**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |  |
| --- | --- | --- |
| СОГЛАСОВАНО  Руководитель,  доцент департамента  программной инженерии факультета компьютерных наук,  кан. тех. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Р.З.Ахметсафина  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия», профессор департамента программной инженерии, кан. тех. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.01-01 12 01-1-ЛУ | | **ИГРА «ПЕРЕМЫЧКИ»**  **Текст программы**  **ЛИСТ УТВЕРЖДЕНИЯ**  RU.17701729.04.01-01 12 01-1-ЛУ | | |
|  |  | |
| Исполнитель  студент группы БПИ197  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Джапаров Э.М. /  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. | |
|  | | |
|  | |  |

**Москва 2020**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| УТВЕРЖДЕН  RU.17701729.04.01-01 12 01-1-ЛУ |  | |  | |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** | RU.17701729.04.01-01 12 01-1-ЛУ | | **ИГРА «ПЕРЕМЫЧКИ»**  **Пояснительная записка**  **RU.17701729.04.01-01 12 01-1-ЛУ**  **Листов 90** | | | | |
|  | |  | | |
|  | | |
|  | | | | |
|  | | | |  |

**Москва 2020**

СОДЕРЖАНИЕ

[1. ВВЕДЕНИЕ 3](#_Toc41336686)

[1.1. Модуль App 4](#_Toc41336687)

[1.1.1. MainForm.cs 4](#_Toc41336688)

[1.1.2. PauzeForm.cs 11](#_Toc41336689)

[1.1.3. SaveResultForm.cs 13](#_Toc41336690)

[1.1.4. GameField.cs 16](#_Toc41336691)

[1.1.5. InfoForSave.cs 73](#_Toc41336692)

[1.2. BridgesExceptionLib.cs 78](#_Toc41336693)

[1.2.1. BridgesException.cs 78](#_Toc41336694)

[1.3. NodeLib 78](#_Toc41336695)

[1.3.1. Node 78](#_Toc41336696)

[1.4. ParserLib 83](#_Toc41336697)

[1.4.1. Parser 83](#_Toc41336698)

# ВВЕДЕНИЕ

Программа состоит из 8 неавтосгенерированных классов.

## Модуль App

### MainForm.cs

using System;

using System.Drawing;

using System.Windows.Forms;

using ParserLib;

namespace App

{

public partial class MainForm : Form

{

// Директория файлов со статистикой о прорешенных или просмотренных заданий.

const string pathEasyLevelStatistics = "resources/EasyLevelStatistics.khan";

const string pathMediumLevelStatistics = "resources/MediumLevelStatistics.khan";

const string pathHardLevelStatistics = "resources/HardLevelStatistics.khan";

public MainForm()

{

InitializeComponent();

}

/// <summary>

/// Обработчик события загрузки формы.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void MainFormLoad(object sender, EventArgs e)

{

deleteRecordButton.Visible = false;

clearAllRecords.Visible = false;

}

/// <summary>

/// Обработчик события нажатия на кнопку "Начать игру".

/// Создается объект второй формы и запускается процесс игры.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StartGameButtonClick(object sender, EventArgs e)

{

if (choisedLevel != diffLevelComboBox.Items[0].ToString() && choisedLevel != diffLevelComboBox.Items[1].ToString() && choisedLevel != diffLevelComboBox.Items[2].ToString())

{

MessageBox.Show("Выберите уровень сложности для начала игры!", "Внимание!");

diffLevelComboBox.Select();

return;

}

this.Hide();

if (diffLevelComboBox.SelectedIndex == 0)

{

GameField gameField = new GameField(0);

gameField.ShowDialog();

}

if (diffLevelComboBox.SelectedIndex == 1)

{

GameField gameField = new GameField(1);

gameField.ShowDialog();

}

if (diffLevelComboBox.SelectedIndex == 2)

{

GameField gameField = new GameField(2);

gameField.ShowDialog();

}

this.Show();

}

/// <summary>

/// Обработчик события нажатия на кнопку "Выход".

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ExitButtonClick(object sender, EventArgs e)

{

this.Close();

}

// Выбранный пользователем уровень.

string choisedLevel;

/// <summary>

/// Обработчик выбора уровня для начала игры.

/// Инициализируется переменная со значением выбранного уровня.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void DiffLevelComboBoxSelectedIndexChanged(object sender, EventArgs e)

{

if (diffLevelComboBox.SelectedIndex == 0)

{

diffLevelComboBox.BackColor = Color.DeepSkyBlue;

}

if (diffLevelComboBox.SelectedIndex == 1)

{

diffLevelComboBox.BackColor = Color.RoyalBlue;

}

if (diffLevelComboBox.SelectedIndex == 2)

{

diffLevelComboBox.BackColor = Color.DarkBlue;

}

startGameButton.Select();

choisedLevel = (string)diffLevelComboBox.SelectedItem;

}

/// <summary>

/// Обработчик нажатия на кнопку "Правила".

/// Пользователю предлагается изучить правила и пройти обучение.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void RulesButtonClick(object sender, EventArgs e)

{

recordsPanel.Visible = false;

gameInfoPanel.Visible = false;

rulesPanel.Visible = true;

}

/// <summary>

/// Обработчик нажатия кнопки "Скрыть", когда открыта часть окна с Правилами.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HideRulesButtonClick(object sender, EventArgs e)

{

rulesPanel.Visible = false;

}

// Массив строк из файла статистики о прорешенных пользователями заданиях.

string[] dataFromStatisticsFile;

/// <summary>

/// Обработчик нажатия на кнопку "Статистика".

/// Пользователю открывается статистика о всех прорешенных заданиях на выбранном уровне.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void StatisticsButtonClick(object sender, EventArgs e)

{

MessageBox.Show("Выберите уровень для которого хотите посмотреть все рекорды.", "Выберите уровень", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

chooseLevelComboBox.Select();

gameInfoPanel.Visible = false;

rulesPanel.Visible = false;

recordsPanel.Visible = true;

}

/// <summary>

/// Обработчик нажатия кнопки "Скрыть", когда открыта часть окна описания игры.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HideGameInfoButtonClick(object sender, EventArgs e)

{

hideGameInfoButton.Parent.Visible = false;

}

/// <summary>

/// Обработчик события закрытия игры.

/// У пользователя уточняется, уверен ли он покинуть игру ?

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void MainFormFormClosing(object sender, FormClosingEventArgs e)

{

if (MessageBox.Show("Вы хотите выйти?", "Выход", MessageBoxButtons.YesNo,

MessageBoxIcon.Question) == DialogResult.No)

{

e.Cancel = true;

}

}

/// <summary>

/// Обработчик нажатия кнопки "Скрыть", когда открыта часть окна о статистике.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HideStatisticsButtonClick(object sender, EventArgs e)

{

recordsPanel.Visible = false;

}

/// <summary>

/// Обработчик события нажатия кнопки "Об игре".

/// Пользователю откроется описание продукта.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void GameInfoButtonClick(object sender, EventArgs e)

{

rulesPanel.Visible = false;

recordsPanel.Visible = false;

gameInfoPanel.Visible = true;

}

// Информация о строках элемента DataGridView.

string[] dataFromDataGrid;

/// <summary>

/// Обработчик нажатия на кнопку "Удалить результат" в окне статистики.

/// Удаляются выбранные пользователем строки и из таблицы, и из файла.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void DeleteRecordsButtonClick(object sender, EventArgs e)

{

if (dataGridView1.SelectedRows.Count == 0)

{

MessageBox.Show("Для удаления рекорда выберите соответствующую строку!", "Удаление рекорда.", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

dataGridView1.Rows.Remove(dataGridView1.SelectedRows[0]);

dataFromDataGrid = new string[dataGridView1.Rows.Count];

for (int i = 0; i < dataGridView1.Rows.Count; i++)

{

for (int j = 0; j < dataGridView1.ColumnCount; j++)

{

dataFromDataGrid[i] += j != dataGridView1.ColumnCount - 1 ? dataGridView1[j, i].Value.ToString() + ";" : dataGridView1[j, i].Value.ToString();

}

}

if (chooseLevelComboBox.SelectedIndex == 0)

Parser.WriteLines(pathEasyLevelStatistics, dataFromDataGrid);

if (chooseLevelComboBox.SelectedIndex == 1)

Parser.WriteLines(pathMediumLevelStatistics, dataFromDataGrid);

if (chooseLevelComboBox.SelectedIndex == 2)

Parser.WriteLines(pathHardLevelStatistics, dataFromDataGrid);

}

/// <summary>

/// Обработчик нажатия на кнопку "Очистить статистику" в окне статистики.

/// Удаляятся вся информация и из таблицы, и из файла.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ClearAllRecordsClick(object sender, EventArgs e)

{

if (chooseLevelComboBox.SelectedItem == null)

{

MessageBox.Show("Для того, чтобы очистить рекорды, выберите уровень!", "Очистка всех рекордов.", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

dataGridView1.Rows.Clear();

if (chooseLevelComboBox.SelectedIndex == 0)

Parser.DeleteFile(pathEasyLevelStatistics);

if (chooseLevelComboBox.SelectedIndex == 1)

Parser.DeleteFile(pathMediumLevelStatistics);

if (chooseLevelComboBox.SelectedIndex == 2)

Parser.DeleteFile(pathHardLevelStatistics);

}

/// <summary>

/// Обработчик события выбора уровня для выведения статистики.

/// При срабатывании события в DataGridView выводится статистика

/// о решенных пользователями полях на выбранном уровне.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void СhooseLevelComboBoxSelectedIndexChanged(object sender, EventArgs e)

{

if (chooseLevelComboBox.SelectedItem == null)

{

MessageBox.Show("Выберите уровень для которого хотите посмотреть все рекорды.", "Выберите уровень", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

return;

}

deleteRecordButton.Visible = true;

clearAllRecords.Visible = true;

dataGridView1.Rows.Clear();

if (chooseLevelComboBox.SelectedIndex == 0 && Parser.FileExists(pathEasyLevelStatistics))

{

dataFromStatisticsFile = Parser.ReadAll(pathEasyLevelStatistics);

if (dataFromStatisticsFile.Length > 0)

{

for (int i = 0; i < dataFromStatisticsFile.Length; i++)

{

dataGridView1.Rows.Add(dataFromStatisticsFile[i].Split(';'));

}

return;

}

}

if (chooseLevelComboBox.SelectedIndex == 1 && Parser.FileExists(pathMediumLevelStatistics))

{

dataFromStatisticsFile = Parser.ReadAll(pathMediumLevelStatistics);

if (dataFromStatisticsFile.Length > 0)

{

for (int i = 0; i < dataFromStatisticsFile.Length; i++)

{

dataGridView1.Rows.Add(dataFromStatisticsFile[i].Split(';'));

}

return;

}

}

if (chooseLevelComboBox.SelectedIndex == 2 && Parser.FileExists(pathHardLevelStatistics))

{

dataFromStatisticsFile = Parser.ReadAll(pathHardLevelStatistics);

if (dataFromStatisticsFile.Length > 0)

{

for (int i = 0; i < dataFromStatisticsFile.Length; i++)

{

dataGridView1.Rows.Add(dataFromStatisticsFile[i].Split(';'));

}

return;

}

}

MessageBox.Show("Данные не найдены!", "Нет данных.", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

}

### PauzeForm.cs

using System;

using System.Windows.Forms;

namespace App

{

public partial class PauzeForm : Form

{

// Информация о том, была ли нажата кнопка "продолжить".

bool continueClicked;

// Информация о том, была ли нажата кнопка "Как играть ?".

bool helpClicked;

// Информация о том, была ли нажата кнопка "В главное меню".

bool exitClicked;

/// <summary>

/// Свойство с доступом к полю exitClicked.

/// </summary>

public bool ExitClicked { get => exitClicked; set => exitClicked = value; }

/// <summary>

/// Свойство с доступом к полю helpClicked.

/// </summary>

public bool HelpClicked { get => helpClicked; set => helpClicked = value; }

/// <summary>

/// Свойство с доступом к полю continueClicked.

/// </summary>

public bool ContinueClicked { get => continueClicked; set => continueClicked = value; }

/// <summary>

/// Конструктор умолчания для данного класса

/// </summary>

public PauzeForm()

{

InitializeComponent();

}

/// <summary>

/// Обработчик события нажатия на кнопку "Продолжить".

/// Продолжается процесс игры.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void СontinueButtonClick(object sender, EventArgs e)

{

continueClicked = true;

this.Close();

}

/// <summary>

/// Обработчик события нажатия на кнопку "В главное меню".

/// Закрывается окно с игровым полем и открывается главное меню.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ExitButtonClick(object sender, EventArgs e)

{

exitClicked = true;

this.Close();

}

/// <summary>

/// Обработчик события нажатия на кнопку "Как играть?".

/// Открывается краткий список правил.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HelpButtonClick(object sender, EventArgs e)

{

MessageBox.Show("Пользователь должен решать игровые поля следующим образом:\n1) Нужно соединить перемычками каждый кружочек с соседними кружочками;\n2) Количество перемычек, проведенных от каждого кружочка, должно быть равно\nчисловому значению, указанному внутри кружочка;\n3) Проведенные перемычки не должны пересекаться с другими перемычками.","Правила игры",MessageBoxButtons.OK,MessageBoxIcon.Asterisk);

}

}

}

### SaveResultForm.cs

using System;

using System.Windows.Forms;

namespace App

{

public partial class SaveResultForm : Form

{

public SaveResultForm()

{

InitializeComponent();

}

/// <summary>

/// Имя пользователя для сохранения результата решения текущего игрового поля.

/// </summary>

string nameForStatistics;

/// <summary>

/// Свойство с доступом к полю nameForStatistics. Возвразщает имя пользователя.

