 0; i < toDelete; i++)
{
    quest1bInts.RemoveAt(Rand.Next(quest1bInts.Count));
}
// Делаем строкой
string quest1b = string.Join(", ", quest1bInts);

// Добавим в список выигрыши от умножения
for (int i = (int)Math.Ceiling((double)toWin / toMult); i <=
upperBound / toMult; i++)
{
    oneMoveWins.Add(i);
}

/// Вопросы 2 и 3
// Создаем переменные для первых и вторых плюсов и минусов
string quest2Win, quest2Lose, quest3Win, quest3Lose;
quest2Win = quest2Lose = quest3Win = quest3Lose = string.Empty;
// Проходимся примерно с середины таблицы в начало
for (int i = upperBound / toMult; i > 0; i--)
{
    // Если клетка проигрыша, то проверям, первая или вторая, и
    записываем ее
    if (table[i] == "-")
    {
        if (quest2Lose.Length == 0)
            quest2Lose = i.ToString();
        else if (quest2Lose.Length > 0 && quest2Win.Length > 0 &&
        quest3Lose.Length == 0)
            quest3Lose = i.ToString();
    }
    // Если клетка выигрыша, то тоже проверяем, а если это вторая
    выигрышная,
    // то выходим из цикла - данные нам больше не нужны
    if (table[i] == "+" || table[i] == ".")
    {
        if (quest2Lose.Length > 0 && quest2Win.Length == 0)
            quest2Win = i.ToString();
        else if (quest3Lose.Length > 0)
        {
            quest3Win = i.ToString();
```

```
            break;
        }
    }
}

// Выбираем рандомом, чтобы в третьем вопросе выводил клетку - или +
string quest3;
if (Rand.Next(2) == 1 && quest3Win.Length > 0)
    quest3 = quest3Win;
else
    quest3 = quest3Lose;

// Строка с шаблоном задания
string text = "Два игрока, Петя и Ваня, играют в следующую игру. Перед
игроками лежит куча " +
    "камней. Игроки ходят по очереди, первый ход делает Петя. За один
    ход игрок может:<br>" +
    "- добавить в кучу любое допустимое количество камней: {0};<br>" +
    "- увеличить количество камней в куче в {1} раза.<br>" +
    "Игра завершается в тот момент, когда количество камней в куче
    становится не менее {2}. " +
    "Если при этом в куче оказалось<br>не более {3} камней, " +
    "то победителем считается игрок, сделавший последний ход. В
    противном случае победителем<br>" +
    "становится его противник. В начальный момент в куче было S
    камней, 1  S  {4}.<br>" +
    "Задание 1.<br>" +
    "а) При каких значениях числа S Петя может выиграть в один ход?
    Укажите все такие " +
    "значения и соответствующие ходы Пети.<br>" +
    "б) У кого из игроков есть выигрышная стратегия при S = {5}?
    Опишите " +
    "выигрышные стратегии для этих случаев.<br>" +
    "Задание 2. У кого из игроков есть выигрышная стратегия при S =
    {6}? " +
    "Опишите соответствующие выигрышные стратегии.<br>" +
    "Задание 3. У кого из игроков есть выигрышная стратегия при S =
    {7}? Постройте дерево " +
    "всех партий, возможных при этой выигрышной стратегии (в виде
    рисунка или таблицы). " +
    "На рёбрах дерева указывайте, кто делает ход, в узлах - количество
    камней в позиции.";

// Строка для форматирования шаблона
string[] data = new string[] { string.Join(", ", toAdd),
toMult.ToString(),
    toWin.ToString(), upperBound.ToString(), (toWin-1).ToString(),
    quest1b,
```

```
            string.Join(", ", new string[] { quest2Win, quest2Lose }), quest3
            };

        // Создаем шаблон для ответа и записываем ответ для каждого пункта.
        string answer = "1a) {0}<br>1б) {1}<br>2) {2}<br>3) {3}<br>Развернутые
        ответы проверяются учителем. " +
            "Но есть таблица, по которой можно проверить успешность любой
            стратегии.<br>В шапке " +
            "указано количество камней, и если ему соответствует знак
            &#171;+&#187;, то находящийся " +
            "в этой позиции игрок выиграет, если &#171;-&#187;, то
            проиграет.<br>Пропуск " +
            "&#171;..&#187; означает много идущих подряд выигрышных
            клеток.<br><br>";

        string ans1b = string.Empty;

        for (int i = 0; i < quest1bInts.Count; i++)
        {
            ans1b += $"S = {quest1bInts[i]}: " +
            ("+.".Contains(table[quest1bInts[i]]) ? "Петя<br>" : "Ваня<br>");
        }

        string ans2 = $"S = {quest2Win}: " +
        ("+.".Contains(table[int.Parse(quest2Win)]) ? "Петя<br>" : "Ваня<br>")
        +
            $"S = {quest2Lose}: " +
            ("+.".Contains(table[int.Parse(quest2Lose)]) ? "Петя<br>" :
            "Ваня<br>");

        string ans3 = $"S = {quest3}: " +
        ("+.".Contains(table[int.Parse(quest3)]) ? "Петя<br>" : "Ваня<br>");

        // Строка для форматирования шаблона
        string[] answers = new string[] { ans1a, ans1b, ans2, ans3 };

        return new string[] { string.Format(text, data), string.Format(answer,
        answers), tableStr };
    }

    /// <summary>
    /// Генерирует задачу и ответ для типа "две кучи камней"
    /// </summary>
    /// <returns>Массив из двух строк - условие и ответ</returns>
    public static string[] GenTwoHeaps()
    {
        // Генерация числовых значений
        int toAdd = Rand.Next(1, 3);
        int toMult = Rand.Next(2, 4);
```

```
int toWin = Rand.Next(35, 81);
// Создание таблицы
string tableStr = string.Empty;
try
{
    tableStr = Tables.TwoHeaps(toAdd, toMult, toWin);
}
catch (Exception e)
{
    MessageBox.Show("Возникла ошибка при рассчитывании решения.\n" +
    e.Message);
    Environment.Exit(0);
}

string[,] table = StrToArray(tableStr);

// Нашли все клетки с -1
List<int[]> quest1array = new List<int[]>();
for (int i = 1; i < toWin; i++)
{
    for (int j = i; j < toWin; j++)
    {
        if (table[i, j] == "-1")
            quest1array.Add(new int[] { i, j });
    }
}

// Теперь ищем клетки с +2 (в окрестности клетки -1 на две координаты
назад)
List<int[]> quest2array = new List<int[]>();
foreach (var pair in quest1array)
{
    for (int i = pair[0] - 2; i <= pair[0]; i++)
    {
        for (int j = pair[1] - 2; j <= pair[1]; j++)
        {
            if (i > 0 && j > 0 && i <= j)
            {
                if (table[i, j] == "+2")
                    quest2array.Add(new int[] { i, j });
            }
        }
    }
}

// Ищем клетки -1/2 или -2 (в окрестностях +2 на одну координату
назад)
List<int[]> quest3array = new List<int[]>();
foreach (var pair in quest2array)
```

```
{
    for (int i = pair[0] - 1; i <= pair[0]; i++)
    {
        for (int j = pair[1] - 1; j <= pair[1]; j++)
        {
            if (i > 0 && j > 0 && i <= j)
            {
                if (table[i, j][0] == '-')
                    quest3array.Add(new int[] { i, j });
            }
        }
    }
}

// Выбрали рандомно две клетки -1, сделали строкой
string[] quest1filtered = new string[2];
int indToAdd;
for (int i = 0; i < 2; i++)
{
    indToAdd = Rand.Next(quest1array.Count);
    quest1filtered[i] = $"({string.Join(", ",
    quest1array[indToAdd])})";
    quest1array.RemoveAt(indToAdd);
}
string quest1 = string.Join(", ", quest1filtered);

// Выбрали рандомно две или три клетки +2, сделали строкой
int len = Rand.Next(2, 4);
string[] quest2filtered = new string[len];
for (int i = 0; i < len; i++)
{
    indToAdd = Rand.Next(quest2array.Count);
    quest2filtered[i] = $"({string.Join(", ",
    quest2array[indToAdd])})";
    quest2array.RemoveAt(indToAdd);
}
string quest2 = string.Join(", ", quest2filtered);