/// </summary>

public string NameForStatistics { get => nameForStatistics; set => nameForStatistics = value; }

/// <summary>

/// Обработчик события изменения текста в TextBox.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void InputNameTextBoxTextChanged(object sender, EventArgs e)

{

CancelClicked = false;

}

/// <summary>

/// Метод проверки на то, вводится ли корректное значение для сохранения результата.

/// </summary>

/// <param name="str"> Имя вводимое пользователем для сохранения результата.</param>

/// <returns></returns>

bool NameIsCorrect(string str)

{

char[] letters = str.ToCharArray();

for(int i = 0;i<letters.Length;i++)

{

if(!(letters[i]>='a'&&letters[i]<='z') && !(letters[i]>='A' && letters[i]<='Z') && !(letters[i]>='0'&&letters[i]<='9') && letters[i] != ' ' && letters[i] != '\_')

{

return false;

}

}

return true;

}

/// <summary>

/// Автосвойство, содержащее информацию о том, была ли нажата кнопка "Отмена".

/// </summary>

public bool CancelClicked { set; get; }

/// <summary>

/// Обработчик события нажатия на кнопку "Готово".

/// Введенное имя сохраняется в файл статистики и затем выводится в DataGridView.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void InputButtonClick(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(inputNameTextBox.Text.Trim()) || !NameIsCorrect(inputNameTextBox.Text.Trim()))

{

MessageBox.Show("Введите корректное имя!\nИмя должно состоять строго из латинских символов.\nРегистр не важен.",

"Некорректный ввод!", MessageBoxButtons.OK, MessageBoxIcon.Error);

inputNameTextBox.Clear();

inputNameTextBox.Select();

return;

}

nameForStatistics = inputNameTextBox.Text;

this.Close();

}

/// <summary>

/// Обработчик события нажатия на клавишу Enter.

/// Введенное имя сохраняется в файл статистики и затем выводится в DataGridView.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void InputNameTextBoxKeyDown(object sender, KeyEventArgs e)

{

if(e.KeyCode == Keys.Enter)

{

e.SuppressKeyPress = true;

if (string.IsNullOrEmpty(inputNameTextBox.Text.Trim()) || !NameIsCorrect(inputNameTextBox.Text.Trim()))

{

MessageBox.Show("Введите корректное имя!\nИмя должно состоять строго из латинских символов.\nРегистр не важен.",

"Некорректный ввод!", MessageBoxButtons.OK, MessageBoxIcon.Error);

inputNameTextBox.Clear();

inputNameTextBox.Select();

return;

}

nameForStatistics = inputNameTextBox.Text;

this.Close();

}

}

/// <summary>

/// Обработчик события нажатия на кнопку "Отмена".

/// Отменяется сохранение данных.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void CancelButtonClick(object sender, EventArgs e)

{

CancelClicked = true;

this.Close();

}

}

}

### GameField.cs

using System;

using System.Drawing;

using System.Windows.Forms;

using ParserLib;

using System.Xml.Serialization;

using System.IO;

using NodeLib;

namespace App

{

public partial class GameField : Form

{

// Директория файла с заданиями на текущем уровне.

string pathCurrentLevel;

// Директория файлов с заданиями и решениями.

const string pathEasyLevel = "resources/EasyLevel.khan";

const string pathMediumLevel = "resources/MediumLevel.khan";

const string pathHardLevel = "resources/HardLevel.khan";

// Директория файлов со статистикой пользователей на соответствующих уровнях.

const string pathEasyLevelStatistics = "resources/EasyLevelStatistics.khan";

const string pathMediumLevelStatistics = "resources/MediumLevelStatistics.khan";

const string pathHardLevelStatistics = "resources/HardLevelStatistics.khan";

static Random rnd = new Random();

// Уровень текущей игры.

private int level;

// Узлы, которые ассоциируются с кнопками.

Node[,] nodes;

// Массив кнопок, выведенный на экран.

Button[,] buttons;

// Размерность текущего поля.

int fieldSize;

/// <summary>

/// Конструктор умолчания для данной формы.

/// </summary>

public GameField()

{

InitializeComponent();

}

/// <summary>

/// Конструктор класса GameField с параметром об уровне.

/// </summary>

/// <param name="level"> Уровень игры: 0 - легкий, 1 - средний, 2 - тяжелый.</param>

public GameField(int level)

{

InitializeComponent();

this.level = level;

}

// Поярдковый номер поля в таблице.

int numOfField;

/// <summary>

/// Обработчик загрузки формы.

/// Здесь строится графическая часть игрового поля и его программная часть.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void GameFieldLoad(object sender, EventArgs e)

{

if (level == 0)

helpButton.Visible = false;

if(level == 1)

helpButton.Visible = true;

if (level == 2)

helpButton.Visible = false;

string levelName;

pathCurrentLevel = level == 0 ? pathEasyLevel : level == 1 ? pathMediumLevel : pathHardLevel;

levelName = level == 0 ? "Легкий" : level == 1 ? "Средний" : "Сложный";

// Чтение файла.

if (!File.Exists(pathCurrentLevel))

{

Parser.WriteText(pathCurrentLevel, levelName + ";" + "Решения" + Environment.NewLine);

Parser.ReadFile(pathCurrentLevel);

GenerateNewField(pathCurrentLevel);

}

else

Parser.ReadFile(pathCurrentLevel);

getFocusLabel.Select();

//<b> Для десериализации

InfoForSave ifr = null;

if (level == 0 && File.Exists(pathSavedGameFieldEasyLevel))

{

ifr = LoadSavedGame(pathSavedGameFieldEasyLevel);

}

if (level == 1 && File.Exists(pathSavedGameFieldMediumLevel))

{

ifr = LoadSavedGame(pathSavedGameFieldMediumLevel);

}

if (level == 2 && File.Exists(pathSavedGameFieldHardLevel))

{

ifr = LoadSavedGame(pathSavedGameFieldHardLevel);

}

//<e>

// Если есть сохраненный процесс на данном уровне.

if (ifr != null && ifr.Level == level)

{

RunSavedGame(ifr);

}

// В случае если нет сохраненного процесса.

else

{

FieldLoad();

timer.Enabled = true;

timer.Start();

}

}

/// <summary>

/// Запускает сохраненный процесс игры, если он есть. Строит графическую часть игры и инициализирует все данные.

/// </summary>

/// <param name="ifr"></param>

private void RunSavedGame(InfoForSave ifr)

{

secs = ifr.Secs;

mins = ifr.Mins;

hours = ifr.Hours;

// Добавить этот кусок кода в другие уровни

isFinished = ifr.Finished;

if (isFinished)

{

if (hours < 10)

timerLabel.Text = $"0{hours}";

else

timerLabel.Text = $"{hours}";

if (mins < 10)

timerLabel.Text += $":0{mins}";

else

timerLabel.Text += $":{mins}";

if (secs < 10)

timerLabel.Text += $":0{secs}";

else

timerLabel.Text += $":{secs}";

MessageBox.Show("Игровое поле уже было успешно решено ранее!", "Внимание!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

}

else

{

timer.Enabled = true;

timer.Start();

}

fieldSize = ifr.FieldSize;

nodes = ifr.FromNodeOfNodeToDoubleNode();

pbIsSet = ifr.FromPbIsSetOfPbIsSetToDoublePbIsSet();

connectedNodes = ifr.FromStrOfStrToDoubleStr();

helpButton.Visible = ifr.HelpButtonVisible;

pb = new PictureBox[pbIsSet.GetLength(0), pbIsSet.GetLength(1)];

buttons = new Button[nodes.GetLength(0), nodes.GetLength(1)];

id = ifr.Id;

idLabel.Text = "ID\_\_" + id.ToString();

int levelLength = Parser.LevelColumnLength(0);

// Проверка на то, имеются ли поля до данного поля по порядковому номеру в базе данных.

if (levelLength > NumOfField())

{

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

{

selectFieldComboBox.Items.Add(i);

}

}

// Можно вынести, как const в область класса.

int x = 200;

int y = 80;

// Массив из троек, то есть из строк поля.

//???

//string[] strdataRows = new string[3];

//string[] strdata = new string[3];

for (int i = 0; i < buttons.GetLength(0); i++)

{

// Массив из элементов заданной строки.

// ????

//strdata = strdataRows[i].Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

for (int j = 0; j < buttons.GetLength(1); j++)

{

buttons[i, j] = new Button();

buttons[i, j].BackColor = Color.MediumPurple;

buttons[i, j].ForeColor = Color.White;

buttons[i, j].Font = new Font("Cambria", 16, FontStyle.Italic);

buttons[i, j].Text = nodes[i, j].Weight.ToString();

buttons[i, j].Size = new Size(50, 50);

buttons[i, j].Location = new Point(x, y);

buttons[i, j].Click += GameButtonClick;

buttons[i, j].MouseEnter += ButtonMouseEnter;

buttons[i, j].MouseLeave += ButtonMouseLeave;

// В имени кнопки(узла) будет содержаться информация о его положении.

buttons[i, j].Name = $"{i};{j}";

System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Drawing2D.GraphicsPath();

myPath.AddEllipse(0, 0, buttons[i, j].Width, buttons[i, j].Height);

Region myRegion = new Region(myPath);

buttons[i, j].Region = myRegion;

buttons[i, j].FlatStyle = FlatStyle.Flat;

buttons[i, j].FlatAppearance.BorderSize = 0;

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

x += 100;

}

x = 200;

y += 100;

}

// Заполнить у каждого элемента nodes[,] поле nodesConnected.

// содержит позиции соседей текущего элемента nodes[i,j]

string[] currentNeighbours = null;

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

if (connectedNodes[i, j] != null)

{

currentNeighbours = connectedNodes[i, j].Split(',');

FillConnectedNodesList(nodes[i, j], currentNeighbours);

if (nodes[i, j].ConnectedNodesCount() > 0 && nodes[i, j].Weight > nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Goldenrod;

}

if (nodes[i, j].Weight == nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Green;

}

if (nodes[i, j].Weight < nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.DarkRed;

}

if (nodes[i, j].ConnectedNodesCount() == 0)

{

buttons[i, j].BackColor = Color.MediumPurple;

}

}

}

// Провести все связи.

SetPictureBoxes();

if (!ConnectionsExist())

helpButton.Visible = true;

}

/// <summary>

/// Метод, в котором выполняется графическое и программное построение игрового поля, если никакого сохраненного процесса нет.

/// </summary>

private void FieldLoad()

{

if (level == 1)

helpButton.Visible = true;

//numOfField = CSVParser.LevelColumnLength(level) - 1;

numOfField = 1;

fieldSize = Parser.GetFieldLength(numOfField, 0);

pb = new PictureBox[2 \* fieldSize - 1, 2 \* fieldSize - 1];

nodes = new Node[fieldSize, fieldSize];

buttons = new Button[fieldSize, fieldSize];

int x = 200;

int y = 80;

// Массив из троек, то есть из строк поля.

string[] strdataRows = new string[fieldSize];

#region Просто красивое выведение ID в форме.

int countOfCiphers = 0;

int copyNumOfField = 0;

copyNumOfField = numOfField;

do

{

copyNumOfField /= 10;

countOfCiphers++;

} while (copyNumOfField >= 1);

// Каждому уровню в числе id соответствует перфая цифра в числе id. 1- легкий, 2 - средний, 3 - сложный.

// Легкому уровню соответствует 0 в числе id.

id = (level + 1) \* (int)Math.Pow(10, countOfCiphers) + numOfField;

idLabel.Text = "ID\_\_" + id.ToString();

selectFieldComboBox.Items.Add(1);

#endregion

strdataRows = Parser.GetField(numOfField, 0).Split(new[] { ',' }, StringSplitOptions.RemoveEmptyEntries);

string[] strdata = new string[fieldSize];

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

{

// Массив из элементов заданной строки.

strdata = strdataRows[i].Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

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

{

buttons[i, j] = new Button();

buttons[i, j].BackColor = Color.MediumPurple;

buttons[i, j].ForeColor = Color.White;

buttons[i, j].Font = new Font("Cambria", 16, FontStyle.Italic);

buttons[i, j].Text = strdata[j];

buttons[i, j].Size = new Size(50, 50);

buttons[i, j].Location = new Point(x, y);

buttons[i, j].Click += GameButtonClick;

buttons[i, j].MouseEnter += ButtonMouseEnter;

buttons[i, j].MouseLeave += ButtonMouseLeave;

System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Drawing2D.GraphicsPath();

myPath.AddEllipse(0, 0, buttons[i, j].Width, buttons[i, j].Height);

Region myRegion = new Region(myPath);

buttons[i, j].Region = myRegion;

buttons[i, j].FlatStyle = FlatStyle.Flat;

buttons[i, j].FlatAppearance.BorderSize = 0;

// В имени кнопки(узла) будет содержаться информация о его положении.

buttons[i, j].Name = $"{i};{j}";

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

x += 100;

}

x = 200;

y += 100;

}

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

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

{

nodes[i, j] = new Node(int.Parse(buttons[i, j].Text), i, j);

}

if (level == 0)

SetFirstHelp();

if (!ConnectionsExist()&&level !=2)

helpButton.Visible = true;

}

/// <summary>

/// Расстановка первоначальных связей на пустом поле в качестве подсказки для пользователя.

/// </summary>

void SetFirstHelp()

{

string solution;

solution = Parser.GetField(NumOfField(), 1);

string[,] solutionDouble = FromStringToDoubleArr(solution);

int size = solutionDouble.GetLength(0);

pbIsSet = new bool[size, size];

// Добавление в список tempNode рандоных элементов Node.

// Сколько рандомных мостов нужно добавить.

int countRnd = rnd.Next(4, 8);

int x = 0;

int y = 0;

bool res = false;

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

{

do

{

x = rnd.Next(0, size);

y = rnd.Next(0, size);

if (!pbIsSet[x, y] && solutionDouble[x, y] != "P")

{

pbIsSet[x, y] = true;

res = true;

}

else

res = false;

} while (!res);

}

for (int i = 0; i < pbIsSet.GetLength(0); i++)

{

for (int j = 0; j < pbIsSet.GetLength(1); j++)

{

if (pbIsSet[i, j])

{

if (solutionDouble[i, j] == "G")

{

DrawSavedLines(i, j, "g");

AddToNodesList(ref nodes[i / 2, (j - 1) / 2], ref nodes[i / 2, (j + 1) / 2]);

continue;

}

if (solutionDouble[i, j] == "V")

{

DrawSavedLines(i, j, "v");

AddToNodesList(ref nodes[(i - 1) / 2, j / 2], ref nodes[(i + 1) / 2, j / 2]);

continue;

}

if (solutionDouble[i, j] == "LVPN")

{

DrawSavedLines(i, j, "lvpn");

AddToNodesList(ref nodes[(i - 1) / 2, (j - 1) / 2], ref nodes[(i + 1) / 2, (j + 1) / 2]);

continue;

}

if (solutionDouble[i, j] == "PVLN")

{

DrawSavedLines(i, j, "pvln");

AddToNodesList(ref nodes[(i - 1) / 2, (j + 1) / 2], ref nodes[(i + 1) / 2, (j - 1) / 2]);

continue;

}

}

}

}

// Установка цвета кнопок.

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

if (nodes[i, j].ConnectedNodesCount() > 0 && nodes[i, j].Weight > nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Goldenrod;

}

if (nodes[i, j].Weight == nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Green;

}

if (nodes[i, j].Weight < nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.DarkRed;

}

if (nodes[i, j].ConnectedNodesCount() == 0)

{

buttons[i, j].BackColor = Color.MediumPurple;

}

}

}

/// <summary>

/// Получить из информации в виде строки ее же в виде двумерного массива.

/// </summary>

/// <param name="info"> Данные которые конвертируются в двумерный массив</param>

/// <returns> Двумерный массив данных.</returns>

string[,] FromStringToDoubleArr(string info)

{

int size = info.Split(',').Length;

string[] currentRow;

string[] rows = info.Split(',');

string[,] doubleInfo = new string[size, size];

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

{

currentRow = rows[i].Split(' ');

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

{

doubleInfo[i, j] = currentRow[j];

}

}

return doubleInfo;

}

/// <summary>

/// Генерируется новое поле и информация о нем записывается в файл с заданиями.

/// В этом методе сначала рандомно генерируется по W-алгоритму игровое поле с гарантированным решением. Потом это поле добавляется

/// в базу данных.

/// </summary>

/// <param name="level"></param>

void GenerateNewField(string path)

{

#region Генерация нового игрового поля, который надо будет добавить в базу данных.

int fieldSize = 0;

//fieldSize = level == 0 ? rnd.Next(3, 5) : level == 1 ? rnd.Next(4, 6) : rnd.Next(5, 7);

fieldSize = level == 0 ? 4 : level == 1 ? 5 : 6;

// Двумерный массив для создания нового игрового поля.

Node[,] nodesForGen = new Node[fieldSize, fieldSize];

// Инициалазация каждого элемента в массиве.

for (int i = 0; i < nodesForGen.GetLength(0); i++)

for (int j = 0; j < nodesForGen.GetLength(1); j++)

{

nodesForGen[i, j] = new Node(i, j);

}

// Заполнение элементов массива и рандомное соединение с соседями. Должны охватиться все возможные связи. То есть должно

// быть проведено ровно (2n-1)^2 - n^2 связей для поля n x n.

for (int i = 0; i < nodesForGen.GetLength(0); i++)

for (int j = 0; j < nodesForGen.GetLength(1); j++)

{

// Если вершина не в правом нижнем углу.

if (i != nodesForGen.GetLength(0) - 1 || j != nodesForGen.GetLength(1) - 1)

{

// Если текущий узел не в последнем столбце И не в последней строке, то мы связываем его с соседями по принципу ПРАВО ВНИЗ ЛВПНилиПВЛН

if (0 <= i && i < nodesForGen.GetLength(0) - 1 && 0 <= j && j < nodesForGen.GetLength(1) - 1)

{

// Соединение с вертикальным соседом.

AddToNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

// Соединение в горизонтальным соседом.

AddToNodesList(ref nodesForGen[i, j], ref nodesForGen[i, j + 1]);

// Соединение с соседом по-диагонали.

if (rnd.Next(0, 2) == 0)

AddToNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j + 1]);

// Соедениние соседей по вертикали и горизонтали для данного элемента nodesForGen[i,j]

else

AddToNodesList(ref nodesForGen[i + 1, j], ref nodesForGen[i, j + 1]);

}

else

{

// Элемент в последней строке.

if (i == nodesForGen.GetLength(1) - 1)

{

AddToNodesList(ref nodesForGen[i, j], ref nodesForGen[i, j + 1]);

}

// Элемент в последнем столбце.

else

{

AddToNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

}

}

}

}

int tempRnd;

int tempConnectedNodesCount;

bool tempBool;

// Рандомное удаление связей, но по принципу, что у каждого узла(острова, элемента, вершины) вес должен быть как минимум 1.

// Удаление связей по правилу П,В,ПВЛН или ЛВПН. Право вниз право-вниз

for (int i = 0; i < nodesForGen.GetLength(0); i++)

for (int j = 0; j < nodesForGen.GetLength(1); j++)

{

tempConnectedNodesCount = nodesForGen[i, j].ConnectedNodesCount();

// Если вершина не в правом нижнем углу.

if (i != nodesForGen.GetLength(0) - 1 || j != nodesForGen.GetLength(1) - 1)

{

// Если текущий узел не в последнем столбце И не в последней строке, то мы связываем его с соседями по принципу ПРАВО ВНИЗ ЛВПНилиПВЛН

if (0 <= i && i < nodesForGen.GetLength(0) - 1 && 0 <= j && j < nodesForGen.GetLength(1) - 1)

{

// Содержит информацию, есть ли проведенная побочная ПВЛН диагональ.

tempBool = nodesForGen[i + 1, j].ElemContains(nodesForGen[i, j + 1]);

if (tempConnectedNodesCount > 2 && tempBool)

{

// Удаление соединения с горизонтальным соседом.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i, j + 1], ref nodesForGen[i, j]);

// Удаление соединения с вертикальным соседом.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

// Удаление соединения между горизонтальным и вертикальным соседями данного элемента.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i + 1, j], ref nodesForGen[i, j + 1]);

continue;

}

if (tempConnectedNodesCount == 2 && tempBool)

{

// Удаление соединения между горизонтальным и вертикальным соседями данного элемента.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i + 1, j], ref nodesForGen[i, j + 1]);

// Вероятность удаления горизнтального или вертикального соседей.

tempRnd = rnd.Next(0, 4);

if (tempRnd == 0)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

if (tempRnd == 2)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i, j + 1]);

continue;

}

if (tempConnectedNodesCount > 3 && !tempBool)

{

// Удаление соединения с горизонтальным соседом.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i, j + 1], ref nodesForGen[i, j]);

// Удаление соединения с вертикальным соседом.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

// Удаление соединения ЛВПН.

if (rnd.Next(0, 2) == 0)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j + 1]);

continue;

}

if (tempConnectedNodesCount == 3 && !tempBool)

{

tempRnd = rnd.Next(0, 7);

// Удаляются g и v связи от текущего элемента.

if (tempRnd == 0)

{

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i, j + 1]);

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i + 1, j]);

}

// Удаляются g и lvpn связи от текущего элемента.

if (tempRnd == 1)

{

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i, j + 1]);

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i + 1, j + 1]);

}

// Удаляются v и lvpn связи от текущего элемента.

if (tempRnd == 2)

{

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i + 1, j]);

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i + 1, j + 1]);

}

// Удаляется v связь от текущего элемента.

if (tempRnd == 3)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

// Удаляется g связь от текущего элемента.

if (tempRnd == 4)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i, j + 1]);

// Вероятность удаления ЛВПН.

if (tempRnd == 5)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j + 1]);

continue;

}

}

else

{

// Элемент в последней строке.

if (i == nodesForGen.GetLength(0) - 1)

{

// Если текущий элемент предпоследний по счету в последней строке.

if (rnd.Next(0, 2) == 0 && j == nodesForGen.GetLength(1) - 2 && nodesForGen[i, j].ConnectedNodesCount() > 1 && nodesForGen[nodesForGen.GetLength(0) - 1, nodesForGen.GetLength(1) - 1].ConnectedNodesCount() > 1)

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i, j + 1]);

if (rnd.Next(0, 2) == 0 && j != nodesForGen.GetLength(1) - 2 && nodesForGen[i, j].ConnectedNodesCount() > 1)

RemoveFromNodesList(nodesForGen[i, j], nodesForGen[i, j + 1]);

}

// Элемент в последнем столбце.

else

{

if (rnd.Next(0, 2) == 0 && nodesForGen[i, j].ConnectedNodesCount() > 1)

RemoveFromNodesList(ref nodesForGen[i, j], ref nodesForGen[i + 1, j]);

}

}

}

}

string genFieldInfo = "";

// Заполнение поля Weight у каждого элемента массива nodesForGen

for (int i = 0; i < nodesForGen.GetLength(0); i++)

{

for (int j = 0; j < nodesForGen.GetLength(1); j++)

{

nodesForGen[i, j].Weight = nodesForGen[i, j].ConnectedNodesCount();

genFieldInfo += j != nodesForGen.GetLength(1) - 1 ? nodesForGen[i, j].Weight.ToString() + " " : nodesForGen[i, j].Weight.ToString();

}

genFieldInfo += i != nodesForGen.GetLength(1) - 1 ? "," : "";

}

#endregion

#region Заполнение массива наличия связей

string solutions = "";

int size = 2 \* fieldSize - 1;

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

{

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

{

if (i % 2 != 0 || j % 2 != 0)

{

// Горизонтальный или вертикальный мост.

if ((i + j) % 2 == 1)

{

// Горизонтальный мост.

if (j % 2 == 1 && nodesForGen[i / 2, (j - 1) / 2].ElemContains(nodesForGen[i / 2, (j + 1) / 2]))

{

solutions += j != size - 1 ? "G" + " " : "G";

continue;

}

// Вертикальный мост

if (i % 2 == 1 && nodesForGen[(i - 1) / 2, j / 2].ElemContains(nodesForGen[(i + 1) / 2, j / 2]))

{

solutions += j != size - 1 ? "V" + " " : "V";

continue;

}

}

else

{

// ЛВПН мост.

if (nodesForGen[(i - 1) / 2, (j - 1) / 2].ElemContains(nodesForGen[(i - 1) / 2 + 1, (j - 1) / 2 + 1]))

{

solutions += j != size - 1 ? "LVPN" + " " : "LVPN";

continue;

}

// ПВЛН мост.

if (nodesForGen[(i - 1) / 2, (j + 1) / 2].ElemContains(nodesForGen[(i + 1) / 2, (j - 1) / 2]))

{

solutions += j != size - 1 ? "PVLN" + " " : "PVLN";

continue;

}

}

}

solutions += j != size - 1 ? "P" + " " : "P";

}

solutions += i != size - 1 ? "," : "";

}

#endregion

// Последний этап генерации нового поля.

// Запись это поля в базу данных.

Parser.AddField(path, genFieldInfo, 0);

Parser.AddField(path, solutions, 1);

Parser.ReadFile(path);

}

/// <summary>

/// Переход к другому полю на текущем уровне.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void ToNewFieldButtonClick(object sender, EventArgs e)

{

//// Нужно сериализовать!

if (level == 1)

helpButton.Visible = true;

//// Добавить лист idList из интов для того, чтобы хранить информацию обо всех уже пройденных уровнях.

string levelName = level == 0 ? "Легкий" : level == 1 ? "Средний" : "Сложный";

string path = pathCurrentLevel;

isFinished = false;

timer.Stop();

timer.Enabled = false;

ResetTimer();

ClearAllGameField();

getFocusLabel.Select();

//

// Сделать так, чтобы решенные поля не показывались.

//

if (!File.Exists(path))

{

Parser.WriteText(path, levelName + ";" + "Решения" + Environment.NewLine);

Parser.ReadFile(path);

GenerateNewField(path);

}

Parser.ReadFile(path);

// ВЫЗОВ ГЕНЕРАЦИИ

GenerateNewField(path);

Parser.ReadFile(path);

ToNewField();

}

/// <summary>

/// Порядковый номер поля по его ID.

/// </summary>

/// <returns> Порядковый номер поля</returns>

int NumOfField()

{

int temp = id;

int ciphersCount = 0;

do

{

temp /= 10;

ciphersCount++;

} while (temp >= 1);

return id - (level + 1) \* (int)Math.Pow(10, ciphersCount - 1);

}

/// <summary>

/// Построение нового поля по информации из файла.

/// </summary>

/// <param name="level"></param>

void ToNewField()

{

string levelName = level == 0 ? "Легкий" : level == 1 ? "Средний" : "Сложный";

string path = pathCurrentLevel;

if (!File.Exists(path))

{

Parser.WriteText(path, levelName + ";" + "Решения" + Environment.NewLine);

Parser.ReadFile(path);

GenerateNewField(path);

}

if (Parser.LevelColumnLength(0) <= NumOfField() + 1)

numOfField = 1;

else

numOfField = Parser.LevelColumnLength(0) - 1;

int levelLength = Parser.LevelColumnLength(0);

selectFieldComboBox.Items.Clear();

if (levelLength > NumOfField())

{

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

{

selectFieldComboBox.Items.Add(i);

}

}

//do

//{

// numOfField = ;

//} while (String.IsNullOrEmpty(CSVParser.GetField(numOfField, level)));

fieldSize = Parser.GetFieldLength(numOfField, 0);

// Массив из строк игрового поля, где каждая строка содержит информацию о весах элементов в своей строке.