// Сделали строкой координаты для третьего вопроса
string quest3 = $"({string.Join(", ",
quest3array[Rand.Next(quest3array.Count)])})";

// Строка с шаблоном задания
string text = "Два игрока, Петя и Ваня, играют в следующую игру. Перед
игроками лежат две кучи " +
    "камней. Игроки ходят по очереди, первый ход делает Петя. За один
    ход игрок может:<br> а) добавить " +
    "в одну из куч (по своему выбору) {0};<br> б) увеличить количество
    камней в куче в {1} " +
```

```
        "раза.<br>Победителем считается игрок, сделавший последний ход,
        т.е. первым получивший такую " +
        "позицию, что в обеих кучах всего будет {2} камней или больше.<br>
        Задание 1. Для каждой из " +
        "начальных позиций {3} укажите, кто из игроков имеет выигрышную
        стратегию.<br>" +
        "Задание 2. Для каждой из начальных позиций {4} укажите, кто из
        игроков " +
        "имеет выигрышную стратегию.<br> Задание 3. Для начальной позиции
        {5} укажите, кто из игроков " +
        "имеет выигрышную стратегию. Постройте дерево всех партий,
        возможных при указанной выигрышной " +
        "стратегии.";

    // Строка для форматирования шаблона
    string[] data = new string[] { toAdd.ToString() + (toAdd == 1 ? "
    камень" : " камня"),
        toMult.ToString(), toWin.ToString(), quest1, quest2, quest3 };

    // Строка с шаблоном ответа
    string answer = "1.<br>{0}<br>2.<br>{1}<br>3. {2}<br>Развернутые
    ответы проверяются учителем. " +
        "Но есть таблица, по которой можно проверить успешность любой
        стратегии.<br>В шапках " +
        "указано количество камней в кучах, и если позиции соответствует
        знак &#171;+&#187;, то " +
        "находящийся в ней игрок выиграет, если &#171;-&#187;, то
        проиграет.<br>Пропуск " +
        "&#171;..&#187; означает много идущих подряд выигрышных
        клеток.<br><br>";
    string ans1 = string.Empty;

    // Создаем ответы
    foreach (var pair in quest1filtered)
    {
        ans1 += $"{pair}: Ваня<br>";
    }

    string ans2 = string.Empty;

    foreach (var pair in quest2filtered)
    {
        ans2 += $"{pair}: Петя<br>";
    }

    string ans3 = quest3 + ": Ваня<br>";

    // Возвращаем отформатированные текст и решение
    return new string[] { string.Format(text, data),
```

```
            string.Format(answer, new string[] { ans1, ans2, ans3 }), tableStr
            };
    }

    /// <summary>
    /// Генерирует задачу и ответ для типа "два слова"
    /// </summary>
    /// <returns>Массив из двух строк - условие и ответ</returns>
    public static string[] GenTwoWords()
    {
        // Генерируем данные для условия
        int[] toAdd = new int[] { Rand.Next(1, 3), Rand.Next(1, 3) };
        int[] toMult = new int[] { Rand.Next(2, 4), Rand.Next(2, 4) };
        Adder[] actionsX = new Adder[] { x => x + toAdd[0], x => x * toMult[0]
        };
        Adder[] actionsY = new Adder[] { x => x + toAdd[1], x => x * toMult[1]
        };
        int toWin = Rand.Next(30, 61);

        // Создаем табличку для этих даннных
        string tableStr = string.Empty;
        try
        {
            tableStr = Tables.TwoWords(actionsX, actionsY, toWin);
        }
        catch (Exception e)
        {
            MessageBox.Show("Возникла ошибка при рассчитывании решения.\n" +
            e.Message);
            Environment.Exit(0);
        }

        string[,] table = StrToArray(tableStr);

        // Нашли все клетки с -1
        List<int[]> minus1array = new List<int[]>();
        for (int i = 1; i < toWin; i++)
        {
            for (int j = i; j < toWin; j++)
            {
                if (table[i, j] == "-1") minus1array.Add(new int[] { i, j });
            }
        }

        // Клетки с +2 (в окрестности клетки -1 на две координаты назад)
        List<int[]> plus2array = new List<int[]>();
        foreach (var pair in minus1array)
        {
            for (int i = pair[0] - 2; i <= pair[0]; i++)
```

```
    {
        for (int j = pair[1] - 2; j <= pair[1]; j++)
        {
            if (i > 0 && j > 0 && i <= j)
            {
                if (table[i, j] == "+2") plus2array.Add(new int[] { i,
                j });
            }
        }
    }
}

// Клетки с -1,2 (в окрестности клетки +2 на две координаты назад)
List<int[]> minus12array = new List<int[]>();
foreach (var pair in plus2array)
{
    for (int i = pair[0] - 2; i <= pair[0]; i++)
    {
        for (int j = pair[1] - 2; j <= pair[1]; j++)
        {
            if (i > 0 && j > 0 && i <= j)
            {
                if (table[i, j] == "-1,2") minus12array.Add(new int[]
                { i, j });
            }
        }
    }
}

// Клетки с + (в окрестностях -1,2 на две координаты назад)
List<int[]> plusArray = new List<int[]>();
foreach (var pair in minus12array)
{
    for (int i = pair[0] - 2; i <= pair[0]; i++)
    {
        for (int j = pair[1] - 2; j <= pair[1]; j++)
        {
            if (i > 0 && j > 0 && i <= j)
            {
                if (table[i, j] == "+") plusArray.Add(new int[] { i, j
                });
            }
        }
    }
}

// Список пар для первого вопроса
List<string> quest1 = new List<string>();
// Список ответов для первого вопроса
```

```
List<string> ans1 = new List<string>();

// Добавляем в него две -1
int indToAdd;
for (int i = 0; i < 2; i++)
{
    indToAdd = Rand.Next(minus1array.Count);
    quest1.Add($"({string.Join(", ", minus1array[indToAdd])})");
    ans1.Add($"({string.Join(", ", minus1array[indToAdd])}): Ваня");
    minus1array.RemoveAt(indToAdd);
}

// Добавляем один + в рандомное место
indToAdd = Rand.Next(plusArray.Count);
// Позиция на которой будет эта пара
int placeToAdd = Rand.Next(quest1.Count + 1);
quest1.Insert(placeToAdd, $"({string.Join(", ",
plusArray[indToAdd])})");
ans1.Insert(placeToAdd, $"({string.Join(", ", plusArray[indToAdd])}):
Петя");
plusArray.RemoveAt(indToAdd);

// Список для второго вопроса
List<string> quest2 = new List<string>();
// Список ответов для второго вопроса
List<string> ans2 = new List<string>();

// Добавляем в него две +2
for (int i = 0; i < 2; i++)
{
    indToAdd = Rand.Next(plus2array.Count);
    quest2.Add($"({string.Join(", ", plus2array[indToAdd])})");
    ans2.Add($"({string.Join(", ", plus2array[indToAdd])}): Петя");
    plus2array.RemoveAt(indToAdd);
}

// В рандомное место в нем одну или две -1/2
for (int i = 0; i < Rand.Next(1, 3); i++)
{
    indToAdd = Rand.Next(minus12array.Count);
    placeToAdd = Rand.Next(quest2.Count + 1);
    quest2.Insert(placeToAdd, $"({string.Join(", ",
    minus12array[indToAdd])})");
    ans2.Insert(placeToAdd, $"({string.Join(", ",
    minus12array[indToAdd])}): Ваня");
    minus12array.RemoveAt(indToAdd);
}

// В третьем вопросе - рандом: либо +2, либо -1/2, либо +
```

```
int indQuest3 = Rand.Next(3);
string quest3 = string.Empty;
string ans3 = string.Empty;
switch (indQuest3)
{
    case 0:
        {
            indToAdd = Rand.Next(plus2array.Count);
            quest3 = $"({string.Join(", ", plus2array[indToAdd])})";
            ans3 = "Петя<br>";
            break;
        }
    case 1:
        {
            indToAdd = Rand.Next(minus12array.Count);
            quest3 = $"({string.Join(", ", minus12array[indToAdd])})";
            ans3 = "Ваня<br>";
            break;
        }
    case 2:
        {
            indToAdd = Rand.Next(plusArray.Count);
            quest3 = $"({string.Join(", ", plusArray[indToAdd])})";
            ans3 = "Петя<br>";
            break;
        }
}

// Строка с шаблоном задания
string text = "Два игрока, Петя и Ваня играют в игру с цепочками
символов. Игра начинается " +
    "со слова, которое состоит из n букв X и m букв Y. Такое слово
    будем обозначать как (n, m). " +
    "Игроки ходят по очереди, первый ход делает Петя. За один ход
    игрок может:<br>" +
    "1) добавить в слово {0} X;<br>2) добавить в слово {1} Y;<br>3)
    увеличить " +
    "количество букв X в {2} раза;<br>4) увеличить количество букв Y в
    {3} раза.<br>Игра " +
    "завершается в тот момент, когда длина слова становится не менее
    {4} символов. Победителем " +
    "считается игрок, сделавший последний ход, т.е. первым получивший
    слово длиной {4} или " +
    "больше.<br>Задание 1. Для каждой из начальных позиций {5}
    укажите, кто из игроков имеет " +
    "выигрышную стратегию.<br>Задание 2. Для каждой из начальных
    позиций {6} укажите, кто из " +
    "игроков имеет выигрышную стратегию.<br>Задание 3. Для начальной
    позиции {7} укажите, " +
```

```
                "кто из игроков имеет выигрышную стратегию. Постройте дерево всех
                партий, возможных при " +
                "указанной выигрышной стратегии.";

            // Данные для форматирования шаблона
            string[] data = new string[] { toAdd[0] == 1 ? "одну букву" : "две
            буквы",
                toAdd[1] == 1 ? "одну букву" : "две буквы", toMult[0].ToString(),
                toMult[1].ToString(),
                toWin.ToString(), string.Join(", ", quest1), string.Join(", ",
                quest2), quest3 };

            // Шаблон ответа
            string answer = "1.<br>{0}<br><br>2.<br>{1}<br><br>3.
            {2}<br>Развернутые ответы проверяются учителем. " +
                "Но есть таблица, по которой можно проверить успешность любой
                стратегии.<br>В шапках " +
                "указано количество камней в кучах, и если позиции соответствует
                знак &#171;+&#187;, то " +
                "находящийся в ней игрок выиграет, если &#171;-&#187;, то
                проиграет.<br>Пропуск " +
                "&#171;..&#187; означает много идущих подряд выигрышных
                клеток.<br><br>";

            return new string[] {
                string.Format(text, data),
                string.Format(answer, new string[] {
                    string.Join("<br>", ans1),
                    string.Join("<br>", ans2),
                    ans3 }),
                tableStr };
        }
    }
}
```

1.3 Tables.cs

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Library
{
    /// <summary>
    /// Класс, содержащий методы для создания таблиц для разных видов задач
    /// </summary>
    public static class Tables
    {
```

```
/// <summary>
/// Делает 2Д таблицу строковой
/// </summary>
/// <param name="arr">Таблица</param>
/// <returns>Таблица в виде одной строки</returns>
static string ArrToStr(string[,] arr)
{
    string[] joinedRows = new string[arr.GetLength(0)];
    for (int i = 0; i < arr.GetLength(0); i++)
    {
        joinedRows[i] = string.Join(" ", GetRow(arr, i));
    }
    return string.Join("\n", joinedRows);
}

/// <summary>
/// Возвращает строку 2Д таблицы из стрингов
/// </summary>
/// <param name="arr">Таблица</param>
/// <param name="rowNum">Номер строки</param>
/// <returns>Массив - строка</returns>
static string[] GetRow(string[,] arr, int rowNum)
{
    return Enumerable.Range(0, arr.GetLength(1))
            .Select(x => arr[rowNum, x])
            .ToArray();
}

/// <summary>
/// Создает таблицу с исходами для любого количества элементов,
/// для одной кучи
/// </summary>
/// <param name="actions">Список арифметических действий</param>
/// <param name="winMin">Минимальное количество камней для
выигрыша</param>
/// <param name="winMax">Максимальное количество камней для
выигрыша</param>
/// <returns>Таблица с исходами в виде строки</returns>
public static string OneHeap(int[] toAdd, int toMult, int winMin, int
winMax)
{
    string[,] table = new string[1, winMin];
    // Флаг для выхода из массива, если уже определили значение клетки
    bool ok;
    // Для каждого количества камней, для каждого действия
    for (int i = winMin - 1; i > 0; i--)
    {
        ok = false;
        // Проверяем эту клетку для каждого слагаемого
```

```
    foreach (var addend in toAdd)
    {
        // Если ходом мы попадаем в зону выигрыша, то в этой клетке 1
        if ((i + addend >= winMin) && (i + addend <= winMax))
        {
            table[0, i] = "+";
            ok = true;
            break;
        }
        else
        {
            // Если мы попадаем в зону проигрыша, то это тоже зона
            выигрыша
            if (i + addend < winMin && table[0, i + addend] == "-")
            {
                table[0, i] = "+";
                ok = true;
                break;
            }
        }
        // Иначе это зона проигрыша
        table[0, i] = "-";
    }
    if (ok) continue;
    // Теперь так же для множителя
    // Если ходом мы попадаем в зону выигрыша, то в этой клетке 1
    if ((i * toMult >= winMin) && (i * toMult <= winMax))
    {
        table[0, i] = "+";
        continue;
    }
    else
    {
        // Если мы попадаем в зону проигрыша, то это тоже зона
        выигрыша
        if (i * toMult < winMin && table[0, i * toMult] == "-")
        {
            table[0, i] = "+";
            continue;
        }
    }
    // Иначе это зона проигрыша
    table[0, i] = "-";
}

// Заменяем много выигрышных клеток подряд на точки.
int k = (int)Math.Ceiling((double)winMin / toMult);
while (table[0, k] == "+")
{
```

```
            table[0, k++] = ".";
        }

        string tableStr = ArrToStr(table);

        return tableStr;
    }

    /// <summary>
    /// Создает таблицу с исходами для любого количества элементов,
    /// для двух куч
    /// </summary>
    /// <param name="add">Сколько камней добавляется</param>
    /// <param name="mult">Во сколько раз умножается количество камней</param>
    /// <param name="toWin">Сколько камней нужно для выигрыша</param>
    /// <returns>Таблица с исходами в виде строки</returns>
    public static string TwoHeaps(int add, int mult, int toWin)
    {
        // Методы, принимающие на вход четыре варианта развития событий
        // после разных ходов, и возвращающие соответствие своему названию
        bool IsWin1(string[] cells)
        {
            for (int i = 0; i < cells.Length; i++)
            {
                if (cells[i] == "!") return true;
            }
            return false;
        }

        bool IsLoss1(string[] cells)
        {
            int k = 0;
            for (int i = 0; i < cells.Length; i++)
            {
                if (cells[i] == "+1") k++;
            }
            return k == 4;
        }

        bool IsWin2(string[] cells)
        {
            for (int i = 0; i < cells.Length; i++)
            {
                if (cells[i] == "-1") return true;
            }
            return false;
        }

        bool IsLoss2(string[] cells)
```

```
{
    int k = 0;
    for (int i = 0; i < cells.Length; i++)
    {
        if (cells[i] == "+2") { k++; break; }
    }
    return k == 4;
}

bool IsWin(string[] cells)
{
    for (int i = 0; i < cells.Length; i++)
    {
        if (cells[i] == "-") return true;
    }
    return false;
}

string GetValue(int row, int col, string[,] tbl)
{
    // Если сумма камней >= нужной, это выигрышная клетка
    if (row + col >= toWin)
    {
        if (row < toWin && col < toWin)
        {
            tbl[row, col] = "!";
        }
        return "!";
    }
    else if (tbl[row, col] is null)
    {
        // Смотрим на четыре возможных варианта развития событий из
        // этого хода (рекурсией)
        var cellsAfterMove = new string[] {
            GetValue(row + add, col, tbl), GetValue(row, col + add,
            tbl),
            GetValue(row * mult, col, tbl), GetValue(row, col * mult,
            tbl) };
        // По этим четырем вариантам решаем, эта клеточка выигрышная
        или
        // проигрышная
        if (IsWin1(cellsAfterMove)) { tbl[row, col] = "+1"; }
        else if (IsLoss1(cellsAfterMove)) { tbl[row, col] = "-1"; }
        else if (IsWin2(cellsAfterMove)) { tbl[row, col] = "+2"; }
        else if (IsLoss2(cellsAfterMove)) { tbl[row, col] = "-2"; }
        else if (IsWin(cellsAfterMove)) { tbl[row, col] = "+"; }
        else tbl[row, col] = "-";
    }
    return tbl[row, col];
```

```
    }

    // Создаем пустую таблицу
    string[,] table = new string[toWin, toWin];

    // Определяем значение для каждой клетки
    for (int row = 1; row < toWin; row++)
    {
        for (int col = 1; col < toWin; col++)
        {
            GetValue(row, col, table);
        }
    }

    // Место, где начинается много плюсов
    int startWins = (int)Math.Ceiling((double)toWin / mult);
    // Меняем много плюсов на точки
    for (int row = 1; row < toWin - 1; row++)
    {
        for (int col = 1; col < toWin - 1; col++)
        {
            if (col >= startWins || row >= startWins) table[row, col] =
            ".";
        }
    }

    string tableStr = ArrToStr(table);

    return tableStr;
}

/// <summary>
/// Создает таблицу с исходами для любого количества букв,
/// для двух слов
/// </summary>
/// <param name="actionsX">Действия для первого слова</param>
/// <param name="actionsY">Действия для второго слова</param>
/// <param name="toWin">Количество букв для победы</param>
/// <returns>Таблица с исходами в виде строки</returns>
public static string TwoWords(Problems.Adder[] actionsX, Problems.Adder[]
actionsY, int toWin)
{
    string[,] table = new string[toWin, toWin];

    // Идем с правого нижнего угла таблицы
    for (var y = toWin - 1; y > 0; y--)
    {
        for (var x = toWin - 1; x > 0; x--)
        {
```

```
// Изначально ячейка помечена минусом
table[y, x] = "-";
// Массив исходов
List<string> targets = new List<string>();
// Если сумма букв больше toWin, это победная ячейка
if ((y + x) >= toWin)
    table[y, x] = "!";
else
{
    // Применяем по очереди действия для буквы X
    foreach (var actionX in actionsX)
    {
        // Если с каким-то действием попадаем в зону выигрыша,
        // добавляем в исходы победу
        if (y + actionX(x) >= toWin)
        {
            targets.Add("!");
            break;
        }
        // Иначе добавляем значение в полученной ячейке
        else
        {
            targets.Add(table[y, actionX(x)]);
        }
    }
    // То же самое для буквы Y
    foreach (var actionY in actionsY)
    {
        if (x + actionY(y) >= toWin)
        {
            targets.Add("!");
            break;
        }
        else
        {
            targets.Add(table[actionY(y), x]);
        }
    }
    // Если в исходах есть !, то ячейка выигрышная с первого
    хода
    if (targets.IndexOf("!") != -1) table[y, x] = "+1";
    // Иначе если там есть -1, то выигрышная со второго хода
    else if (targets.IndexOf("-1") != -1) table[y, x] = "+2";
    // Если есть более "далекие" выигрыши, то просто
    выигрышная
    else if (targets.IndexOf("-1,2") != -1) table[y, x] = "+";
    else if (targets.IndexOf("-") != -1) table[y, x] = "+";
    // Иначе считаем количество исходов +1 и +2, чтобы
    определить,
```

```
                    // с какого хода данная ячейка проигрышная. Если не с 1 и
                    не с 1/2,
                    // то она просто проигрышная
                    else
                    {
                        table[y, x] = "-";
                        int k1 = 0;
                        int k2 = 0;
                        foreach (string target in targets)
                        {
                            if (target == "+1") k1 += 1;
                            else if (target == "+2") k2 += 1;
                        }
                        if (k1 == targets.Count) table[y, x] = "-1";
                        else if ((k1 + k2) == targets.Count) table[y, x] =
                        "-1,2";
                    }
                }
            }
        }

        // Применяем действия умножения к единице, если получим 2 -
        // значит минимум происходит умножение на 2
        int minMult = Math.Min(actionsX[1](1), actionsY[1](1));

        // Место, где начинается много плюсов, с запасом
        int startWins = (int)Math.Ceiling((double)toWin / minMult);
        // Меняем много плюсов на точки
        for (int row = 1; row < toWin - 1; row++)
        {
            for (int col = 1; col < toWin - 1; col++)
            {
                if (col >= startWins || row >= startWins) table[row, col] =
                ".";
            }
        }

        string tableStr = ArrToStr(table);

        return tableStr;
    }
  }
}
```