// Нужно этот массив задать определенный длины, которая будет соответствовать количеству строк поля, информация о котором

// запишется в этот массив.

// Например, если с файла будет считываться поле размера n x n, то есть с n строками то массив должен быть инициализирован

// с длиной n.

string[] strdataRows;

// записывается массив из полей трех уровней с одинаковых порядковым номером строки.

strdataRows = Parser.GetField(numOfField, 0).Split(new[] { ',' }, StringSplitOptions.RemoveEmptyEntries);

string[] strdata = new string[fieldSize];

timer.Enabled = true;

timer.Start();

pb = new PictureBox[2 \* fieldSize - 1, 2 \* fieldSize - 1];

nodes = new Node[fieldSize, fieldSize];

buttons = new Button[fieldSize, fieldSize];

int x = 200;

int y = 80;

#region Просто красивое выведение ID в форме.

int countOfCiphers = 0;

int copyNumOfField = 0;

copyNumOfField = numOfField;

do

{

copyNumOfField /= 10;

countOfCiphers++;

} while (copyNumOfField >= 1);

// Каждому уровню в числе id соответствует перфая цифра в числе id. 1- легкий, 2 - средний, 3 - сложный.

// Легкому уровню соответствует 0 в числе id.

id = (level + 1) \* (int)Math.Pow(10, countOfCiphers) + numOfField;

idLabel.Text = "ID\_\_" + id.ToString();

#endregion

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

{

// Массив из элементов заданной строки.

strdata = strdataRows[i].Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

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

{

buttons[i, j] = new Button();

buttons[i, j].BackColor = Color.MediumPurple;

buttons[i, j].ForeColor = Color.White;

buttons[i, j].Font = new Font("Cambria", 16, FontStyle.Italic);

buttons[i, j].Text = strdata[j];

buttons[i, j].Size = new Size(50, 50);

buttons[i, j].Location = new Point(x, y);

buttons[i, j].Click += GameButtonClick;

buttons[i, j].MouseEnter += ButtonMouseEnter;

buttons[i, j].MouseLeave += ButtonMouseLeave;

// В имени кнопки(узла) будет содержаться информация о его положении.

buttons[i, j].Name = $"{i};{j}";

System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Drawing2D.GraphicsPath();

myPath.AddEllipse(0, 0, buttons[i, j].Width, buttons[i, j].Height);

Region myRegion = new Region(myPath);

buttons[i, j].Region = myRegion;

buttons[i, j].FlatStyle = FlatStyle.Flat;

buttons[i, j].FlatAppearance.BorderSize = 0;

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

x += 100;

}

x = 200;

y += 100;

}

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

nodes[i, j] = new Node(int.Parse(buttons[i, j].Text), i, j);

}

if (level == 0)

SetFirstHelp();

}

/// <summary>

/// Очищает все игровое пространство: ОЧИЩАЕТ КНОПКИ(вершины), мосты и очищает массивы nodes, buttons, pb

/// </summary>

void ClearAllGameField()

{

if (level == 1)

helpButton.Visible = true;

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

nodes[i, j] = null;

buttons[i, j].Dispose();

buttons[i, j] = null;

}

nodes = null;

for (int i = 0; i < pb.GetLength(0); i++)

{

for (int j = 0; j < pb.GetLength(1); j++)

{

if (pb[i, j] != null)

pb[i, j].Dispose();

pb[i, j] = null;

pbIsSet = null;

}

}

pb = null;

}

/// <summary>

/// Рисование линий после десериализации.

/// </summary>

/// <param name="i"> Строка, в которой находится связь.</param>

/// <param name="j"> Столбец, в которой находится связь.</param>

/// <param name="type"> Тип связи</param>

void DrawSavedLines(int i, int j, string type)

{

int x = j \* 50 + 200;

int y = i \* 50 + 80;

pb[i, j] = new PictureBox();

pb[i, j].Click += PictureBoxRemovingByClick;

if (type == "lvpn")

pb[i, j].Image = Image.FromFile(@"resources/lvpn.png");

if (type == "pvln")

pb[i, j].Image = Image.FromFile(@"resources/pvln.png");

if (type == "g")

pb[i, j].Image = Image.FromFile(@"resources/g.png");

if (type == "v")

pb[i, j].Image = Image.FromFile(@"resources/v.png");

pb[i, j].BackColor = Color.Transparent;

pb[i, j].Size = new Size(50, 50);

pb[i, j].Location = new Point(x, y);

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

}

/// <summary>

/// Расстановка связей сохраненной игры.

/// </summary>

void SetPictureBoxes()

{

for (int i = 0; i < pbIsSet.GetLength(0); i++)

{

for (int j = 0; j < pbIsSet.GetLength(1); j++)

{

if (pbIsSet[i, j])

{

if ((i + j) % 2 == 1)

{

if (j % 2 == 1)

{

DrawSavedLines(i, j, "g");

continue;

}

if (i % 2 == 1)

{

DrawSavedLines(i, j, "v");

continue;

}

}

else

{

if (nodes[(i - 1) / 2, (j - 1) / 2].ElemContains(nodes[(i - 1) / 2 + 1, (j - 1) / 2 + 1]))

{

DrawSavedLines(i, j, "lvpn");

continue;

}

else

{

DrawSavedLines(i, j, "pvln");

continue;

}

}

}

}

}

}

/// <summary>

/// Заполнение листа смежных вершин, с которыми соединен данный узел.

/// </summary>

/// <param name="node"> Элемент Node, чей список смежных соседей заполняется.</param>

/// <param name="neighbours"> Позиция соседа в виде "i,j"</param>

void FillConnectedNodesList(Node node, string[] neighbours)

{

for (int i = 0; i < neighbours.Length; i++)

{

node.AddToNodesList(nodes[int.Parse((neighbours[i].Split(' '))[0]), int.Parse((neighbours[i].Split(' '))[1])]);

}

}

#region Графика наведения мыши на кнопки(меняется их подсветка)

// Содержит в себе значение последного цвета кнопки.

Color previousColor;

/// <summary>

/// Обработчик покидания курсора игровой кнопки.

/// Работает алгоритм графической подсветки.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void ButtonMouseLeave(object sender, EventArgs e)

{

// Если нажата кнопка и курсор покидает зажатую кнопку.

if ((Button)sender == tempButton1)

{

((Button)sender).BackColor = Color.DodgerBlue;

return;

}

// Никакие кнопки не нажаты - курсор просто гуляет по полю, либо зажата одна кнопка и курсор уходит с другой кнопки.

if (tempButton1 == null || ((Button)sender) != tempButton1)

{

if (((Button)sender).BackColor == Color.OrangeRed || ((Button)sender).BackColor == Color.DodgerBlue)

{

((Button)sender).BackColor = previousColor;

return;

}

//buttons[int.Parse((((Button)sender).Name.Split(';'))[0]), int.Parse((((Button)sender).Name.Split(';'))[1])].BackColor

}

}

/// <summary>

/// Обработчик вхождения курсора в игровую кнопку.

/// Работает алгоритм графической подсветки.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void ButtonMouseEnter(object sender, EventArgs e)

{

// Если зажата первая кнопка и курсор наводится на нее.

if (((Button)sender == tempButton1))

{

//previousColor = ((Button)sender).BackColor;

((Button)sender).BackColor = Color.OrangeRed;

return;

}

// Если курсор наводится не на соседнюю кнопку к нажатой кнопке.

if (tempButton1 != null && (Math.Abs(int.Parse((((Button)sender).Name.Split(';'))[0]) - row1) > 1 || Math.Abs(int.Parse((((Button)sender).Name.Split(';'))[1]) - col1) > 1))

{

previousColor = ((Button)sender).BackColor;

((Button)sender).BackColor = Color.OrangeRed;

return;

}

// Никакие кнопки не нажаты - курсор просто двигается по полю, либо курсор наводится на соседнюю кнопку.

if (tempButton1 == null || Math.Abs(int.Parse((((Button)sender).Name.Split(';'))[0]) - row1) <= 1 && Math.Abs(int.Parse((((Button)sender).Name.Split(';'))[1]) - col1) <= 1)

{

previousColor = ((Button)sender).BackColor;

((Button)sender).BackColor = Color.DodgerBlue;

return;

}

}

#endregion

#region Работа с кнопками игрового поля и параллельно с соответствующими узлами Node.

/// <summary>

/// Установка цвета кнопки в зависимости от количества связей соответствующего узла.

/// </summary>

/// <param name="node"> Узел, цвет которого меняется.</param>

void SetButtonColor(Node node)

{

#region Выставление цвета для узла в зависимости от количества соединенных с ним мостов.

if (node.ConnectedNodesCount() > 0 && node.Weight > node.ConnectedNodesCount())

{

buttons[node.X, node.Y].BackColor = Color.Goldenrod;

}

if (node.Weight == node.ConnectedNodesCount())

{

buttons[node.X, node.Y].BackColor = Color.Green;

}

if (node.Weight < node.ConnectedNodesCount())

{

buttons[node.X, node.Y].BackColor = Color.DarkRed;

}

if (node.ConnectedNodesCount() == 0)

{

buttons[node.X, node.Y].BackColor = Color.MediumPurple;

}

#endregion

}

/// <summary>

/// Возвращает false -если связей вообще нет во всем поле.

/// true - если хотя бы одна связь проведена в поле.

/// </summary>

bool ConnectionsExist()

{

if(level!=2)

{

int counter = 0;

foreach (PictureBox elem in pb)

{

if (elem == null)

counter++;

}

if (counter == (2 \* fieldSize - 1) \* (2 \* fieldSize - 1))

return false;

return true;

}

return true;

}

/// <summary>

/// Удаление связи по нажатию на связь. Удаляются элементы из массивов друг друга, которые были соединены этой связью

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void PictureBoxRemovingByClick(object sender, EventArgs e)

{

for (int i = 0; i < pb.GetLength(0); i++)

{

for (int j = 0; j < pb.GetLength(1); j++)

{

if ((PictureBox)sender == pb[i, j])

{

if ((i + j) % 2 == 1 && i % 2 == 1)

{

RemoveFromNodesList(nodes[(i - 1) / 2, j / 2], nodes[(i - 1) / 2 + 1, j / 2]);

DeleteLines(ref pb[i, j]);

#region Выставление цвета для узла в зависимости от количества соединенных с ним мостов.

SetButtonColor(nodes[(i - 1) / 2, j / 2]);

SetButtonColor(nodes[(i - 1) / 2 + 1, j / 2]);

#endregion

}

if ((i + j) % 2 == 1 && i % 2 == 0)

{

RemoveFromNodesList(nodes[i / 2, (j - 1) / 2], nodes[i / 2, (j + 1) / 2]);

DeleteLines(ref pb[i, j]);

#region Выставление цвета для узла в зависимости от количества соединенных с ним мостов.

SetButtonColor(nodes[i / 2, (j - 1) / 2]);

SetButtonColor(nodes[i / 2, (j + 1) / 2]);

#endregion

}

if ((i + j) % 2 == 0)

{

if (nodes[(i - 1) / 2, (j - 1) / 2].ElemContains(nodes[(i - 1) / 2 + 1, (j - 1) / 2 + 1]))

{

RemoveFromNodesList(nodes[(i - 1) / 2, (j - 1) / 2], nodes[(i - 1) / 2 + 1, (j - 1) / 2 + 1]);

DeleteLines(ref pb[i, j]);

#region Выставление цвета для узла в зависимости от количества соединенных с ним мостов.

SetButtonColor(nodes[(i - 1) / 2, (j - 1) / 2]);

SetButtonColor(nodes[(i - 1) / 2 + 1, (j - 1) / 2 + 1]);

#endregion

}

else

{

RemoveFromNodesList(nodes[(i - 1) / 2, (j - 1) / 2 + 1], nodes[(i - 1) / 2 + 1, (j - 1) / 2]);

DeleteLines(ref pb[i, j]);

#region Выставление цвета для узла в зависимости от количества соединенных с ним мостов.

SetButtonColor(nodes[(i - 1) / 2, (j - 1) / 2 + 1]);

SetButtonColor(nodes[(i - 1) / 2 + 1, (j - 1) / 2]);

#endregion

}

}

}

}

}

if (!Checker())

isFinished = false;

if (!ConnectionsExist())

helpButton.Visible = true;

}

// Первая кнопка игрового поля, на которую нажал пользователь.

Button tempButton1;

// Индексы этой кнопки, то есть ее позиции.

string[] pos1;

int row1;

int col1;

// Вторая кнопка игрового поля, на которую нажал пользователь.

Button tempButton2;

// Индексы этой кнопки, то есть ее позиции.

string[] pos2;

int row2;

int col2;

// Массив всех связей(объектов Picture Box).

PictureBox[,] pb;

/// <summary>

/// Обработчик события нажатия по игровой кнопке поля.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void GameButtonClick(object sender, EventArgs e)

{

if (tempButton1 == null)

{

tempButton1 = (Button)sender;

tempButton1.BackColor = Color.DodgerBlue;

pos1 = tempButton1.Name.Split(';');

row1 = int.Parse(pos1[0]);

col1 = int.Parse(pos1[1]);

return;

}

if (tempButton2 == null)

{

tempButton2 = (Button)sender;

pos2 = tempButton2.Name.Split(';');

row2 = int.Parse(pos2[0]);

col2 = int.Parse(pos2[1]);

// В случае, если вторая кнопка не соседняя, то никакие линии не проводятся и кнопки расфокусируются и

// все временные значения умалчиваются.

if (!((row1 == row2 || row2 == row1 + 1 || row2 == row1 - 1) && (col2 == col1 || col2 == col1 + 1 || col2 == col1 - 1)) || (row1 == row2 && col1 == col2))

{

getFocusLabel.Select();

SetDefaultValues();

return;

}

// Если все ок и кнопки оказались соседними.