1.4 HTMLwriter.cs

```
using System;
using System.Diagnostics;
using System.IO;
```

```
using HtmlAgilityPack;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Library
{
    /// <summary>
    /// Класс для записи данных в html-файл
    /// </summary>
    public static class HTMLWriter
    {
        // Открывать ли файл сразу
        public static bool ToOpen { get; set; }

        /// <summary>
        /// Записывает сгенерированные задачи в html-файл
        /// </summary>
        /// <param name="problems">Данные о задачах</param>
        public static void WriteHTML(string[,] problems)
        {
            // Добываем путь к файлу, в который будем записывать
            string docpath =
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            string path = Path.Combine(docpath, "Task26.html");
            int count = 1;

            // При существовании файла создаем новый, а не перезаписываем
            while (File.Exists(path))
            {
                path = Path.Combine(docpath, $"Task26({count++}).html");
            }

            PDFwriter.FileNum = count - 1;

            // Добываем текст из шаблона и записываем его в файл
            string baseText = File.ReadAllText("HtmlTemplate.txt");
            File.WriteAllText(path, baseText, Encoding.UTF8);

            HtmlAgilityPack.HtmlDocument doc = new HtmlAgilityPack.HtmlDocument();
            doc.Load(path);
            HtmlNode body = doc.DocumentNode.SelectSingleNode("//html/body"),
                task, button, input1, input2, answer, span;

            // Проходимся по каждой задаче из списка
            for (int i = 0; i <= problems.GetUpperBound(1); i++)
            {
```

```
            // Создаем тег для условия задачи и записываем его
            task = doc.CreateElement("p");
            task.InnerHtml = problems[0, i];
            body.AppendChild(task);
            button = doc.CreateElement("p");

            // Создаем два тега для кнопочек, записываем туда атрибуты
            input1 = doc.CreateElement("input");
            input1.SetAttributeValue("class", "colored");
            input1.SetAttributeValue("type", "button");
            input1.SetAttributeValue("value", "Решение");
            input1.SetAttributeValue("onclick", $"showAnswer(\"{problems[1, i]
            + Create2DTable(problems[2, i])}\", 'answer{i}')");
            button.AppendChild(input1);

            input2 = doc.CreateElement("input");
            input2.SetAttributeValue("class", "colored");
            input2.SetAttributeValue("type", "button");
            input2.SetAttributeValue("value", "Спрятать решение");
            input2.SetAttributeValue("onclick", $"hideAnswer('answer{i}')");
            button.AppendChild(input2);

            // Добавляем кнопочки в body
            body.AppendChild(button);

            // Создаем тег для ответа и записываем
            answer = doc.CreateElement("p");
            span = doc.CreateElement("span");
            span.SetAttributeValue("id", $"answer{i}");
            answer.AppendChild(span);
            body.AppendChild(answer);
        }
        doc.Save(path);

        // Открываем страницу в браузере, если стоит галочка
        if (ToOpen)
            Process.Start(new ProcessStartInfo(path) { UseShellExecute = true
            });
    }

    /// <summary>
    /// Создает код для двумерной html-таблицы из строки
    /// </summary>
    /// <param name="str">Строка</param>
    /// <returns>Код таблицы</returns>
    static string Create2DTable(string str)
    {
        // Делаем массив строк таблицы, разделяя по переносу строки
        string[] arrOfRows = str.Split('\n');
```

```
// Длина по горизонтали - длина первой строки
int lenHor = arrOfRows[0].Split(' ').Length;
// Длина по вертикали - количество строк
int lenVert = arrOfRows.Length;

StringBuilder table = new StringBuilder();
// Добавляем тег таблицы и начинаем новую строку
table.Append("<table id = 'table' border = '1'><tr>");

// Добавляем пустую верхнюю левую ячейку
table.Append("<th> </th>");

// Для удобства делаем первую строку массивом
string[] firstRow = lenVert == 1 ? arrOfRows[0].Split(' ') :
arrOfRows[1].Split(' ');
// Длина шапки с учетом пропусков
int headLen = 0;
// Делаем верхнюю шапку
for (int i = 1; i < lenHor; i++)
{
    // Пропускаем все колонки, в которых в исходной таблице стоит
    точка,
    // единожды пишем знак пропуска ".."
    if (firstRow[i] == ".")
    {
        if (firstRow[i - 1] != ".")
            table.Append($"<th class='h'>..</th>");
        continue;
    }
    headLen++;
    if (i < 10) table.Append($"<th class='h'>{i} </th>");
    else table.Append($"<th class='h'>{i}</th>");
}

// Заканчиваем строку
table.Append("</tr>");

// Добавляем сами строки
// Для однострочной таблицы
if (lenVert == 1)
{
    table.Append($"<tr><th class='h'> </th>");
    AddRow(firstRow, ref table);
}
// Для квадратных таблиц
else
{
    bool flag = true;
    string[] cells;
```

```
// Создаем основную часть таблицы
for (int j = 1; j < lenVert; j++)
{
    // Следующая строка таблицы
    cells = arrOfRows[j].Split(' ');

    // Если строка непустая (второй элемент непустой (первый может
    // быть пустой даже у непустой строки))
    if (cells[1].Length > 0)
    {
        // Если состоит из точек, её надо сокращать
        if (cells[1] == ".")
        {
            // Первый раз, пока флаг еще true, создаем строку из
            пропусков
            if (flag)
            {
                table.Append($"<tr><th class='h'>..</th>");
                for (int i = 0; i < headLen; i++)
                {
                    table.Append("<td class='w'>..</td>");
                }
                // В конце - восклицательный знак и конец строки
                table.Append("<td class='r'>!</td></tr>");
                // Меняем флаг, чтобы больше строк не создавалось
                flag = false;
            }
            // И в любом случае - continue
            continue;
        }
        // Если состоит из восклицательных знаков, ее надо
        сокращать
        else if (cells[1] == "!")
        {
            table.Append($"<tr><th class='h'>{cells.Length -
            1}</th>");
            for (int i = 0; i < headLen; i++)
            {
                table.Append("<td class='r'>!</td>");
            }
            // В конце - восклицательный знак и конец строки
            table.Append("<td class='r'>!</td></tr>");
            // Цикл закончен
            break;
        }
    }
    // Первый элемент - индекс для боковой шапки
    table.Append($"<tr><th class='h'>{j}</th>");
    AddRow(cells, ref table);
```

```
                // Заканчиваем строку
                table.Append("</tr>");
            }
        }
        // Заканчиваем таблицу
        table.Append("</table><br>");
        return table.ToString();
    }

    /// <summary>
    /// Добавляет в таблицу строку
    /// </summary>
    /// <param name="cells">Массив элементов строки</param>
    /// <param name="table">Ссылка на таблицу</param>
    static void AddRow(string[] cells, ref StringBuilder table)
    {
        for (int i = 0; i < cells.Length; i++)
        {
            if (cells[i].Length < 1) continue;
            // Красим клетки
            // Если это клетка пропуска, "сокращаем" их до одной
            if (cells[i][0] == '.')
            {
                if (cells[i - 1][0] != '.')
                    table.Append("<td class='w'>..</td>");
                continue;
            }
            else if (cells[i][0] == '+')
                table.Append("<td class='w'>+</td>");
            else if (cells[i][0] == '-')
                table.Append("<td class='l'>-</td>");
            else
                table.Append("<td class='r'>!</td>");
        }
    }
}
```

1.5 PDFwriter.cs

```
using System;
using System.Diagnostics;
using System.IO;
using System.Threading;
using MigraDoc;
using MigraDoc.DocumentObjectModel;
using MigraDoc.Rendering;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading.Tasks;

namespace Library
{
    /// <summary>
    /// Класс для записи данных в pdf-файл
    /// </summary>
    public static class PDFwriter
    {
        // Открывать ли файл сразу
        public static bool ToOpen { get; set; }
        // Размер текста, установленный пользователем
        public static int FontSize { get; set; }
        public static int FileNum { get; set; }

        /// <summary>
        /// Создает одну строку текста с условиями из данных о задачах
        /// </summary>
        /// <param name="problems">Сгенерированные данные о задачах</param>
        public static void BuildText(string[,] problems)
        {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < problems.GetLength(1); i++)
            {
                sb.Append(problems[0, i].Replace("<br>", "\n") + "\n\n");
            }

            // Вызываем функцию записи этого текста в файл
            WritePdf(sb.ToString());
        }

        /// <summary>
        /// Записывает строку текста (условия задач) в pdf с заголовком
        /// </summary>
        /// <param name="text">Условия задач</param>
        static void WritePdf(string text)
        {
            // Добываем путь к файлу, в который будем записывать
            string docpath =
            Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            string path = Path.Combine(docpath, FileNum == 0 ? "Task26.pdf" :
                $"Task26({FileNum}).pdf");

            // Произойдет, если удалить html-файл из пары html+pdf и создать новый
            с таким же номером
            if (File.Exists(path))
```

```
        {
            MessageBox.Show($"Файл с названием, аналогичным создаваемому
            ({path}), уже существует. " +
                "Он будет перезаписан.");
        }

        // Создаем документ
        Document doc = new Document();

        // Определяем стили
        DefineStyles(doc);

        // Создаем секцию текста
        doc.AddSection();
        // Пишем заголовок
        doc.LastSection.AddParagraph("Задание №26 ЕГЭ по информатике",
        "Heading");
        // Пишем основной текст
        doc.LastSection.AddParagraph(text, "BaseText");

        PdfDocumentRenderer renderer = new PdfDocumentRenderer(true)
        {
            Document = doc
        };
        renderer.RenderDocument();
        // Сохраняем документ
        renderer.PdfDocument.Save(path);

        // Открываем файл, если стоит галочка
        if (ToOpen)
        {
            Process.Start(path);
        }
    }

    /// <summary>
    /// Задает стили документа
    /// </summary>
    /// <param name="document">PDF-документ</param>
    static void DefineStyles(Document document)
    {
        // Базовый стиль
        Style style = document.Styles["Normal"];

        // Стиль по умолчанию
        style.Font.Name = "Times New Roman";

        // Стиль для заголовка
        style = document.Styles.AddStyle("Heading", "Normal");
```

```
            style.Font.Size = 18;
            style.Font.Bold = true;
            style.ParagraphFormat.PageBreakBefore = true;
            style.ParagraphFormat.SpaceAfter = 12;
            style.ParagraphFormat.LeftIndent = -15;

            // Стиль основного текста
            style = document.Styles.AddStyle("BaseText", "Normal");
            if (FontSize == 0)
                style.Font.Size = 10;
            else style.Font.Size = FontSize;
            style.ParagraphFormat.PageBreakBefore = false;
            style.ParagraphFormat.SpaceBefore = 20;
            style.ParagraphFormat.SpaceAfter = 6;
            style.ParagraphFormat.LineSpacingRule = LineSpacingRule.AtLeast;
            style.ParagraphFormat.LineSpacing = 15;
            style.ParagraphFormat.LeftIndent = -15;

            // Верхний колонтитул
            style = document.Styles[StyleNames.Header];
            style.ParagraphFormat.AddTabStop("3cm",
            MigraDoc.DocumentObjectModel.TabAlignment.Right);

            // Нижний колонтитул
            style = document.Styles[StyleNames.Footer];
            style.ParagraphFormat.AddTabStop("5cm",
            MigraDoc.DocumentObjectModel.TabAlignment.Center);
        }
    }
}
```

2 Проект ProblemsGenerator

2.1 Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ProblemGenerator
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа программы.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            Application.ThreadException += new
            ThreadExceptionEventHandler(ExceptionHandlingMethod);

            Form1 mainForm = new Form1();
            Application.Run(mainForm);
        }

        // Обработчик события, вызываемого при возникновении исключений
        private static void ExceptionHandlingMethod(object sender,
        ThreadExceptionEventArgs t)
        {
            MessageBox.Show("Произошла ошибка при работе Windows-формы.\n" +
                "Приложение принудительно завершит работу.");
            Environment.Exit(0);
        }
    }
}
```

2.2 Form1.cs