// Проверка на то, содержатся ли уже элементы в массивах друг друга.

// Можно было сделать через Contains.

if (nodes[row1, col1].ElemCount(nodes[row2, col2]) == 0)

{

if (row1 < row2 && col1 < col2)

{

if (nodes[row1, col1 + 1].ElemCount(nodes[row1 + 1, col1]) == 0)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X + 50, tempButton1.Location.Y + 50, "lvpn");

SetDefaultValues();

}

else

{

SetDefaultValues();

return;

}

}

if (row1 < row2 && col1 > col2)

{

if (nodes[row1, col1 - 1].ElemCount(nodes[row1 + 1, col1]) == 0)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X - 50, tempButton1.Location.Y + 50, "pvln");

SetDefaultValues();

}

else

{

SetDefaultValues();

return;

}

}

if (row1 > row2 && col1 < col2)

{

if (nodes[row1 - 1, col1].ElemCount(nodes[row1, col1 + 1]) == 0)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X + 50, tempButton1.Location.Y - 50, "pvln");

SetDefaultValues();

}

else

{

SetDefaultValues();

return;

}

}

if (row1 > row2 && col1 > col2)

{

if (nodes[row1, col1 - 1].ElemCount(nodes[row1 - 1, col1]) == 0)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X - 50, tempButton1.Location.Y - 50, "lvpn");

SetDefaultValues();

}

else

{

SetDefaultValues();

return;

}

}

if (row1 == row2 && col1 < col2)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X + 50, tempButton1.Location.Y, "g");

SetDefaultValues();

}

if (row1 == row2 && col1 > col2)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X - 50, tempButton1.Location.Y, "g");

SetDefaultValues();

}

if (col1 == col2 && row1 < row2)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X, tempButton1.Location.Y + 50, "v");

SetDefaultValues();

}

if (col1 == col2 && row1 > row2)

{

AddToNodesList(nodes[row1, col1], nodes[row2, col2]);

DrawLines(tempButton1.Location.X, tempButton1.Location.Y - 50, "v");

SetDefaultValues();

}

}

// Если кнопки уже были соединены, то связь убирается и они удаляются из массивов nodesConnected у друг друга

else

{

RemoveFromNodesList(nodes[row1, col1], nodes[row2, col2]);

DeleteLines(ref pb[row1 + row2, col1 + col2]);

pb[row1 + row2, col1 + col2] = null;

SetDefaultValues();

}

if (!Checker())

isFinished = false;

if (Checker() && !isFinished)

saveResultButton.PerformClick();

if (!ConnectionsExist())

helpButton.Visible = true;

else

helpButton.Visible = false;

}

}

#endregion

#region Работа с узлами Node(!не кнопками): удаление элементов из массива, добавление элементов.

/// <summary>

/// Метод, который добавляет в поле nodesConnected двух заданных в параметре элементов друг друга.

/// </summary>

/// <param name="elem1"> Первый узел, который был зафиксирован пользователем.</param>

/// <param name="elem2"> Второй узел, который был зафиксирован пользователем.</param>

void AddToNodesList(Node elem1, Node elem2)

{

elem1.AddToNodesList(elem2);

elem2.AddToNodesList(elem1);

}

/// <summary>

/// Метод, который добавляет в поле nodesConnected двух заданных в параметрах элементов, переданных по ссылке, друг друга.

/// </summary>

/// <param name="elem1"> Первый узел, который был зафиксирован пользователем.</param>

/// <param name="elem2"> Второй узел, который был зафиксирован пользователем.</param>

void AddToNodesList(ref Node elem1, ref Node elem2)

{

elem1.AddToNodesList(elem2);

elem2.AddToNodesList(elem1);

}

/// <summary>

/// Взаимное удаление из своих списков смежных соседей друг друга.

/// </summary>

/// <param name="elem1"> Первый узел, который был зафиксирован пользователем.</param>

/// <param name="elem2"> Второй узел, который был зафиксирован пользователем.</param>

void RemoveFromNodesList(Node elem1, Node elem2)

{

elem1.RemoveFromNodesList(elem2);

elem2.RemoveFromNodesList(elem1);

}

/// <summary>

/// Взаимное удаление узлов, переданных по ссылке, из своих списков смежных соседей друг друга.

/// </summary>

/// <param name="elem1"> Первый узел, который был зафиксирован пользователем.</param>

/// <param name="elem2"> Второй узел, который был зафиксирован пользователем.</param>

void RemoveFromNodesList(ref Node elem1, ref Node elem2)

{

elem1.RemoveFromNodesList(elem2);

elem2.RemoveFromNodesList(elem1);

}

#endregion

#region Рисование и удаление линий.

/// <summary>

/// проведение линии между tempButton1 и tempButton2.

/// </summary>

/// <param name="x"> Горизонтальная координата связи.</param>

/// <param name="y"> Вертикальная координата связи.</param>

/// <param name="type"></param>

void DrawLines(int x, int y, string type)

{

pb[row1 + row2, col1 + col2] = new PictureBox();

pb[row1 + row2, col1 + col2].Click += PictureBoxRemovingByClick;

if (type == "lvpn")

pb[row1 + row2, col1 + col2].Image = Image.FromFile(@"resources/lvpn.png");

if (type == "pvln")

pb[row1 + row2, col1 + col2].Image = Image.FromFile(@"resources/pvln.png");

if (type == "g")

pb[row1 + row2, col1 + col2].Image = Image.FromFile(@"resources/g.png");

if (type == "v")

pb[row1 + row2, col1 + col2].Image = Image.FromFile(@"resources/v.png");

pb[row1 + row2, col1 + col2].BackColor = Color.Transparent;

pb[row1 + row2, col1 + col2].Size = new Size(50, 50);

pb[row1 + row2, col1 + col2].Location = new Point(x, y);

this.Controls.Add(pb[row1 + row2, col1 + col2]);

}

/// <summary>

/// Графическое удаление связи.

/// </summary>

/// <param name="pb"> Связь в поле, которая удаляется.</param>

void DeleteLines(ref PictureBox pb)

{

pb.Dispose();

pb = null;

}

#endregion

#region Установка значений по умолчанию строго для двух кнопок tempButton1 и tempButton2(!)

/// <summary>

/// Сбрасывает все значения , которые относятся к кнопкам tempButton1 и tempButton2, до умалчиваемых.

/// Также отводит фокус на label.

/// </summary>

void SetDefaultValues()

{

getFocusLabel.Select();

if (tempButton1 != null)

{

#region Выставление текущего цвета для tempButton1

if (nodes[row1, col1].ConnectedNodesCount() > 0 && nodes[row1, col1].Weight > nodes[row1, col1].ConnectedNodesCount())

{

tempButton1.BackColor = Color.Goldenrod;

}

if (nodes[row1, col1].Weight == nodes[row1, col1].ConnectedNodesCount())

{

tempButton1.BackColor = Color.Green;

}

if (nodes[row1, col1].Weight < nodes[row1, col1].ConnectedNodesCount())

{

tempButton1.BackColor = Color.DarkRed;

}

if (nodes[row1, col1].ConnectedNodesCount() == 0)

{

tempButton1.BackColor = Color.MediumPurple;

}

#endregion

tempButton1 = null;

}

pos1 = null;

row1 = 0;

col1 = 0;

if (tempButton2 != null)

{

#region Выставление текущего цвета для tempButton2

if (nodes[row2, col2].ConnectedNodesCount() > 0 && nodes[row2, col2].Weight > nodes[row2, col2].ConnectedNodesCount())

{

tempButton2.BackColor = Color.Goldenrod;

}

if (nodes[row2, col2].Weight == nodes[row2, col2].ConnectedNodesCount())

{

tempButton2.BackColor = Color.Green;

}

if (nodes[row2, col2].Weight < nodes[row2, col2].ConnectedNodesCount())

{

tempButton2.BackColor = Color.DarkRed;

}

if (nodes[row2, col2].ConnectedNodesCount() == 0)

{

tempButton2.BackColor = Color.MediumPurple;

}

#endregion

tempButton2 = null;

}

pos2 = null;

row2 = 0;

col2 = 0;

}

#endregion

#region МОЖНО УДАЛИТЬ! Информация о соседних элементах данного узла.

/// <summary>

/// Информация о соседних элементов данного узла.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void nodesInfoButton\_Click(object sender, EventArgs e)

{

if (tempButton1 != null)

{

MessageBox.Show(nodes[row1, col1].ShowInfo() + $"\nВес={nodes[row1, col1].Weight}");

SetDefaultValues();

}

}

#endregion

#region Проверка, правильно ли решена задача.

/// <summary>

/// Сохранение результата решенной задачи пользователем.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

void SaveResultButtonClick(object sender, EventArgs e)

{

string recordName = "";

if (Checker())

{

isFinished = true;

timer.Stop();

timer.Enabled = false;

isFinished = true;

// Сохранить данное игровое поле в уже прорешенные(создать для этого список).

MessageBox.Show($"ПРАВИЛЬНО!\nВы решили данную задачу за время:\n\t{timerLabel.Text}", "ПОЗДРАВЛЯЕМ!", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);

//ResetTimer();

// Запись рекорда.

SaveResultForm inputNameForm = new SaveResultForm();

inputNameForm.ShowDialog();

if (!inputNameForm.CancelClicked)

{

recordName = $"ID\_{id}" + ";" + $"{timerLabel.Text}" + ";" + inputNameForm.NameForStatistics;

if (level == 0)

Parser.AppendText(pathEasyLevelStatistics, recordName + Environment.NewLine);

if (level == 1)

Parser.AppendText(pathMediumLevelStatistics, recordName + Environment.NewLine);

if (level == 2)

Parser.AppendText(pathHardLevelStatistics, recordName + Environment.NewLine);

}

}

else

{

MessageBox.Show("Результат не может быть сохранен, так как еще не решено правильно!", "Проверка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

#region Checker

/// <summary>

/// Проверка, правильно ли решено поле.

/// </summary>

/// <returns></returns>

bool Checker()

{

foreach (var memb in nodes)

{

if (memb.Weight != memb.ConnectedNodesCount())

{

return false;

}

}

return true;

}

#endregion

#endregion

#region Работа с timer

// Показания часов таймера.

int hours;

// Показания минут таймера.

int mins;

// Показания секунд таймера.

int secs;

/// <summary>

/// Сбрасывает таймер

/// </summary>

private void ResetTimer()

{

hours = 0;

mins = 0;

secs = 0;

timerLabel.Text = "00:00:00";

}

/// <summary>

/// Обработчик события, когда проходит интервал в 1 секунду.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void TimerTick(object sender, EventArgs e)

{

if (secs < 59)

{

secs++;

if (secs < 10)

{

if (mins < 10)

timerLabel.Text = $"00:0{mins}:0{secs}";

else

timerLabel.Text = $"00:{mins}:0{secs}";

}

else

{

if (mins < 10)

timerLabel.Text = $"00:0{mins}:{secs}";

else

timerLabel.Text = $"00:{mins}:{secs}";

}

}

else

{

secs = 0;

if (mins < 59)

{

mins++;

if (mins < 10)

if (hours < 10)

timerLabel.Text = $"0{hours}:0{mins}:00";

else

timerLabel.Text = $"{hours}:0{mins}:00";

else

timerLabel.Text = $"{hours}:{mins}:00";

}

else

{

mins = 0;

if (hours < 24)

{

hours++;

if (hours < 10)

timerLabel.Text = $"0{hours}:00:00";

else

timerLabel.Text = $"{hours}:00:00";

}

else

{

ResetTimer();

}

}

}

}

#endregion

#region Сериализация

// Адреса соответствующих xml файлов для сохранения текущего процесса игры.

const string pathSavedGameFieldEasyLevel = "resources/savedGameFieldEasyLevel.xml";

const string pathSavedGameFieldMediumLevel = "resources/savedGameFieldMediumLevel.xml";

const string pathSavedGameFieldHardLevel = "resources/savedGameFieldHardLevel.xml";

// ID текущего игрового поля.

int id;

// Информация о том, решен ли раунд.

bool isFinished;

// Двумерный массив pbIsSet ДЛЯ УДОБСТВА. Содержит информацию о проведенных связях.

// Если есть связь на позиции i,j, то pbIsSet[i,j]=true

bool[,] pbIsSet;

// Массив для того, чтобы потом при сериализации можно было спокойно расставить объекты picture box.

bool[][] pbIsSetCopy;

/// <summary>

/// Заполнение массива pbIsSetCopy. True - если на текущей позиции имеется элемент PictureBox,иначе false.

/// Для дальнейшей сериализации.

/// </summary>

void FillPBIsSetCopy()

{

pbIsSetCopy = new bool[pb.GetLength(0)][];

for (int i = 0; i < pb.GetLength(0); i++)

{

pbIsSetCopy[i] = new bool[pb.GetLength(1)];

for (int j = 0; j < pb.GetLength(1); j++)

{

if (pb[i, j] != null)

{

pbIsSetCopy[i][j] = true;

}

}

}

}

bool close;

// Содержит информацию о соединенных узлах для каждого элемента из nodes.

// т.е. элемент connections[i,j] содержит позиции смежных соседей для элемента nodes[i,j]

// для десериализации.

string[,] connectedNodes;

// Массив размер такой же как и nodes[,] и он ассоциирован с nodes[,], а точнее

// i-ый,j-ый элемент массива bridges содержит информацию о позициях элементов, которые соединены с элементом nodes[i,j] в формате 1 2,0 1,2 2

// Метод нужен для того, чтобы потом можно было с файла считать строку из смежных элементов для этого элемента nodes[i,j].

// Записывает в connections.

string[][] connectedNodesCopy;

/// <summary>

/// Заполнение зубчатого массива connectedNodesCopy информацией о соседях каждого элемента в массиве nodes.

/// Для дальнейшей сериализации.

/// </summary>

void FillConnectedNodesCopy()

{

connectedNodesCopy = new string[nodes.GetLength(0)][];

for (int i = 0; i < nodes.GetLength(0); i++)

{

connectedNodesCopy[i] = new string[nodes.GetLength(1)];

for (int j = 0; j < nodes.GetLength(1); j++)

{

connectedNodesCopy[i][j] = nodes[i, j].ShowInfoForSave();

}

}

}

// То же самое представление данных двумерного массива nodes в виде массива массивов.

Node[][] nodesCopy;

/// <summary>

/// Заполняет зубчатый массив nodesCopy информацией из nodes. Для дальнейшей сериализации.

/// </summary>

void FillNodesCopyArr()

{

nodesCopy = new Node[nodes.GetLength(0)][];

for (int i = 0; i < nodes.GetLength(0); i++)

{

nodesCopy[i] = new Node[nodes.GetLength(1)];

for (int j = 0; j < nodes.GetLength(1); j++)

{

nodesCopy[i][j] = nodes[i, j];

}

}

}

/// <summary>

/// Механизм сериализации данных.

/// Сохраняет информацию, записанную в объект для сохранения InfoForSave, по директории указанной в typeOfFile.

/// </summary>

/// <param name="obj"></param>

/// <param name="typeOfFile"> Директория файла с сохраненным процессом.</param>

private void AutoSave(InfoForSave obj, string typeOfFile)

{

XmlSerializer serGameField = new XmlSerializer(typeof(InfoForSave));

using (FileStream fs = new FileStream(typeOfFile, FileMode.Create))

{

serGameField.Serialize(fs, obj);

}

}

/// <summary>

/// Механизм десериализации данных.

/// Загрузить информацию из файла с сохраненным процессом игры.

/// </summary>

/// <param name="typeOFFile"> Директория файла с сохраненным процессом.</param>

/// <returns></returns>

private InfoForSave LoadSavedGame(string typeOFFile)

{

InfoForSave obj = null;

XmlSerializer deserGameField = new XmlSerializer(typeof(InfoForSave));

using (FileStream fs = new FileStream(typeOFFile, FileMode.Open))

{

obj = (InfoForSave)deserGameField.Deserialize(fs);

}

return obj;

}

/// <summary>

/// Обработчик закрытия окна с игровым полем. Внутри срабатывает механизм сериализация для сохранения текущего процесса игры.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void GameFieldFormClosing(object sender, FormClosingEventArgs e)

{

// Раскомментировать.

timer.Stop();

if (MessageBox.Show("Вы хотите выйти?", "Выход", MessageBoxButtons.YesNo,

MessageBoxIcon.Question) == DialogResult.No)

{

e.Cancel = true;

close = false;

timer.Start();

return;

}

timer.Stop();

timerLabel.Enabled = false;

FillPBIsSetCopy();

FillConnectedNodesCopy();

FillNodesCopyArr();

InfoForSave info = new InfoForSave(id, level, hours, mins, secs, isFinished, nodesCopy, pbIsSetCopy, connectedNodesCopy

, fieldSize, helpButton.Visible);

if (level == 0)

AutoSave(info, pathSavedGameFieldEasyLevel);

if (level == 1)

AutoSave(info, pathSavedGameFieldMediumLevel);

if (level == 2)

AutoSave(info, pathSavedGameFieldHardLevel);

close = true;

e.Cancel = false;

}

#endregion

/// <summary>

/// Заполняет двумерный массив PbISSet, в котором каждый элемент содержит информацию о том, есть ли на данной позиции связь или нет.

/// true - есть . false - нет связи.

/// </summary>

void FillPBIsSet()

{

pbIsSet = new bool[pb.GetLength(0), pb.GetLength(1)];

for (int i = 0; i < pb.GetLength(0); i++)

{

for (int j = 0; j < pb.GetLength(1); j++)

{

// Проверка на то, валидная ли это позиция для связи. То есть не совпадает ли она с вершинами.

// Каждая вершина в матрице связей находится на положительном i и j местах.

if (i % 2 != 0 || j % 2 != 0)

{

// Горизонтальный или вертикальный мост.

if ((i + j) % 2 == 1)

{

// Горизонтальный мост.

if (j % 2 == 1 && nodes[i / 2, (j - 1) / 2].ElemContains(nodes[i / 2, (j + 1) / 2]))

{

pbIsSet[i, j] = true;

continue;

}

// Вертикальный мост

if (i % 2 == 1 && nodes[(i - 1) / 2, j / 2].ElemContains(nodes[(i + 1) / 2, j / 2]))

{

pbIsSet[i, j] = true;

continue;

}

}

else

{

// ЛВПН мост.

if (nodes[(i - 1) / 2, (j - 1) / 2].ElemContains(nodes[(i - 1) / 2 + 1, (j - 1) / 2 + 1]))

{

pbIsSet[i, j] = true;

continue;

}

// ПВЛН мост.

if (nodes[(i - 1) / 2, (j + 1) / 2].ElemContains(nodes[(i + 1) / 2, (j - 1) / 2]))

{

pbIsSet[i, j] = true;

continue;

}

}

}

}

}

}

#region НЕДОSOLVER

/// <summary>

/// У каждого элемента node связи проводятся по принципу Правый || Левый || Правый+Левый.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void SolveGameFieldButton\_Click(object sender, EventArgs e)

{

int counter = 0;

bool notSuccess = false;

do

{

notSuccess = false;

int currentProbability = 0;

ClearField();

timer.Enabled = false;

timer.Stop();

ResetTimer();

counter++;

for (int i = 0; i < nodes.GetLength(0); i++)

{

for (int j = 0; j < nodes.GetLength(1); j++)

{

// Положение узла в поле размера n x n.

if (i != nodes.GetLength(0) - 1 || j != nodes.GetLength(1) - 1)

{

// Положение узла в поле размера n x n.

// Если текущий узел не в последнем столбце или не в последней строке.

if (0 <= i && i < nodes.GetLength(0) - 1 && 0 <= j && j < nodes.GetLength(1) - 1)

{

// Если надо провести только одну связь. 4 варианта и в двух из этих вариантов(где есть горизонт или вертик связь)

// надо также рассмотреть два случая для того, чтобы провести побочную диагональ(pvln).

// Если никакие связи не надо проводить, то рассматриваем 50 на 50 случай с проведением побочной диагонали.

if (nodes[i, j].Weight - nodes[i, j].ConnectedNodesCount() == 0)

{

if (rnd.Next(0, 2) == 0)

{

AddToNodesList(nodes[i + 1, j], nodes[i, j + 1]);

}

}

// Если осталось провести одну связь.

if (nodes[i, j].Weight - nodes[i, j].ConnectedNodesCount() == 1)

{

currentProbability = rnd.Next(0, 3);

// Горизонтальная связь.

if (currentProbability == 0)

{

AddToNodesList(nodes[i, j], nodes[i, j + 1]);

// 50 на 50 будет проведена побочаня диагональ.

if (rnd.Next(0, 2) == 0)

{

AddToNodesList(nodes[i + 1, j], nodes[i, j + 1]);

}

}

// Вертикальная связь.

if (currentProbability == 1)

{

AddToNodesList(nodes[i, j], nodes[i + 1, j]);

// 50 на 50 будет проведена побочаня диагональ.

if (rnd.Next(0, 2) == 0)

{

AddToNodesList(nodes[i + 1, j], nodes[i, j + 1]);

}

}

// Главная диагональ. В этом случае побочная не проводится.

if (currentProbability == 2)

{

AddToNodesList(nodes[i, j], nodes[i + 1, j + 1]);

}

}

// Если осталось провести ровно две связи.

if (nodes[i, j].Weight - nodes[i, j].ConnectedNodesCount() == 2)

{

currentProbability = rnd.Next(0, 3);

// Проводятся g и v связи от текущего элемента.

if (currentProbability == 0)

{

AddToNodesList(nodes[i, j], nodes[i, j + 1]);

AddToNodesList(nodes[i, j], nodes[i + 1, j]);

// 1/2 вероятность соединять горизонт nodes[i,j+1] и вертик nodes[i+1,j] соседей

// текущего nodes[i,j] друг с другом.

// Условие ndpvln

if (rnd.Next(0, 2) == 0)

{

// Побочная диагональ(ПВЛН)

AddToNodesList(nodes[i + 1, j], nodes[i, j + 1]);

}

}

// Проводятся g и lvpn связи от текущего элемента.

if (currentProbability == 1)

{

AddToNodesList(nodes[i, j], nodes[i, j + 1]);

AddToNodesList(nodes[i, j], nodes[i + 1, j + 1]);

}

// Проводятся v и lvpn связи от текущего элемента.

if (currentProbability == 2)

{

AddToNodesList(nodes[i, j], nodes[i + 1, j]);

AddToNodesList(nodes[i, j], nodes[i + 1, j + 1]);

}

}

// Если осталось провести ровно три связи.

if (nodes[i, j].Weight - nodes[i, j].ConnectedNodesCount() == 3)

{

AddToNodesList(nodes[i, j], nodes[i, j + 1]);

AddToNodesList(nodes[i, j], nodes[i + 1, j]);

AddToNodesList(nodes[i, j], nodes[i + 1, j + 1]);

}

}

// Узел в последнем столбце или последней строке.

else

{

// Узел в последнем столбце.

if (j == nodes.GetLength(1) - 1 && i != nodes.GetLength(0) - 1 && rnd.Next(0, 2) == 0)

{

// 50 на 50 соединять текущий nodes[i,j] с соседом по вертикали.

if (nodes[i, j].ConnectedNodesCount() != nodes[i, j].Weight)

AddToNodesList(nodes[i, j], nodes[i + 1, j]);

}

// Узел в последней строке.

// Условие g

if (i == nodes.GetLength(0) - 1 && j != nodes.GetLength(1) - 1 && rnd.Next(0, 2) == 0)

{

// 50 на 50 соединять текущий nodes[i,j] с соседом по горизонтали.

if (nodes[i, j].ConnectedNodesCount() != nodes[i, j].Weight)

AddToNodesList(nodes[i, j], nodes[i, j + 1]);

}

}

}

// Если неправильно проведены связи.

if (nodes[i, j].Weight != nodes[i, j].ConnectedNodesCount())

{

notSuccess = true;

break;

}

//MessageBox.Show($"Элемент {i},{j} провел свое максимальное кол-во мостов к соседям \n{nodes[i, j].ShowInfo()}");

}

if (notSuccess)

break;

}

if (notSuccess)

continue;

if (Checker())

{

MessageBox.Show($"УСПЕХ!\nПопытка №{++counter}.");

getFocusLabel.Select();

isFinished = true;

pbIsSet = new bool[pb.GetLength(0), pb.GetLength(1)];

// Можно вынести, как const в область класса.

// Заполнить у каждого элемента nodes[,] поле nodesConnected.

// содержит позиции соседей текущего элемента nodes[i,j]

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

if (nodes[i, j].ConnectedNodesCount() > 0 && nodes[i, j].Weight > nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Goldenrod;

}

if (nodes[i, j].Weight == nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Green;

}

if (nodes[i, j].Weight < nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.DarkRed;

}

if (nodes[i, j].ConnectedNodesCount() == 0)

{

buttons[i, j].BackColor = Color.MediumPurple;

}

}

// Провести все связи.

FillPBIsSet();

SetPictureBoxes();

}

else

{

pbIsSet = new bool[pb.GetLength(0), pb.GetLength(1)];

// Можно вынести, как const в область класса.

// Заполнить у каждого элемента nodes[,] поле nodesConnected.

// содержит позиции соседей текущего элемента nodes[i,j]

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

if (nodes[i, j].ConnectedNodesCount() > 0 && nodes[i, j].Weight > nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Goldenrod;

}

if (nodes[i, j].Weight == nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Green;

}

if (nodes[i, j].Weight < nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.DarkRed;

}

if (nodes[i, j].ConnectedNodesCount() == 0)

{

buttons[i, j].BackColor = Color.MediumPurple;

}

}

// Провести все связи.

FillPBIsSet();

SetPictureBoxes();

DialogResult res = MessageBox.Show($"Неудача!\nПопытка №{counter}.", "", MessageBoxButtons.YesNo);

if (res == DialogResult.No)

{

break;

}

}

// Пока поле решено неправильно, то есть пока finished == false;

} while (!isFinished);

}

#endregion

/// <summary>

/// Обработчик события нажатия на кнопку "Решить заново".

/// Метод, который отвечает за то, чтобы пользователь заново начал решать текущее поле.

/// Очищается все игровое пространство.

/// Возможность выбрать один из двух вариантов: полностью очищается и таймер сбрасывается или не сбрасывается.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void TryAgainButtonClick(object sender, EventArgs e)

{

ClearField();

timer.Stop();

isFinished = false;

DialogResult res = MessageBox.Show("Сбросить таймер ?", "Решить поле заново", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (res == DialogResult.Yes)

{

ResetTimer();

isFinished = false;

timer.Enabled = true;

timer.Start();

}

timer.Start();

}

/// <summary>

/// Очистка всех связей в игровом поле и очистка списка соседей у каждого элемента Node.

/// Вся графическая часть так же сбрасывается до умолчания.