```
using System;
using System.Threading;
using System.IO;
using System.Text.RegularExpressions;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Library;

namespace ProblemGenerator
{
    public partial class Form1 : Form
    {
        private static readonly Random rand = new Random();
        public Form1()
        {
            InitializeComponent();
            FormClosing += new FormClosingEventHandler(Form1_Closing);
            Size = new Size(650, 500);
            // Заранее устанавливаем свойство true, т.к. галочка изначально стоит
            HTMLWriter.ToOpen = true;
            // Ставим единицу, которая стоит по умолчанию
            Generator.ProblemsNum = 1;
            // Загружаем иконку
            try
            {
                Icon = new Icon("favicon.ico");

            }
            catch (Exception)
            {
                MessageBox.Show("Возникли некоторые проблемы с иконкой приложения.
                " +
                    "Будет использована иконка по умолчанию.");
            }
        }
        private void NumTextBox_TextChanged(object sender, EventArgs e)
        {
            // Проверяем, чтобы было введено положительное целое число
            // меньше 1001
            if (numTextBox.Text.Length > 0 && numTextBox.Text[0] == '0')
            {
                numTextBox.Text = numTextBox.Text.Substring(1);
                errorLabel.Visible = true;
                return;
            }
            if (!int.TryParse(numTextBox.Text, out int num) || num > 1000 || num <
            1)
            {
```

```
            // Обрезаем число, убирая введенный символ
            numTextBox.Text = numTextBox.Text.Substring(0,
            Math.Max(numTextBox.Text.Length - 1, 0));
            // Ставим курсор в конец
            numTextBox.SelectionStart = numTextBox.Text.Length;
            numTextBox.SelectionLength = 0;
            errorLabel.Visible = true;
            return;
        }
        Generator.ProblemsNum = num;
        errorLabel.Visible = false;
    }

    private void SeedBox_TextChanged(object sender, EventArgs e)
    {
        // Если пустота, то всё ок
        if (seedBox.Text.Length == 0 && (randBox.Checked ||
        !(typeComboBox.SelectedItem is null)))
        {
            genButton.Visible = true;
            infoLabel.Visible = true;
        }
        // Если что-то начали писать - прячем кнопки, пока не будет
        // 4-хзначного числа
        else
        {
            genButton.Visible = false;
            infoLabel.Visible = false;
        }

        // Проверяем, чтобы на первом месте не было нуля
        if (seedBox.Text.Length > 0 && seedBox.Text[0] == '0')
        {
            seedBox.Text = seedBox.Text.Substring(1);
            return;
        }
        // Проверяем, что это число меньше 5 знаков
        if (!int.TryParse(seedBox.Text, out int num) || num > 9999 || num < 1)
        {
            // Обрезаем число, оставляем только первые четыре цифры
            seedBox.Text = seedBox.Text.Substring(0,
            Math.Max(seedBox.Text.Length - 1, 0));
            // Ставим курсор в конец
            seedBox.SelectionStart = seedBox.Text.Length;
            seedBox.SelectionLength = 0;
            return;
        }
        // Если оно четырехзначное, можно генерировать
        if (num > 999)
```

```
    {
        Generator.Seed = num;
        if (randBox.Checked || !(typeComboBox.SelectedItem is null))
        {
            genButton.Visible = true;
            infoLabel.Visible = true;
        }
    }
}

private void TypeComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
    // Передаем тип в класс генератора, делаем видимой кнопку запуска
    Generator.ProblemsType = typeComboBox.SelectedIndex;
    if (seedBox.Text.Length == 0 || seedBox.Text.Length == 4)
    {
        genButton.Visible = true;
        infoLabel.Visible = true;
    }
}

private void GenButton_Click(object sender, EventArgs e)
{
    // Если количество задач не указано, сообщение об ошибке
    if (numTextBox.Text == "")
    {
        errorLabel.Visible = true;
        return;
    }

    string[,] problemsData;

    if (randBox.Checked)
        problemsData = Generator.RandomGenerate();
    else
        problemsData = Generator.Generate();

    try
    {
        HTMLWriter.WriteHTML(problemsData);
        // Если нужно создать pdf, запускаем это в новом треде,
        // а на экране блокируем все элементы и выводим сообщение
        // об ожидании
        if (createPdfCheckBox.Checked)
        {
            waitLabel.Visible = true;
            StartPdfThread(problemsData);
        }
    }
```

```
        catch (Exception ex)
        {
            MessageBox.Show("Произошла ошибка при создании файла.\n" +
                "Приложение принудительно завершит работу." + ex.Message);
            Environment.Exit(0);
        }
    }

    /// <summary>
    /// Запускает отдельный тред для создания pdf-файла
    /// </summary>
    /// <param name="data">Данные для текстов задач </param>
    private void StartPdfThread(string[,] data)
    {
        void ThreadStarter()
        {
            try
            {
                DisableAll();
                PDFwriter.BuildText(data);
            }
            finally
            {
                EnableAll();
                waitLabel.Visible = false;
            }
        }
        var thread = new Thread(ThreadStarter);
        thread.Start();
    }

    /// <summary>
    /// Делает неактивными все элементы формы
    /// </summary>
    private void DisableAll()
    {
        foreach (Control con in Controls)
        {
            con.Enabled = false;
        }
    }

    /// <summary>
    /// Делает активными все элементы формы
    /// </summary>
    private void EnableAll()
    {
        foreach (Control con in Controls)
        {
```

```
            con.Enabled = true;
        }
    }

    private void RandBox_CheckedChanged(object sender, EventArgs e)
    {
        // Если галочка стоит, делаем видимой кнопку запуска и скрываем выбор
        типа
        if (randBox.Checked && (seedBox.Text.Length == 0 ||
        seedBox.Text.Length == 4))
        {
            genButton.Visible = true;
            infoLabel.Visible = true;
            typeComboBox.Visible = false;
        }
        // Если галочка не стоит и тип задач не выбран, наоборот
        if (!randBox.Checked)
        {
            typeComboBox.Visible = true;
            if (typeComboBox.SelectedItem is null)
            {
                genButton.Visible = false;
                infoLabel.Visible = false;
            }
        }
    }

    private void CreatePdfCheckBox_CheckedChanged(object sender, EventArgs e)
    {
        if (createPdfCheckBox.Checked)
        {
            pdfCheckBox.Visible = true;
            textSizeLabel.Visible = true;
            textSizeComboBox.Visible = true;
        }
        else if (!createPdfCheckBox.Checked)
        {
            pdfCheckBox.Visible = false;
            textSizeLabel.Visible = false;
            textSizeComboBox.Visible = false;
        }
    }

    private void HtmlCheckBox_CheckedChanged(object sender, EventArgs e)
    {
        if (htmlCheckBox.Checked)
            HTMLWriter.ToOpen = true;
        else if (!htmlCheckBox.Checked)
            HTMLWriter.ToOpen = false;
```

```
        }

        private void PdfCheckBox_CheckedChanged(object sender, EventArgs e)
        {
            if (pdfCheckBox.Checked)
                PDFwriter.ToOpen = true;
            else if (!pdfCheckBox.Checked)
                PDFwriter.ToOpen = false;
        }

        private void TextSizeComboBox_SelectedIndexChanged(object sender,
        EventArgs e)
        {
            int.TryParse(textSizeComboBox.Text, out int size);
            if (size == 0) size = 10;
            PDFwriter.FontSize = size;
        }

        private void EscButton_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void DifLabel_Click(object sender, EventArgs e)
        {
            difLabel.ForeColor = Color.FromArgb(rand.Next(256), rand.Next(256),
            rand.Next(256));
        }

        private void NumLabel_Click(object sender, EventArgs e)
        {
            numLabel.ForeColor = Color.FromArgb(rand.Next(256), rand.Next(256),
            rand.Next(256));
        }

        private void Form1_Closing(object sender, FormClosingEventArgs e)
        {
            Application.Exit();
        }
    }
}
```

2.3 Form1.Designer.cs

```
namespace ProblemGenerator
{
    partial class Form1
    {
        /// <summary>
```

```
/// Required designer variable.
/// </summary>
private System.ComponentModel.IContainer components = null;