/// </summary>

void ClearField()

{

if (level == 1 || level == 0)

helpButton.Visible = true;

int fieldSize = this.fieldSize;

isFinished = false;

if (tempButton1 != null)

{

SetDefaultValues();

}

for (int i = 0; i < 2 \* fieldSize - 1; i++)

{

for (int j = 0; j < 2 \* fieldSize - 1; j++)

{

if (pb[i, j] != null)

{

pb[i, j].Dispose();

pb[i, j] = null;

}

}

}

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

{

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

{

nodes[i, j].RemoveAllList();

}

}

for (int i = 0; i < buttons.GetLength(0); i++)

for (int j = 0; j < buttons.GetLength(1); j++)

{

buttons[i, j].BackColor = Color.MediumPurple;

}

}

/// <summary>

/// Обработчик события нажатия на кнопку "Пауза". Открывается диалоговое окно паузы, где пользователь может продолжить или покинуть игру,

/// а также посмотреть правила.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void PauzeButtonClick(object sender, EventArgs e)

{

timer.Stop();

PauzeForm pf = new PauzeForm();

pf.ShowDialog();

if (pf.ExitClicked)

{

this.Close();

pf.ExitClicked = false;

// Раскомментировать.

if (!close)

pauzeButton.PerformClick();

}

if (pf.ContinueClicked)

timer.Start();

}

/// <summary>

/// Обработчик события нажатия на кнопку "Перейти к полю". Метод предоставляет пользователю перейти к ранее решенным или просмотренным полям.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void TurnToFieldButtonClick(object sender, EventArgs e)

{

if (selectFieldComboBox.Items.Count == 0)

{

MessageBox.Show("Не найдено полей, которые решались до данного поля.");

return;

}

if (selectFieldComboBox.SelectedItem == null)

{

MessageBox.Show("Выберите поле.");

selectFieldComboBox.Select();

return;

}

if (selectFieldComboBox.SelectedItem.ToString() == NumOfField().ToString())

{

return;

}

if (level == 1)

helpButton.Visible = true;

//MessageBox.Show("Заполнение текущего поля пропадет!\nВы точно хотите перейти к ", "Внимание!",);

isFinished = false;

timer.Stop();

timer.Enabled = false;

ResetTimer();

ClearAllGameField();

getFocusLabel.Select();

string levelName;

string path = pathCurrentLevel;

levelName = level == 0 ? "Легкий" : level == 1 ? "Средний" : "Сложный";

//

// Сделать так, чтобы решенные поля не показывались.

//

if (!File.Exists(path))

{

Parser.WriteText(path, levelName + ";" + "Решения" + Environment.NewLine);

Parser.ReadFile(path);

GenerateNewField(path);

}

ToFieldAt(int.Parse((selectFieldComboBox.SelectedItem.ToString())));

if (level == 0)

SetFirstHelp();

}

/// <summary>

/// Откроет игровое поле по указанному порядковому номеру в базе данных.

/// </summary>

/// <param name="level"> Порядковый номер задания в файле с заданиями.</param>

void ToFieldAt(int number)

{

int numOfField = number;

fieldSize = Parser.GetFieldLength(numOfField, 0);

// Массив из строк игрового поля, где каждая строка содержит информацию о весах элементов в своей строке.

// Например, если с файла будет считываться поле размера n x n, то есть с n строками то массив должен быть инициализирован

// с длиной n.

string[] strdataRows;

// записывается массив из полей трех уровней с одинаковых порядковым номером строки.

strdataRows = Parser.GetField(numOfField, 0).Split(new[] { ',' }, StringSplitOptions.RemoveEmptyEntries);

string[] strdata = new string[fieldSize];

timer.Enabled = true;

timer.Start();

pb = new PictureBox[2 \* fieldSize - 1, 2 \* fieldSize - 1];

nodes = new Node[fieldSize, fieldSize];

buttons = new Button[fieldSize, fieldSize];

int x = 200;

int y = 80;

#region Просто красивое выведение ID в форме.

int countOfCiphers = 0;

int copyNumOfField = 0;

copyNumOfField = numOfField;

do

{

copyNumOfField /= 10;

countOfCiphers++;

} while (copyNumOfField >= 1);

// Каждому уровню в числе id соответствует перфая цифра в числе id. 1- легкий, 2 - средний, 3 - сложный.

// Легкому уровню соответствует 0 в числе id.

id = (level + 1) \* (int)Math.Pow(10, countOfCiphers) + numOfField;

idLabel.Text = "ID\_\_" + id.ToString();

#endregion

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

{

// Массив из элементов заданной строки.

strdata = strdataRows[i].Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

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

{

buttons[i, j] = new Button();

buttons[i, j].BackColor = Color.MediumPurple;

buttons[i, j].ForeColor = Color.White;

buttons[i, j].Font = new Font("Cambria", 16, FontStyle.Italic);

buttons[i, j].Text = strdata[j];

buttons[i, j].Size = new Size(50, 50);

buttons[i, j].Location = new Point(x, y);

buttons[i, j].Click += GameButtonClick;

buttons[i, j].MouseEnter += ButtonMouseEnter;

buttons[i, j].MouseLeave += ButtonMouseLeave;

// В имени кнопки(узла) будет содержаться информация о его положении.

buttons[i, j].Name = $"{i};{j}";

System.Drawing.Drawing2D.GraphicsPath myPath = new System.Drawing.Drawing2D.GraphicsPath();

myPath.AddEllipse(0, 0, buttons[i, j].Width, buttons[i, j].Height);

Region myRegion = new Region(myPath);

buttons[i, j].Region = myRegion;

buttons[i, j].FlatStyle = FlatStyle.Flat;

buttons[i, j].FlatAppearance.BorderSize = 0;

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

x += 100;

}

x = 200;

y += 100;

}

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

nodes[i, j] = new Node(int.Parse(buttons[i, j].Text), i, j);

}

}

/// <summary>

/// Обработчик события нажатия по кнопке "Показать решение".

/// Метод, в котором считывается информация о решении к текущему игрововому полю.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void SolutionButtonClick(object sender, EventArgs e)

{

ResetTimer();

timer.Stop();

ClearField();

string solution;

solution = Parser.GetField(NumOfField(), 1);

string[,] solutionDouble = FromStringToDoubleArr(solution);

int size = solutionDouble.GetLength(0);

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

{

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

{

if (solutionDouble[i, j] != "P")

{

if (solutionDouble[i, j] == "G")

{

DrawSavedLines(i, j, "g");

AddToNodesList(ref nodes[i / 2, (j - 1) / 2], ref nodes[i / 2, (j + 1) / 2]);

continue;

}

if (solutionDouble[i, j] == "V")

{

DrawSavedLines(i, j, "v");

AddToNodesList(ref nodes[(i - 1) / 2, j / 2], ref nodes[(i + 1) / 2, j / 2]);

continue;

}

if (solutionDouble[i, j] == "LVPN")

{

DrawSavedLines(i, j, "lvpn");

AddToNodesList(ref nodes[(i - 1) / 2, (j - 1) / 2], ref nodes[(i + 1) / 2, (j + 1) / 2]);

continue;

}

if (solutionDouble[i, j] == "PVLN")

{

DrawSavedLines(i, j, "pvln");

AddToNodesList(ref nodes[(i - 1) / 2, (j + 1) / 2], ref nodes[(i + 1) / 2, (j - 1) / 2]);

continue;

}

}

}

}

// Установка цвета кнопок.

for (int i = 0; i < nodes.GetLength(0); i++)

for (int j = 0; j < nodes.GetLength(1); j++)

{

if (nodes[i, j].ConnectedNodesCount() > 0 && nodes[i, j].Weight > nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Goldenrod;

}

if (nodes[i, j].Weight == nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.Green;

}

if (nodes[i, j].Weight < nodes[i, j].ConnectedNodesCount())

{

buttons[i, j].BackColor = Color.DarkRed;

}

if (nodes[i, j].ConnectedNodesCount() == 0)

{

buttons[i, j].BackColor = Color.MediumPurple;

}

}

}

/// <summary>

/// Обработчик нажатия кнопки "Подсказка".

/// Считывает информацию из файла с заданиями о решении к текущему заданию, и предоставляет пользователю только этого решения,

/// что и является только подсказкой.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HelpButtonClick(object sender, EventArgs e)

{

foreach (PictureBox elem in pb)

{

if (elem != null)

{

MessageBox.Show("Подсказка дается только в начале игры, когда никаких связей нет.\nЕсли хотите" +

"воспользоваться подсказкой, очистите поле по кнопке <Решить заново>", "Внимание!", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

}

SetFirstHelp();

helpButton.Visible = false;

}

}

}

### InfoForSave.cs

using System;

using NodeLib;

namespace App

{

/// <summary>

/// Класс для того, чтобы держать информацию о текущих задачах.

/// </summary>

[Serializable]

public class InfoForSave

{

/// <summary>

/// Конструктор умолчания. Для сериализации/десериализации.

/// </summary>

public InfoForSave() { }

// Информация, о том, решено ли поле или нет.

bool finished;

// Зубчатый массив из узлов типа Node.

Node[][] nodes;

// Идентификатор сохраненного поля.

int id;

// Уровень сохраненного поля.

int level;

// Показания таймера часы.

int hours;

// Показания таймера: минуты.

int mins;

// Показания таймера: секунды.

int secs;

// Зубчатый массив с информацией о наличии связей на позициях поля.

bool[][] pbIsSet;

// Размер сохраненного поля.

int fieldSize;

// Содержит информацию о том, скрыта ли кнопка посдказки.

bool helpButtonVisible;

// содержит информацию о соединенных узлах для каждого элемента из nodes.

// т.е. элемент connections[i][j] содержит позиции смежных соседей для элемента nodes[i][j].

string[][] connections;

/// <summary>

/// Свойство с доступом к полю id. Возвращает идентификатор сохраненного поля.

/// </summary>

public int Id { get => id; set => id = value; }

/// <summary>

/// Свойство с доступом к полю level. Возвращает уровень сохраненного поля.

/// </summary>

public int Level { get => level; set => level = value; }

/// <summary>

/// Свойство с доступом к полю hours. Возвращает сохраненное время в часах.

/// </summary>

public int Hours { get => hours; set => hours = value; }

/// <summary>

/// Свойство с доступом к полю mins. Возвращает сохраненное время в минутах.

/// </summary>

public int Mins { get => mins; set => mins = value; }

/// <summary>

/// Свойство с доступом к полю secs. Возвращает сохраненное время в секундах.

/// </summary>

public int Secs { get => secs; set => secs = value; }

/// <summary>

/// Свойство с доступом к массиву nodes. Возвращает зубчатый массив с элементами - узлами.

/// </summary>

public Node[][] Nodes { get => nodes; set => nodes = value; }

/// <summary>

/// Свойство с доступом к полю pbIsSet. Возвразщает зубчатый массив, элементы которого

/// содержат информацию о проведенных связях в позициях поля.

/// </summary>

public bool[][] PbIsSet { get => pbIsSet; set => pbIsSet = value; }

/// <summary>

/// Свойство с доступом к полю connections. Возвразщает зубчатый массив, элементы которого

/// содержат информацию о соединенных соседях для каждого элемента из nodes.

/// </summary>

public string[][] Connections { get => connections; set => connections = value; }

/// <summary>

/// Свойство с доступом к полю finished. Возвразщает информацию, решено ли текущее поле или нет.

/// </summary>

public bool Finished { get => finished; set => finished = value; }

/// <summary>

/// Свойство с доступом к полю fieldSize. Возвращает размер сохраненного поля.

/// </summary>

public int FieldSize { get => fieldSize; set => fieldSize = value; }

/// <summary>

/// Свойство с доступом к полю helpButtonVisible. Возвращает информацию, скрыта ли подсказка или нет.

/// </summary>

public bool HelpButtonVisible { get => helpButtonVisible; set => helpButtonVisible = value; }

/// <summary>

/// Конструктор класса, в котором инициализируются все необходимые для сохранения поля.

/// </summary>

/// <param name="id"> Идентификатор уровня.</param>

/// <param name="level"> Уровень поля.</param>

/// <param name="hours"> Часы.</param>

/// <param name="mins"> Минуты.</param>

/// <param name="secs"> Секунды.</param>

/// <param name="finished"> Решен ли уровень.</param>

/// <param name="nodes"> Зубчатый массив узлов.</param>

/// <param name="pbIsSet"> Зубчатый массив с информацией о том, проведены ли связи.</param>

/// <param name="connections"> Зубчатый массив с информацией о соседях узлов на соответствующих позициях массива.</param>

/// <param name="fieldSize"> Размерность поля.</param>

/// <param name="helpButtonVisible"> Информация о том, скрыта ли кнопка подсказки.</param>

public InfoForSave(int id, int level, int hours, int mins, int secs,bool finished, Node[][] nodes, bool[][] pbIsSet, string[][] connections,int fieldSize, bool helpButtonVisible)

{

this.id = id;

this.level = level;

this.hours = hours;

this.mins = mins;

this.secs = secs;

this.finished = finished;

this.FieldSize = fieldSize;

this.helpButtonVisible = helpButtonVisible;

// Метод для инициализации массивов массивов.

this.nodes = new Node[nodes.Length][];

Array.Copy(nodes, this.nodes, nodes.Length);

this.pbIsSet = new bool[pbIsSet.Length][];

Array.Copy(pbIsSet, this.pbIsSet, pbIsSet.Length);

this.Connections = new string[connections.Length][];

Array.Copy(connections, this.Connections, connections.Length);

}

/// <summary>

/// Метод который из массива массивов nodes возвращает двумерный массив doubNodes.

/// </summary>

/// <returns> Двумерный массив с теми же элементами , что у массива массивов.</returns>

public Node[,] FromNodeOfNodeToDoubleNode()

{

// Массив массивов у нас представляет квадратную матрицу, поэтому длина каждого массива в массиве nodes[][] будет одинаковой.