/// <summary>
/// Clean up any resources being used.
/// </summary>
/// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}

#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.difLabel = new System.Windows.Forms.Label();
    this.typeComboBox = new System.Windows.Forms.ComboBox();
    this.numTextBox = new System.Windows.Forms.TextBox();
    this.numLabel = new System.Windows.Forms.Label();
    this.errorLabel = new System.Windows.Forms.Label();
    this.genButton = new System.Windows.Forms.Button();
    this.title1 = new System.Windows.Forms.Label();
    this.title2 = new System.Windows.Forms.Label();
    this.infoLabel = new System.Windows.Forms.Label();
    this.randBox = new System.Windows.Forms.CheckBox();
    this.seedLabel = new System.Windows.Forms.Label();
    this.seedBox = new System.Windows.Forms.TextBox();
    this.escButton = new System.Windows.Forms.Button();
    this.htmlCheckBox = new System.Windows.Forms.CheckBox();
    this.pdfCheckBox = new System.Windows.Forms.CheckBox();
    this.textSizeComboBox = new System.Windows.Forms.ComboBox();
    this.textSizeLabel = new System.Windows.Forms.Label();
    this.createPdfCheckBox = new System.Windows.Forms.CheckBox();
    this.waitLabel = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // difLabel
    //
```

```
this.difLabel.AutoSize = true;
this.difLabel.Font = new System.Drawing.Font("Comic Sans MS", 13.125F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.difLabel.Location = new System.Drawing.Point(40, 305);
this.difLabel.Name = "difLabel";
this.difLabel.Size = new System.Drawing.Size(389, 49);
this.difLabel.TabIndex = 0;
this.difLabel.Text = "Выберите тип задачи:";
this.difLabel.Click += new System.EventHandler(this.DifLabel_Click);
//
// typeComboBox
//
this.typeComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
this.typeComboBox.Font = new System.Drawing.Font("Microsoft Sans
Serif", 10F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.typeComboBox.FormattingEnabled = true;
this.typeComboBox.Items.AddRange(new object[] {
"одна куча камней",
"две кучи камней",
"два слова"});
this.typeComboBox.Location = new System.Drawing.Point(529, 309);
this.typeComboBox.Name = "typeComboBox";
this.typeComboBox.Size = new System.Drawing.Size(304, 39);
this.typeComboBox.TabIndex = 1;
this.typeComboBox.SelectedIndexChanged += new
System.EventHandler(this.TypeComboBox_SelectedIndexChanged);
//
// numTextBox
//
this.numTextBox.Font = new System.Drawing.Font("Microsoft Sans Serif",
10F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.numTextBox.Location = new System.Drawing.Point(529, 390);
this.numTextBox.Name = "numTextBox";
this.numTextBox.Size = new System.Drawing.Size(100, 38);
this.numTextBox.TabIndex = 2;
this.numTextBox.Text = "1";
this.numTextBox.TextChanged += new
System.EventHandler(this.NumTextBox_TextChanged);
//
// numLabel
//
this.numLabel.AutoSize = true;
this.numLabel.Font = new System.Drawing.Font("Comic Sans MS", 13.125F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
```

```
this.numLabel.Location = new System.Drawing.Point(40, 386);
this.numLabel.Name = "numLabel";
this.numLabel.Size = new System.Drawing.Size(474, 49);
this.numLabel.TabIndex = 3;
this.numLabel.Text = "Введите количество задач:";
this.numLabel.Click += new System.EventHandler(this.NumLabel_Click);
//
// errorLabel
//
this.errorLabel.AutoSize = true;
this.errorLabel.Font = new System.Drawing.Font("Comic Sans MS",
10.125F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.errorLabel.ForeColor =
System.Drawing.Color.FromArgb(((int)(((byte)(192)))),
((int)(((byte)(0)))), ((int)(((byte)(0)))));
this.errorLabel.Location = new System.Drawing.Point(638, 396);
this.errorLabel.Name = "errorLabel";
this.errorLabel.Size = new System.Drawing.Size(613, 38);
this.errorLabel.TabIndex = 4;
this.errorLabel.Text = "Количество задач - целое число от 1 до 1000.";
this.errorLabel.Visible = false;
//
// genButton
//
this.genButton.AutoSize = true;
this.genButton.BackColor =
System.Drawing.Color.FromArgb(((int)(((byte)(255)))),
((int)(((byte)(255)))), ((int)(((byte)(192)))));
this.genButton.Font = new System.Drawing.Font("Comic Sans MS", 13F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.genButton.ForeColor = System.Drawing.Color.Green;
this.genButton.Location = new System.Drawing.Point(485, 624);
this.genButton.Name = "genButton";
this.genButton.Size = new System.Drawing.Size(285, 59);
this.genButton.TabIndex = 6;
this.genButton.Text = "Сгенерировать!";
this.genButton.UseVisualStyleBackColor = false;
this.genButton.Visible = false;
this.genButton.Click += new System.EventHandler(this.GenButton_Click);
//
// title1
//
this.title1.Font = new System.Drawing.Font("Comic Sans MS", 30F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.title1.ForeColor = System.Drawing.Color.Green;
this.title1.Location = new System.Drawing.Point(289, 40);
```

```
this.title1.Name = "title1";
this.title1.Size = new System.Drawing.Size(701, 135);
this.title1.TabIndex = 6;
this.title1.Text = "Генератор задач\r\n";
//
// title2
//
this.title2.Font = new System.Drawing.Font("Comic Sans MS", 25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.title2.ForeColor = System.Drawing.Color.Green;
this.title2.Location = new System.Drawing.Point(165, 162);
this.title2.Name = "title2";
this.title2.Size = new System.Drawing.Size(931, 125);
this.title2.TabIndex = 7;
this.title2.Text = "№26 ЕГЭ по информатике";
//
// infoLabel
//
this.infoLabel.Font = new System.Drawing.Font("Comic Sans MS",
10.125F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.infoLabel.ForeColor = System.Drawing.Color.DarkGoldenrod;
this.infoLabel.Location = new System.Drawing.Point(260, 697);
this.infoLabel.Name = "infoLabel";
this.infoLabel.Size = new System.Drawing.Size(733, 94);
this.infoLabel.TabIndex = 8;
this.infoLabel.Text = "Файлы с готовыми задачами (html и pdf) будут
сохранены в папке «Документы».\r\n";
this.infoLabel.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter;
this.infoLabel.Visible = false;
//
// randBox
//
this.randBox.AutoSize = true;
this.randBox.Font = new System.Drawing.Font("Comic Sans MS", 12F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.randBox.Location = new System.Drawing.Point(908, 307);
this.randBox.Name = "randBox";
this.randBox.Size = new System.Drawing.Size(317, 49);
this.randBox.TabIndex = 9;
this.randBox.Text = "Рандомные типы";
this.randBox.UseVisualStyleBackColor = true;
this.randBox.CheckedChanged += new
System.EventHandler(this.RandBox_CheckedChanged);
//
// seedLabel
```

```
//
this.seedLabel.Font = new System.Drawing.Font("Comic Sans MS", 9.4F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.seedLabel.Location = new System.Drawing.Point(12, 456);
this.seedLabel.Name = "seedLabel";
this.seedLabel.Size = new System.Drawing.Size(490, 81);
this.seedLabel.TabIndex = 10;
this.seedLabel.Text = "Ключ генерации (любое 4-значное число,
необязательно):";
this.seedLabel.TextAlign = System.Drawing.ContentAlignment.TopRight;
//
// seedBox
//
this.seedBox.Font = new System.Drawing.Font("Microsoft Sans Serif",
10F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.seedBox.Location = new System.Drawing.Point(529, 468);
this.seedBox.Name = "seedBox";
this.seedBox.Size = new System.Drawing.Size(100, 38);
this.seedBox.TabIndex = 3;
this.seedBox.TextChanged += new
System.EventHandler(this.SeedBox_TextChanged);
//
// escButton
//
this.escButton.BackColor = System.Drawing.Color.LemonChiffon;
this.escButton.Font = new System.Drawing.Font("Comic Sans MS", 11F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.escButton.ForeColor = System.Drawing.Color.DarkGoldenrod;
this.escButton.Location = new System.Drawing.Point(521, 805);
this.escButton.Name = "escButton";
this.escButton.Size = new System.Drawing.Size(210, 60);
this.escButton.TabIndex = 7;
this.escButton.Text = "Выход";
this.escButton.UseVisualStyleBackColor = false;
this.escButton.Click += new System.EventHandler(this.EscButton_Click);
//
// htmlCheckBox
//
this.htmlCheckBox.AutoSize = true;
this.htmlCheckBox.Checked = true;
this.htmlCheckBox.CheckState =
System.Windows.Forms.CheckState.Checked;
this.htmlCheckBox.Font = new System.Drawing.Font("Comic Sans MS",
9.4F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.htmlCheckBox.Location = new System.Drawing.Point(55, 560);
```

```
this.htmlCheckBox.Name = "htmlCheckBox";
this.htmlCheckBox.Size = new System.Drawing.Size(215, 40);
this.htmlCheckBox.TabIndex = 4;
this.htmlCheckBox.Text = "Открыть html";
this.htmlCheckBox.UseVisualStyleBackColor = true;
this.htmlCheckBox.CheckedChanged += new
System.EventHandler(this.HtmlCheckBox_CheckedChanged);
//
// pdfCheckBox
//
this.pdfCheckBox.AutoSize = true;
this.pdfCheckBox.Font = new System.Drawing.Font("Comic Sans MS", 9.4F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((byte)(204)));
this.pdfCheckBox.Location = new System.Drawing.Point(594, 560);
this.pdfCheckBox.Name = "pdfCheckBox";
this.pdfCheckBox.Size = new System.Drawing.Size(203, 40);
this.pdfCheckBox.TabIndex = 5;
this.pdfCheckBox.Text = "Открыть pdf";
this.pdfCheckBox.UseVisualStyleBackColor = true;
this.pdfCheckBox.Visible = false;
this.pdfCheckBox.CheckedChanged += new
System.EventHandler(this.PdfCheckBox_CheckedChanged);
//
// textSizeComboBox
//
this.textSizeComboBox.DropDownStyle =
System.Windows.Forms.ComboBoxStyle.DropDownList;
this.textSizeComboBox.Font = new System.Drawing.Font("Microsoft Sans
Serif", 10F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.textSizeComboBox.FormattingEnabled = true;
this.textSizeComboBox.Items.AddRange(new object[] {
"8",
"10",
"11",
"12",
"13",
"14",
"16",
"18",
"20"});
this.textSizeComboBox.Location = new System.Drawing.Point(1114, 556);
this.textSizeComboBox.Name = "textSizeComboBox";
this.textSizeComboBox.Size = new System.Drawing.Size(80, 39);
this.textSizeComboBox.TabIndex = 11;
this.textSizeComboBox.Visible = false;
this.textSizeComboBox.SelectedIndexChanged += new
System.EventHandler(this.TextSizeComboBox_SelectedIndexChanged);
```

```
//
// textSizeLabel
//
this.textSizeLabel.Font = new System.Drawing.Font("Comic Sans MS",
9.4F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.textSizeLabel.Location = new System.Drawing.Point(837, 560);
this.textSizeLabel.Name = "textSizeLabel";
this.textSizeLabel.Size = new System.Drawing.Size(265, 40);
this.textSizeLabel.TabIndex = 12;
this.textSizeLabel.Text = "Размер текста в pdf";
this.textSizeLabel.Visible = false;
//
// createPdfCheckBox
//
this.createPdfCheckBox.AutoSize = true;
this.createPdfCheckBox.Font = new System.Drawing.Font("Comic Sans MS",
9.4F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.createPdfCheckBox.Location = new System.Drawing.Point(343, 560);
this.createPdfCheckBox.Name = "createPdfCheckBox";
this.createPdfCheckBox.Size = new System.Drawing.Size(192, 40);
this.createPdfCheckBox.TabIndex = 13;
this.createPdfCheckBox.Text = "Создать pdf";
this.createPdfCheckBox.UseVisualStyleBackColor = true;
this.createPdfCheckBox.CheckedChanged += new
System.EventHandler(this.CreatePdfCheckBox_CheckedChanged);
//
// waitLabel
//
this.waitLabel.BackColor = System.Drawing.Color.LemonChiffon;
this.waitLabel.Font = new System.Drawing.Font("Comic Sans MS",
13.125F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte)(204)));
this.waitLabel.Location = new System.Drawing.Point(395, 388);
this.waitLabel.Name = "waitLabel";
this.waitLabel.Size = new System.Drawing.Size(472, 163);
this.waitLabel.TabIndex = 14;
this.waitLabel.Text = "Пожалуйста, подождите, пока создается
pdf-файл...";
this.waitLabel.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
this.waitLabel.Visible = false;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(12F, 25F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.BackColor = System.Drawing.Color.Ivory;
this.ClientSize = new System.Drawing.Size(1273, 1029);
```

```
            this.Controls.Add(this.waitLabel);
            this.Controls.Add(this.createPdfCheckBox);
            this.Controls.Add(this.textSizeLabel);
            this.Controls.Add(this.textSizeComboBox);
            this.Controls.Add(this.pdfCheckBox);
            this.Controls.Add(this.htmlCheckBox);
            this.Controls.Add(this.escButton);
            this.Controls.Add(this.seedBox);
            this.Controls.Add(this.seedLabel);
            this.Controls.Add(this.randBox);
            this.Controls.Add(this.infoLabel);
            this.Controls.Add(this.title2);
            this.Controls.Add(this.title1);
            this.Controls.Add(this.genButton);
            this.Controls.Add(this.errorLabel);
            this.Controls.Add(this.numLabel);
            this.Controls.Add(this.numTextBox);
            this.Controls.Add(this.typeComboBox);
            this.Controls.Add(this.difLabel);
            this.Name = "Form1";
            this.Text = "Генератор задач №26 ЕГЭ по информатике";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        private System.Windows.Forms.Label difLabel;
        private System.Windows.Forms.ComboBox typeComboBox;
        private System.Windows.Forms.TextBox numTextBox;
        private System.Windows.Forms.Label numLabel;
        private System.Windows.Forms.Label errorLabel;
        private System.Windows.Forms.Button genButton;
        private System.Windows.Forms.Label title1;
        private System.Windows.Forms.Label title2;
        private System.Windows.Forms.Label infoLabel;
        private System.Windows.Forms.CheckBox randBox;
        private System.Windows.Forms.Label seedLabel;
        private System.Windows.Forms.TextBox seedBox;
        private System.Windows.Forms.Button escButton;
        private System.Windows.Forms.CheckBox htmlCheckBox;
        private System.Windows.Forms.CheckBox pdfCheckBox;
        private System.Windows.Forms.ComboBox textSizeComboBox;
        private System.Windows.Forms.Label textSizeLabel;
        private System.Windows.Forms.CheckBox createPdfCheckBox;
        private System.Windows.Forms.Label waitLabel;
    }
}
```

2.4 HtmlTemplate.txt

```
<!doctype html>
<html>
 <head>
  <meta charset="utf-8">
  <title>Задание №26 ЕГЭ по информатике</title>
  <style type="text/css">
   h1 {
       font-size: 210%;
       font-family: Comic Sans MS, Comic Sans, cursive;
       color: #008000;
   }
   p {
          font-size: 90%;
       font-family: Comic Sans MS, Comic Sans, cursive;
       color: #000000;
       font-weight: 100;
   }
   body {
          padding: 0px 30px;
       background: #FFFFF0;
       border: 3px double black;
       margin: 20px;
   }
   input.colored {
          border-radius: 5px;
       color: #008000;
       background: #FFFFFF;
   }
   th.h {
           background-color: #FFFADD;
   }
   td.w {
           background-color: lightgreen;
   }
   td.l {
           background-color: #FED6BC;
   }
   td.r {
           background-color: #FFFFF0;
   }
   table {
           border-collapse: collapse;
           font-family: Andale Mono, monospace;
   }
  </style>
  <script>
   function showAnswer(ans, id) {
```

```
    var answer = document.getElementById(id);
    answer.innerHTML = ans;
    }
   function hideAnswer(id) {
    var answer = document.getElementById(id);
    answer.innerHTML = "";
   }
  </script>
 </head>
 <body>

  <h1> Задание №26 ЕГЭ по информатике </h1>

</html>
```