Node[,] doubNodes = new Node[nodes.Length, nodes[0].Length];

for(int i = 0; i<doubNodes.GetLength(0);i++)

{

for (int j = 0; j < doubNodes.GetLength(1); j++)

{

doubNodes[i, j] = nodes[i][j];

}

}

return doubNodes;

}

/// <summary>

/// Метод, который всю информарцию из зубчатого массива pbIsSet дублирует в двумерный массив doubIsSet.

/// </summary>

/// <returns> Двумерный массив с теми же элементами , что у массива массивов.</returns>

public bool[,] FromPbIsSetOfPbIsSetToDoublePbIsSet()

{

bool[,] doubpbIsSet = new bool[pbIsSet.Length,pbIsSet.Length];

for (int i = 0; i < doubpbIsSet.GetLength(0); i++)

{

for (int j = 0; j < doubpbIsSet.GetLength(1); j++)

{

doubpbIsSet[i, j] = pbIsSet[i][j];

}

}

return doubpbIsSet;

}

/// <summary>

/// Метод, который из зубчатого массива connections дублирует всю информацию в двумерный массив doubConnections.

/// </summary>

/// <returns></returns>

public string[,] FromStrOfStrToDoubleStr()

{

string[,] doubConnections = new string[connections.Length, connections[0].Length];

for (int i = 0; i < doubConnections.GetLength(0); i++)

{

for (int j = 0; j < doubConnections.GetLength(1); j++)

{

doubConnections[i, j] = connections[i][j];

}

}

return doubConnections;

}

}

}

## BridgesExceptionLib.cs

### BridgesException.cs

using System;

namespace BridgesExceptionLib

{

[Serializable]

public class BridgesException : Exception

{

/// <summary>

/// Конструктор умолчания.

/// </summary>

public BridgesException() { }

/// <summary>

/// Конструктор с одним параметром о сообщении об ошибке.

/// </summary>

/// <param name="message"></param>

public BridgesException(string message) : base(message) { }

/// <summary>

/// Конструктор с двумя параметрами с информациями об ошибке.

/// </summary>

/// <param name="message"></param>

/// <param name="inner"></param>

public BridgesException(string message, Exception inner) : base(message, inner) { }

/// <summary>

/// Конструктор с информацией об ошибке.

/// </summary>

/// <param name="info"></param>

/// <param name="context"></param>

protected BridgesException(

System.Runtime.Serialization.SerializationInfo info,

System.Runtime.Serialization.StreamingContext context) : base(info, context) { }

}

}

## NodeLib

### Node

using System;

using System.Collections.Generic;

namespace NodeLib

{

[Serializable]

public class Node

{

// Степень узла.

int weight;

// Список смедных соседей данного объекта.

List<Node> connectedNodesList = new List<Node>();

// Информация о соседних узлах данного объекта в виде строки.

string connectedNodesInfo;

// Поле, значение которого отвечает за то, проведено ли необходимое количество связей.

bool isFull;

// Номер строки, в которой расположен элемент node.

int x;

// Номер столбца, в котором расположен элемент node.

int y;

/// <summary>

/// Конструктор умолчания.

/// </summary>

public Node() { }

/// <summary>

/// Конструктор класса, в котором инициализируются номер строки и номер столбца данного создаваемого объекта.

/// </summary>

/// <param name="x"> Номер строки объекта в поле.</param>

/// <param name="y"> Номер столбца объекта в поле.</param>

public Node(int x, int y)

{

this.X = x;

this.Y = y;

}

/// <summary>

/// Основной конструктор класса, в котором инициализируются позиции создаваемого объекта в поле и его степень(вес).

/// </summary>

/// <param name="weight"></param>

/// <param name="x"></param>

/// <param name="y"></param>

public Node(int weight, int x, int y)

{

this.Weight = weight;

this.X = x;

this.Y = y;

}

/// <summary>

/// Метод, который возвращает элемент из списка соседей с индексом i.

/// </summary>

/// <param name="i"> Индекс, по которому возвращается элемент.</param>

/// <returns> Вовзращаемый элемент из списка соседей, соответствующий индексу.</returns>

public Node GetElemAt(int i)

{

return connectedNodesList[i];

}

/// <summary>

/// Очистка всего списка соседей.

/// </summary>

public void RemoveAllList()

{

connectedNodesList.Clear();

}

/// <summary>

/// Показывает, содержится ли данный элемент node в массиве connectedNodesList у элемента, для которого вызван этот метод.

/// </summary>

/// <param name="node"> Элемент, который проверяется на содержание в списке.</param>

/// <returns></returns>

public bool ElemContains(Node node)

{

return connectedNodesList.Contains(node);

}

/// <summary>

/// Возвращает информацию о соединенных соседях в виде их позиций в виде строки.

/// </summary>

/// <returns> Позиции соединенных соседей.</returns>

public string ShowInfo()

{

int counter = 0;

connectedNodesInfo = null;

foreach (Node elem in connectedNodesList)

{

connectedNodesInfo += $"{++counter}) "+elem.X + ";" + elem.Y + "\n";

}

return connectedNodesInfo;

}

/// <summary>

/// Возвращает информацию о соединенных соседях данного элемента в виде строки для дальнейшего сохранения и восстановления

/// этого списка соседей.

/// </summary>

/// <returns> Позиции смежных соседей, разделенные запятой и пробелами.</returns>

public string ShowInfoForSave()

{

connectedNodesInfo = null;

int counter = 0;

foreach (Node elem in connectedNodesList)

{

counter++;

if (counter != connectedNodesList.Count)

connectedNodesInfo += elem.X + " " + elem.Y + ",";

else

connectedNodesInfo += elem.X + " " + elem.Y;

}

return connectedNodesInfo;

}

/// <summary>

/// Считает, сколько раз элемент node встречается в списке.

/// </summary>

/// <param name="node"> Элемент, количество вхождений в список которого считается.</param>

/// <returns> Количество вхождения элемента в список элемента node.</returns>

public int ElemCount(Node node)

{

int counter = 0;

foreach (var item in connectedNodesList)

{

if(item == node)

{

counter++;

}

}

return counter;

}

/// <summary>

/// Свойсвто для доступа к private полю weight, которое содержит информацию о весе вершины.

/// </summary>

public int Weight { get => weight; set => weight = value; }

/// <summary>

/// Свойство для доступа к private полю номера строки, в которой располагается данный объект.

/// </summary>

public int X { get => x; set => x = value; }

/// <summary>

/// Свойство для доступа к private полю номера столбца, в котором располагается данный объект.

/// </summary>

public int Y { get => y; set => y = value; }

/// <summary>

/// Свойства с доступом к private полю isFull.

/// Вовзращает информацию о том, заполнен ли список соседей или нет.

/// </summary>

public bool IsFull { get => isFull; set => isFull = value; }

/// <summary>

/// Добавление элемента elem в список соседей данного объекта.

/// </summary>

/// <param name="elem"> Добавляемый элемент.</param>

public void AddToNodesList(Node elem)

{

connectedNodesList.Add(elem);

}

/// <summary>

/// Удаление соседа из списка соседей по его индексу в этом списке.

/// </summary>

/// <param name="i"> Индекс, по которому удаляется смежный сосед.</param>

public void RemoveFromNodesListAt(int i)

{

connectedNodesList.RemoveAt(i);

}

/// <summary>

/// Удаление соседа elem из списка соседей.

/// </summary>

/// <param name="elem"> Удаляемый элемент.</param>

public void RemoveFromNodesList(Node elem)

{

connectedNodesList.Remove(elem);

}

/// <summary>

/// Количество связей или количество элементов, с которыми соединен данный объект.

/// Или, грубо говоря, количество соседей.

/// </summary>

/// <returns></returns>

public int ConnectedNodesCount()

{

return connectedNodesList.Count;

}

}

}

## ParserLib

### Parser

using System;

using System.Text;

using System.IO;

using BridgesExceptionLib;

namespace ParserLib

{

public class Parser

{

// Массив, содержащий все строки файла.

static string[] allLines;

// Массив массивов с информацией о строках файла.

static string[][] dataDouble;

/// <summary>

/// Возвращает длину массива из строк файла.

/// </summary>

public static int Length { get => allLines.Length; }

/// <summary>

/// Возвращает информацию о том, существует ли файл по данной директории path.

/// </summary>

/// <param name="path"> Директория файла.</param>

/// <returns></returns>

public static bool FileExists(string path)

{

return File.Exists(path);

}

/// <summary>

/// Метод, который считывает массив строк из файла по директории path.

/// </summary>

/// <param name="path"> Директория файла.</param>

/// <returns> Массив строк из файла.</returns>

public static string[] ReadAll(string path)

{

try

{

return File.ReadAllLines(path);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

/// <summary>

/// Метод, который перезаписывает файл по директории из параметра, записывает в него информацию из параметра text.

/// </summary>

/// <param name="path"></param>

/// <param name="text"></param>

public static void WriteText(string path, string text)

{

try

{

File.WriteAllText(path, text, Encoding.UTF8);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

/// <summary>

/// Метод, который добавляет в файл по директории path информацию, содержащуюся в text.

/// </summary>

/// <param name="path"></param>

/// <param name="text"></param>

public static void AppendText(string path, string text)

{

try

{

File.AppendAllText(path, text);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

/// <summary>

/// Метод, который записывает массив строк text в файл по директории path. Перезаписывает файл, если он уже создан.

/// </summary>

/// <param name="path"></param>

/// <param name="text"></param>

public static void WriteLines(string path, string[] text)

{

try

{

File.WriteAllLines(path, text, Encoding.UTF8);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

/// <summary>

/// Удаляет файл по директории path.

/// </summary>

/// <param name="path"></param>

public static void DeleteFile(string path)

{

try

{

File.Delete(path);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

/// <summary>

/// Возвращает длину столбца с заданиями из файла, который соответствует данному уровню.

/// </summary>

/// <param name="column"></param>

/// <returns></returns>

public static int LevelColumnLength(int column)

{

for (int i = 1; i < dataDouble.Length; i++)

{

if (String.IsNullOrEmpty(dataDouble[i][column]) || String.IsNullOrWhiteSpace(dataDouble[i][column]))

return i;

}

return dataDouble.Length;

}

/// <summary>

/// Метод, который считывает инфорацию из базы данных.

/// Обязателен для вызова после каждой записи в файл, так как здесь инициализируется ассоциированный с базой данных

/// массив allLines.

/// </summary>

/// <param name="path"> Директория файла, который выступает в качестве базы данных.</param>

/// <returns> Массив строк из файла.</returns>

public static void ReadFile(string path)

{

try

{

allLines = File.ReadAllLines(path);

dataDouble = new string[allLines.Length][];

ConvertFromSingleToDouble(allLines, ref dataDouble);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

/// <summary>

/// Метод, который возвращает необходимое поле(ячейку из базы данных).

/// </summary>

/// <param name="column"> 0-задание, 1-решение.</param>

/// <param name="field"> Номер строки в базе данных. Оно же поле, которое соответствует данному уровню.</param>

/// <returns> Возвращает еобходимую информацию из файла.</returns>

public static string GetField(int field, int column)

{

return dataDouble[field][column];

}

/// <summary>

/// Возвращает размер поля, который будет построен по информации из файла.

/// </summary>

/// <param name="field"></param>

/// <param name="column"></param>

/// <returns></returns>

public static int GetFieldLength(int field, int column)

{

return dataDouble[field][column].Split(',').Length;

}

/// <summary>

/// Записывает всю информацию из singleData[] в dataDouble[][].

/// </summary>

/// <param name="singleData"> Массив строк из файла.</param>

/// <param name="doubleData"> Зубчатый массив ровно с тем же данными, что и у singleData</param>

public static void ConvertFromSingleToDouble(string[] singleData, ref string[][] doubleData)

{

for (int i = 0; i < doubleData.Length; i++)

{

doubleData[i] = new string[2];

for (int j = 0; j < doubleData[i].Length; j++)

{

doubleData[i][j] = (singleData[i].Split(';'))[j];

}

}

}

/// <summary>

/// Запись сгенерированных данных в файл с заданиями и решениями в зависимости от column. 0 - задания, 1- решения.

/// </summary>

/// <param name="path"></param>

/// <param name="currentFieldInfo"> Данные: само игровое поле или его решение</param>

/// <param name="column"> Столбец: 0 - поля, 1 - решения к полям.</param>

public static void AddField(string path, string currentFieldInfo, int column)

{

int levelLength = LevelColumnLength(column);

if (levelLength != dataDouble.Length && levelLength > 1 && (dataDouble[levelLength][column] == "" || dataDouble[levelLength][column] == null || String.IsNullOrEmpty(dataDouble[levelLength][column])))

dataDouble[levelLength][column] = currentFieldInfo;

else

{

if (levelLength < dataDouble.Length)

dataDouble[levelLength][column] = currentFieldInfo;

else

{

Array.Resize(ref dataDouble, dataDouble.Length + 1);

dataDouble[dataDouble.Length - 1] = new string[2];

dataDouble[dataDouble.Length - 1][column] = currentFieldInfo;

}

}

// Массив строк с информациями о строках файла.

string[] dataSingle = new string[dataDouble.Length];

for (int i = 0; i < dataDouble.Length; i++)

{

for (int j = 0; j < dataDouble[i].Length; j++)

{

dataSingle[i] += j != dataDouble[i].Length - 1 ? dataDouble[i][j] + ";" : dataDouble[i][j];

}

}

try

{

File.WriteAllLines(path, dataSingle, Encoding.UTF8);

ReadFile(path);

}

catch (Exception e)

{

throw new BridgesException("Ошибка при работе с файлом игры <<BridgesPuzzle>>:" + " " + e.Message);

}

}

}

}

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |