**Аннотация**

Данный документ описывает процесс разработки приложения для решения японских кроссвордов. Приложение предоставляет пользователям интерактивный интерфейс, систему сохранения прогресса, подсказки и статистику. Разработка выполнена на языке C# с использованием .NET Framework и MS SQL Server.

**Cодержание**

[**Аннотация** 2](#_Toc198516017)

[**Введение** 4](#_Toc198516018)

[**1** **Назначение и область применения** 5](#_Toc198516019)

[**2** **Постановка задачи** 6](#_Toc198516020)

[**3** **Описание программы** 7](#_Toc198516021)

[**4** **Программа и методика испытаний** 10](#_Toc198516022)

[**5** **Руководство пользователя** 11](#_Toc198516023)

[**Заключение** 12](#_Toc198516024)

[**Приложения** 13](#_Toc198516025)

[**Источники, используемые при разработке** 51](#_Toc198516026)

## **Введение**

Японские кроссворды - популярный вид головоломок, где необходимо восстановить изображение, основываясь на цифровых подсказках, указанные по строкам и столбцам. Решение таких кроссвордов сильно развивает внимание, логическое мышление и абстракцию. Разбиение кроссвордов на разные уровни сложности позволяет пользователю переживать потоковое состояние, не давая заскучать или чувствовать тревогу.

Создание приложения для решения японских кроссвордов позволит пользователям удобно и быстро решать головоломки на своих устройствах. Приложение будет включать в себя базу данных с готовыми кроссвордами, пользователями, процессами и результатами решения кроссвордов.

## **Назначение и область применения**

**Назначение**:

* Решение японских кроссвордов с возможностью выбора уровня сложности;
* Сохранение прогресса и статистики пользователя;
* Поддержка пользователей с особенностями внимания.

**Область применения**:

* Использование в образовательных, развлекательных и терапевтических целях.

## **Постановка задачи**

**Цель**: cоздать приложение с функционалом:

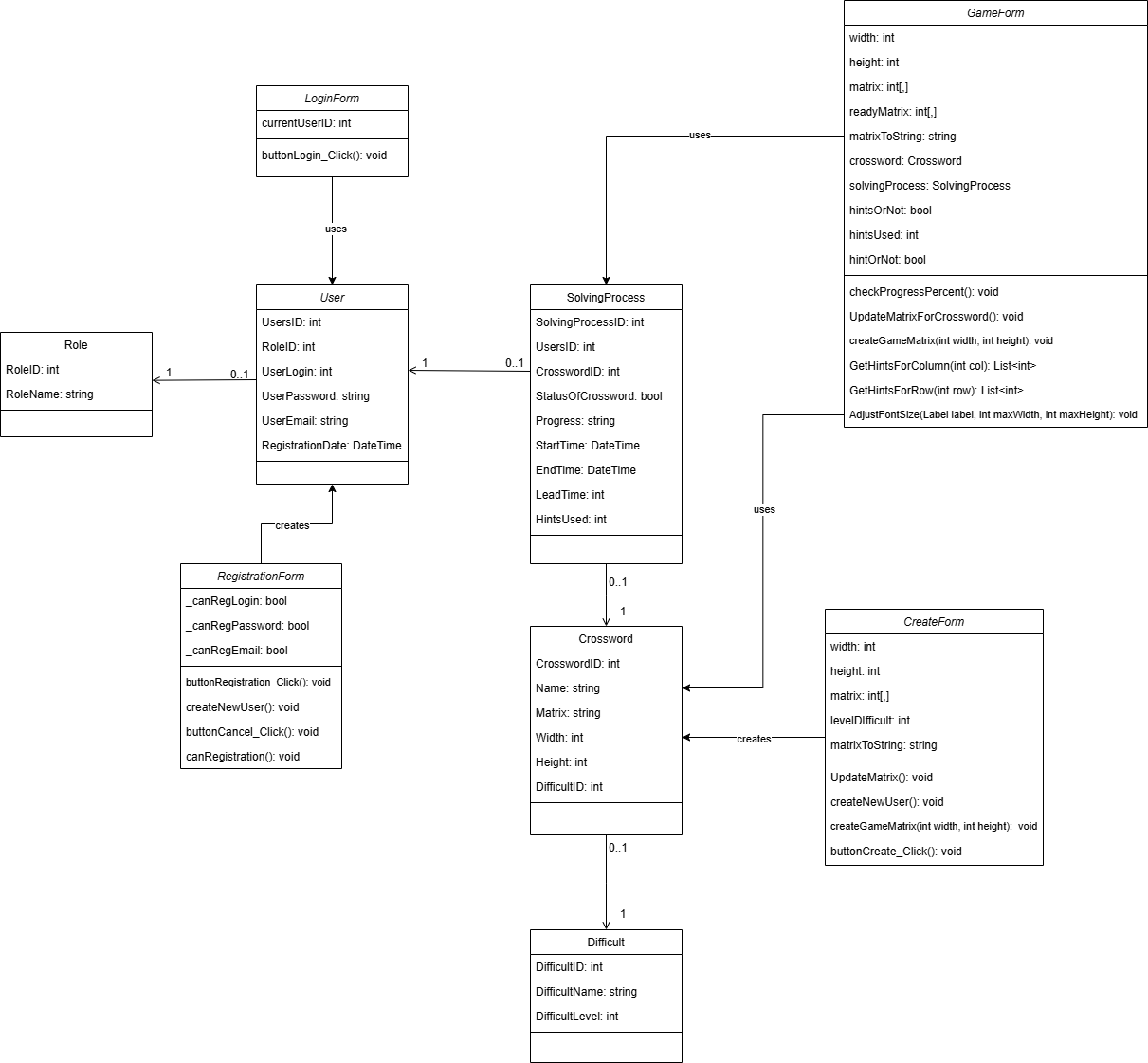
* Интерактивная сетка для решения кроссвордов;
* Система проверки правильности ответов;
* Авторизация и сохранение статистики;
* Подсказки и сохранение прогресса;
* Создание новых кроссвордов

**Ограничения**:

* Разработка на C# (.NET Framework);
* Использование MS SQL Server для базы данных.

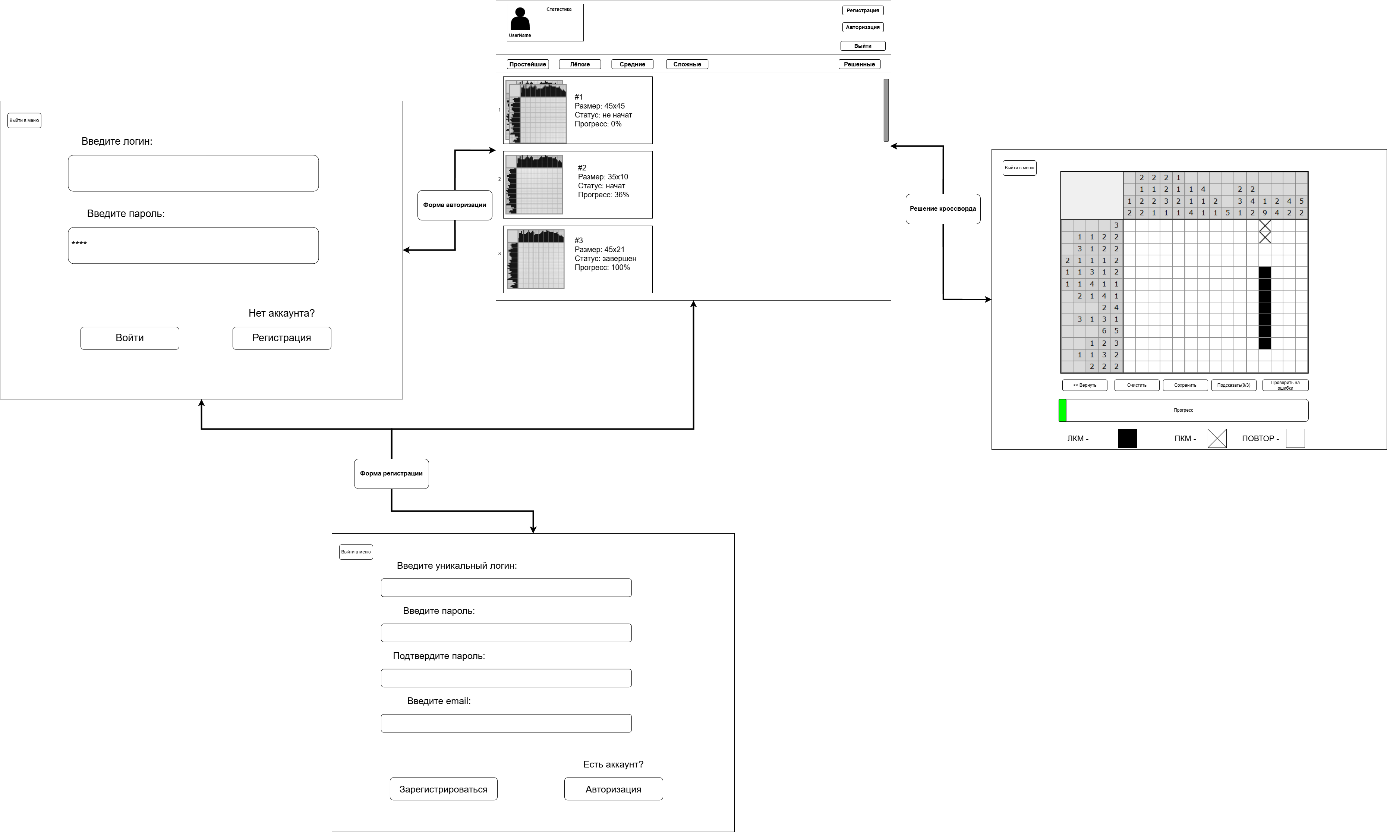
## **Описание программы**

**Архитектура**:

* Клиент-серверная модель.
* **База данных**: Хранение кроссвордов, пользователей, статистики (диаграмма сущностей — см. Приложение 1).
* **Классы**:
* Role;
* User;
* SolvingProcess;
* Crossword;
* Difficult;
* RegistrationForm;
* LoginForm;
* GameForm;
* CreateForm.
  + 

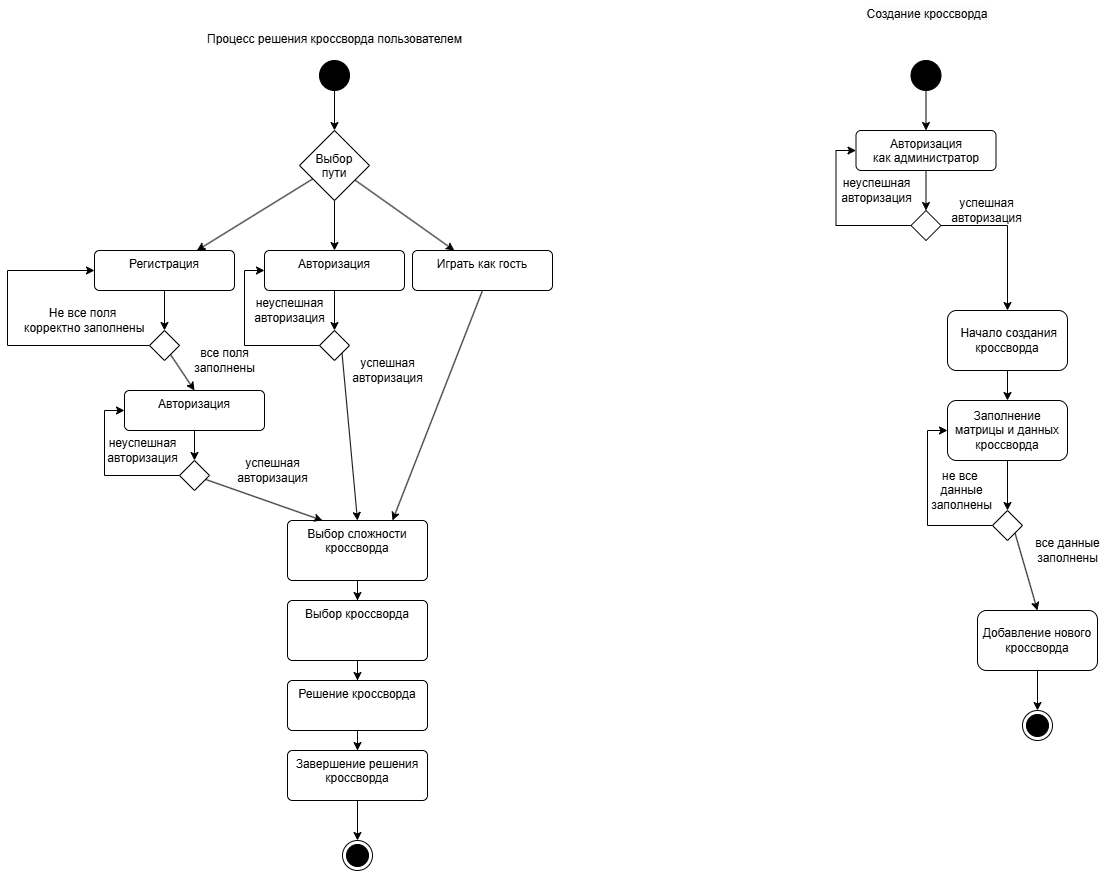
**Интерфейс**:

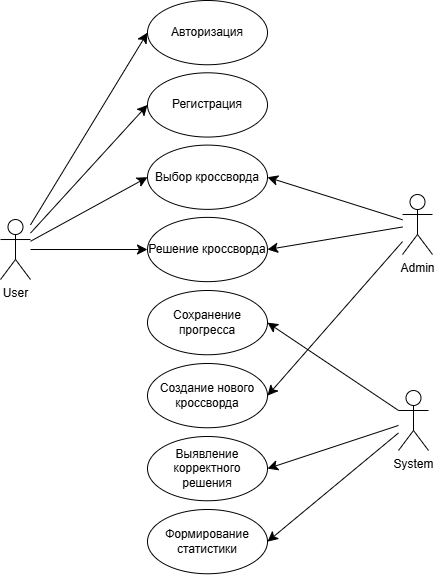
* Форма главного меню с выбором кроссвордов;
* Форма решения с сеткой и подсказками;
* Форма авторизации;
* Форма регистрации;
* Форма создания кроссворда;
* Форма конечного рисунка кроссворда с поздравлением.



**Логика работы**:

* Пользователь авторизовывается/регистрируется/играет как гость – выбирает сложность кроссворда – выбирает кроссоворд – начинает решение – программа считывает значения клеток от пользователя – сохраняет прогресс – сверяет с правильным решением – пользователь завершает решение – при верном решении программа поздравляет пользователя и сохраняет процесс как завершенный





## **Программа и методика испытаний**

**Этапы тестирования**:

1. **Функциональное тестирование**:
   * Проверка корректности авторизации.
   * Тестирование сохранения прогресса.
   * Валидация проверки решений.
2. **Тестирование ошибок**:
   * Обработка некорректного ввода (например, буквы вместо цифр).

**Результаты**: Все функции соответствуют требованиям ТЗ.

## **Руководство пользователя**

1. **Регистрация**: Введите логин и пароль на странице авторизации.
2. **Выбор кроссворда**: на главной странице выберите уровень сложности.
3. **Решение**:
   * Кликните по клетке, чтобы закрасить её.
   * Используйте кнопку «Проверить» для валидации.
   * Сохраните прогресс кнопкой «Сохранить».
4. **Статистика**: Просматривайте результаты в верхней панели главного меню.

## **Заключение**

Приложение успешно реализует все требования ТЗ. Использование современных технологий (C#, MS SQL) обеспечивает стабильность и масштабируемость. Дальнейшее развитие может включать добавление новых уровней и редактирование старых.

## **Приложения**

1. **Текст программы**:
   * + **Program.cs:**

using Nonograms\_1.\_1.Models;

using System;

using System.Windows.Forms;

namespace Nonograms\_1.\_1

{

static class Program

{

public static ModelNonograms context = new ModelNonograms();

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new MenuForm());

}

}

}

* + - **Crossword.cs:**

namespace Nonograms\_1.\_1.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

[Table("Crossword")]

public partial class Crossword

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Crossword()

{

SolvingProcesses = new HashSet<SolvingProcess>();

}

public int CrosswordID { get; set; }

[Required]

[StringLength(255)]

public string Name { get; set; }

[Required]

public string Matrix { get; set; }

public int Width { get; set; }

public int Height { get; set; }

public int DifficultID { get; set; }

public virtual Difficult Difficult { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SolvingProcess> SolvingProcesses { get; set; }

}

}

* + - **Difficult.cs:**

namespace Nonograms\_1.\_1.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

[Table("Difficult")]

public partial class Difficult

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Difficult()

{

Crosswords = new HashSet<Crossword>();

}

public int DifficultID { get; set; }

[Required]

[StringLength(255)]

public string DifficultName { get; set; }

public int DifficultLevel { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<Crossword> Crosswords { get; set; }

}

}

* + - **Role.cs:**

namespace Nonograms\_1.\_1.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

[Table("Role")]

public partial class Role

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public Role()

{

Users = new HashSet<User>();

}

public int RoleID { get; set; }

[Required]

[StringLength(255)]

public string RoleName { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<User> Users { get; set; }

}

}

* + - **SolvingProcess.cs:**

namespace Nonograms\_1.\_1.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

[Table("SolvingProcess")]

public partial class SolvingProcess

{

public int SolvingProcessID { get; set; }

public int? UsersID { get; set; }

public int? CrosswordID { get; set; }

public bool? StatusOfCrossword { get; set; }

public string Progress { get; set; }

[Column(TypeName = "datetime2")]

public DateTime? StartTime { get; set; }

[Column(TypeName = "datetime2")]

public DateTime? EndTime { get; set; }

public int? LeadTime { get; set; }

public int? HintsUsed { get; set; }

public virtual Crossword Crossword { get; set; }

public virtual User User { get; set; }

}

}

* + - **User.cs:**

namespace Nonograms\_1.\_1.Models

{

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity.Spatial;

public partial class User

{

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]

public User()

{

SolvingProcesses = new HashSet<SolvingProcess>();

}

[Key]

public int UsersID { get; set; }

[Required]

[StringLength(255)]

public string UserLogin { get; set; }

[Required]

[StringLength(255)]

public string UserPassword { get; set; }

[Required]

[StringLength(255)]

public string UserEmail { get; set; }

[Column(TypeName = "date")]

public DateTime? RegistrationDate { get; set; }

public int? RoleID { get; set; }

public virtual Role Role { get; set; }

[System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]

public virtual ICollection<SolvingProcess> SolvingProcesses { get; set; }

}

}

* + - **ModelNonograms.cs:**

using System;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity;

using System.Linq;

namespace Nonograms\_1.\_1.Models

{

public partial class ModelNonograms : DbContext

{

public ModelNonograms()

: base("name=ModelNonograms4")

{

}

public virtual DbSet<Crossword> Crosswords { get; set; }

public virtual DbSet<Difficult> Difficults { get; set; }

public virtual DbSet<Role> Roles { get; set; }

public virtual DbSet<SolvingProcess> SolvingProcesses { get; set; }

public virtual DbSet<User> Users { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Difficult>()

.HasMany(e => e.Crosswords)

.WithRequired(e => e.Difficult)

.WillCascadeOnDelete(false);

}

}

}

* + - **MenuForm.cs:**

using Nonograms\_1.\_1.forms;

using Nonograms\_1.\_1.Forms;

using Nonograms\_1.\_1.Models;

using System;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using System.Windows.Forms.VisualStyles;

namespace Nonograms\_1.\_1

{

public partial class MenuForm : Form

{

private User currentUser;

private int currentLevel;

private bool showOnlyReady;

public MenuForm()

{

InitializeComponent();

addButtonLevels();

updateUIForUser(currentUser);

}

private void addButtonLevels()

{

// добавляем сложности

var difficults = Program.context.Difficults.OrderBy(d => d.DifficultLevel).ToList();

foreach (var difficult in difficults)

{

Button button = new Button()

{

Name = $"buttonDifficult{difficult.DifficultLevel}",

Width = buttonCompleted.Width,

Text = difficult.DifficultName,

BackColor = SystemColors.Window,

Margin = new Padding(13, 13, 0, 13),

Font = buttonCompleted.Font,

Tag = difficult.DifficultLevel,

};

button.Click += buttonLevel\_Click;

FLPLevels.Controls.Add(button);

}

}

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

{

Button button = sender as Button;

if (button == null) return;

foreach (Button buttons in FLPLevels.Controls)

{

if (buttons.Tag != button.Tag)

{

buttons.BackColor = SystemColors.Window;

}

}

if (currentLevel != (int)button.Tag)

{

button.BackColor = Color.LightGreen;

currentLevel = (int)button.Tag;

}

else

{

button.BackColor = SystemColors.Window;

currentLevel = 0;

}

fillPanelOfCrosswords();

}

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

{

if (!showOnlyReady)

{

buttonCompleted.BackColor = Color.LightGreen;

showOnlyReady = true;

}

else

{

buttonCompleted.BackColor = SystemColors.Window;

showOnlyReady = false;

}

fillPanelOfCrosswords();

}

private void fillPanelOfCrosswords()

{

FLPanelOfCrosswords.Controls.Clear();

var crosswords = Program.context.Crosswords.OrderBy(x => x.CrosswordID).ToList();

if (currentLevel != 0)

{

var difficult = Program.context.Difficults.Where(d=>d.DifficultLevel == currentLevel).FirstOrDefault();

crosswords = crosswords.Where(x => x.DifficultID == difficult.DifficultID).ToList();

}

// заполнение листа кроссвордов для авторизованного пользователя

foreach (Crossword crossword in crosswords)

{

CrosswordUserControl crosswordUserControl = new CrosswordUserControl(crossword, currentUser);

if (currentUser != null)

{

SolvingProcess solvingProcess = Program.context.SolvingProcesses

.Where(s => s.CrosswordID == crossword.CrosswordID && s.UsersID == currentUser.UsersID).FirstOrDefault();

if (solvingProcess != null)

{

crosswordUserControl = new CrosswordUserControl(solvingProcess);

if (!showOnlyReady || solvingProcess.StatusOfCrossword == true)

{

FLPanelOfCrosswords.Controls.Add(crosswordUserControl);

}

}

else

{

if (!showOnlyReady)

{

FLPanelOfCrosswords.Controls.Add(crosswordUserControl);

}

}

crosswordUserControl.GameFormClosed += (s, args) =>

{

updateUIForUser(currentUser);

};

}

else

{

if (!showOnlyReady)

{

FLPanelOfCrosswords.Controls.Add(crosswordUserControl);

}

}

}

}

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

{

if (currentUser != null)

{

currentUser = null;

updateUIForUser(currentUser);

MessageBox.Show(

"Вы вышли из аккаунта",

"",

MessageBoxButtons.OK,

MessageBoxIcon.Information

);

}

}

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

{

LoginForm loginForm = new LoginForm();

if (loginForm.ShowDialog() == DialogResult.OK)

{

// передаем значение ID авторизованного пользователя из формы авторизации в эту форму

currentUser = Program.context.Users.FirstOrDefault(x => x.UsersID == loginForm.currentUserID);

// обновляем интерфейс главной формы

updateUIForUser(currentUser);

}

}

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

{

}

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

{

RegistrationForm registrationForm = new RegistrationForm();

if (registrationForm.ShowDialog() == DialogResult.OK)

{

}

}

private void updateUIForUser(User currentUser)

{

if (currentUser != null)

{

// обновление интерфейса

var readySolvingProcessesOfUser = Program.context.SolvingProcesses.Where(s => s.UsersID == currentUser.UsersID && s.StatusOfCrossword == true).ToList();

if (readySolvingProcessesOfUser.Count > 0)

{

labelStat.Text = $"Пройденных кроссвордов: {readySolvingProcessesOfUser.Count}" + Environment.NewLine;

int countOfDifficult = 0;

int countOfHintsUsed = 0;

foreach (var process in readySolvingProcessesOfUser)

{

var crossword = process.Crossword;

countOfDifficult += crossword.Difficult.DifficultLevel;

countOfHintsUsed += (int)process.HintsUsed;

}

var middleDiff = Program.context.Difficults.Where(d => d.DifficultLevel ==

countOfDifficult / readySolvingProcessesOfUser.Count).FirstOrDefault();

labelStat.Text += $"Средняя сложность кроссворда:{middleDiff.DifficultName}";

labelStat.Text += Environment.NewLine;

labelStat.Text += $"В среднем подсказок: {countOfHintsUsed / readySolvingProcessesOfUser.Count}";

}

else

{

labelStat.Text = $"Пройденных кроссвордов: 0" + Environment.NewLine;

labelStat.Text += $"Средняя сложность кроссворда: неизвестно" + Environment.NewLine;

labelStat.Text += $"В среднем подсказок: 0";

}

labelStatTitle.Visible = true;

labelStat.Visible = true;

labelUsername.Text = currentUser.UserLogin;

if (currentUser.RoleID == 1)

{

buttonGoCreate.Enabled = true;

buttonGoCreate.Visible = true;

}

else

{

buttonGoCreate.Enabled = false;

buttonGoCreate.Visible = false;

}

}

else

{

labelStat.Text = "";

labelUsername.Text = "User";

labelStatTitle.Visible = false;

labelStat.Visible = false;

buttonGoCreate.Enabled = false;

buttonGoCreate.Visible = false;

}

fillPanelOfCrosswords();

}

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

{

}

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

{

this.Hide(); // Скрываем главную форму

CreateForm createForm = new CreateForm();

if (createForm.ShowDialog() == DialogResult.OK)

{

}

fillPanelOfCrosswords();

this.Show();

}

private void FLPanelOfCrosswords\_Paint(object sender, PaintEventArgs e)

{

}

}

}

* + - **LoginForm.cs:**

using Nonograms\_1.\_1.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Nonograms\_1.\_1.Forms

{

public partial class LoginForm : Form

{

public int currentUserID { get; private set; }

public LoginForm()

{

InitializeComponent();

}

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

{

}

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

{

this.Close();

}

private void textBoxPassword\_TextChanged(object sender, EventArgs e)

{

}

private void label2\_Click(object sender, EventArgs e)

{

}

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

{

pictureBoxUnvisible.Visible = false;

pictureBoxVisible.Visible = true;

textBoxPassword.UseSystemPasswordChar = false;

}

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

{

pictureBoxVisible.Visible = false;

pictureBoxUnvisible.Visible = true;

textBoxPassword.UseSystemPasswordChar = true;

}

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

{

// Пример проверки авторизации

string login = textBoxLogin.Text;

string password = textBoxPassword.Text;

try

{

List<User> users = Program.context.Users.ToList();

User u = users.FirstOrDefault(p => p.UserLogin == login && p.UserPassword == password);

if (u != null)

{

currentUserID = u.UsersID;

// Если авторизация успешна, закрываем форму с результатом OK

MessageBox.Show(

"Успешный вход!",

"Заголовок",

MessageBoxButtons.OK,

MessageBoxIcon.Information

);

DialogResult = DialogResult.OK;

this.Close();

}

else

{

MessageBox.Show(

"Неверный логин или пароль!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error

);

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message.ToString());

}

}

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

{

textBoxLogin.Text = "";

textBoxPassword.Text = "";

}

private void textBoxPassword\_TextChanged\_1(object sender, EventArgs e)

{

}

}

}

* + - **RegistrationForm.cs:**

using Nonograms\_1.\_1.Models;

using System;

using System.Data.SqlClient;

using System.Drawing;

using System.Linq;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace Nonograms\_1.\_1.Forms

{

public partial class RegistrationForm: Form

{

private bool \_canRegLogin = false;

private bool \_canRegPassword = false;

private bool \_canRegEmail = false;

public RegistrationForm()

{

InitializeComponent();

}

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

{

this.Close();

}

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

{

pictureBoxUnvisible.Visible = false;

pictureBoxVisible.Visible = true;

textBoxPassword.UseSystemPasswordChar = false;

textBoxRetryPassword.UseSystemPasswordChar = false;

}

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

{

pictureBoxVisible.Visible = false;

pictureBoxUnvisible.Visible = true;

textBoxPassword.UseSystemPasswordChar = true;

textBoxRetryPassword.UseSystemPasswordChar = true;

}

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

{

}

private void label2\_Click(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void textBoxUsername\_TextChanged(object sender, EventArgs e)

{

if (textBoxLogin.BackColor == Color.Red) {

textBoxLogin.BackColor = SystemColors.Window;

}

}

private void textBoxPassword\_TextChanged(object sender, EventArgs e)

{

if (textBoxPassword.BackColor == Color.Red)

{

textBoxPassword.BackColor = SystemColors.Window;

}

}

private void textBoxRetryPassword\_TextChanged(object sender, EventArgs e)

{

}

private void textBoxEmail\_TextChanged(object sender, EventArgs e)

{

if(textBoxEmail.BackColor == Color.Red)

{

textBoxEmail.BackColor = SystemColors.Window;

}

}

private void canRegistration()

{

\_canRegLogin = true;

\_canRegEmail = true;

\_canRegPassword = true;

string login = textBoxLogin.Text;

string password = textBoxPassword.Text.Trim();

string retryPassword = textBoxRetryPassword.Text;

string email = textBoxEmail.Text;

string patternEmail = @"^[^@\s]+@[^@\s]+\.[^@\s]+$";

if (Program.context.Users.Any(u => u.UserLogin == login) ||

login.Length < 3)

{

\_canRegLogin = false;

}

if (password.Length < 8 ||

password != retryPassword ||

password.Contains(" "))

{

\_canRegPassword = false;

}

if(!Regex.IsMatch(email, patternEmail)){

\_canRegEmail = false;

}

}

private void textBoxColors()

{

if (!\_canRegLogin)

{

textBoxLogin.BackColor = Color.Red;

}

if (!\_canRegPassword)

{

textBoxPassword.BackColor = Color.Red;

}

if (!\_canRegEmail)

{

textBoxEmail.BackColor = Color.Red;

}

}

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

{

textBoxLogin.Text = "";

textBoxPassword.Text = "";

textBoxRetryPassword.Text = "";

textBoxEmail.Text = "";

}

private void createNewUser()

{

string login = textBoxLogin.Text;

string password = textBoxPassword.Text;

string email = textBoxEmail.Text;

User user = new User()

{

UserLogin = login,

UserPassword = password,

UserEmail = email,

RegistrationDate = DateTime.Now,

};

Program.context.Users.Add(user);

Program.context.SaveChanges();

// фикс странного бага с добавлением пробелов к концу пароля (trim не работает)

string connectionString = "data source=localhost;initial catalog=Nonograms;integrated security=True;trustservercertificate=True;MultipleActiveResultSets=True;App=EntityFramework";

using (SqlConnection connection = new SqlConnection(connectionString))

{

try

{

connection.Open();

string sql = @"UPDATE Users

SET UserPassword = TRIM(UserPassword)

WHERE UserPassword LIKE '% ' OR UserPassword LIKE ' %'";

using (SqlCommand command = new SqlCommand(sql, connection)) {

command.ExecuteNonQuery();

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка: {ex.Message}");

}

}

}

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

{

}

private void textBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if (char.IsWhiteSpace(e.KeyChar)) // Проверяем все пробельные символы

{

e.Handled = true;

System.Media.SystemSounds.Beep.Play();

}

}

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

{

canRegistration(); // проверка на корректность ввода данных

textBoxColors(); // маркировка неправильно введенных данных

if (\_canRegLogin && \_canRegPassword && \_canRegEmail)

{

// создание нового пользователя

createNewUser();

// оповещение

MessageBox.Show(

"Успешная регистрация!",

"Регистрация",

MessageBoxButtons.OK,

MessageBoxIcon.Information

);

DialogResult = DialogResult.OK;

this.Close();

}

}

}

}

* + - **GameForm.cs:**

using Nonograms\_1.\_1.forms;

using Nonograms\_1.\_1.Models;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Nonograms\_1.\_1.Forms

{

public partial class GameForm : Form

{

// свойства матриц

private int \_width; // длина в клетках

private int \_height; // ширина в клетках

private int[,] matrix; // заполняемая матрица

private int[,] readyMatrix; // готовая матрица

// значения клеток в цифрах

private int valueOfX = 2; // зачеркнутый крестик

private int valueOfBlack = 1; // черная клетка

private int valueOfWhite = 0; // белая клетка

// переменные для получения данных из бд

private string name;

private Crossword \_crossword;

private SolvingProcess \_solvingProcess;

// проценты

private int \_countOfBlack = 0; // счетчик черных клеток готового кроссворда для вычисления процента

private int \_readyPercent = 0; // процент готовности решения

//подсказки

private bool hintOrNot = false;

private int hintsUsed = 0;

private const int maxOfHints = 10;

// таймер

public int A = 0; // десятки часов таймера

public int B = 0; // часы таймера

public int C = 0; // десятки минут таймера

public int D = 0; // минуты таймер

public int E = 0; // десятки секунд таймера

public int F = 0; // секунды таймера

/// <summary>

/// Конструктор игровой формы для неавторизованного пользователя

/// </summary>

/// <param name="crossword"></param>

public GameForm(Crossword crossword)

{

InitializeComponent();

this.\_crossword = crossword;

// инициализируем компоненты матрицы

\_width = \_crossword.Width;

\_height = \_crossword.Height;

// размер заполняемой матрицы

matrix = new int[\_width, \_height];

name = \_crossword.Name;

// размер готовой числовой матрицы

readyMatrix = new int[\_width, \_height];

// Заполняем готовую матрицу

string matrixString = \_crossword.Matrix;

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

int index = i \* \_height + j;

if (index < matrixString.Length)

{

readyMatrix[i, j] = int.Parse(Convert.ToString(\_crossword.Matrix[index]));

}

}

}

// создаем матрицу(игровое поле) из Label

createGameMatrix(\_width, \_height);

// вычисление количества черных клеток у готового кроссворда

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

if (readyMatrix[i, j] == valueOfBlack)

{

\_countOfBlack++;

}

}

}

// устанавливаем таймер

timer.Enabled = true;

// подсказки

buttonHint.Text = $"Подсказать {hintsUsed}/{maxOfHints}";

}

/// <summary>

/// Конструктор игровой формы для авторизованного пользователя

/// </summary>

/// <param name="solvingProcess"></param>

public GameForm(SolvingProcess solvingProcess)

{

this.\_solvingProcess = solvingProcess;

this.\_crossword = Program.context.Crosswords.FirstOrDefault(x => x.CrosswordID == \_solvingProcess.CrosswordID);

if (\_solvingProcess.LeadTime == null)

{

\_solvingProcess.LeadTime = 0;

}

InitializeComponent();

// инициализируем компоненты матрицы

\_width = \_crossword.Width;

\_height = \_crossword.Height;

// размер заполняемой матрицы

matrix = new int[\_width, \_height];

// заполняем заполняемую матрицу

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

int index = i \* \_height + j;

if (index < \_solvingProcess.Progress.Length)

{

matrix[i, j] = int.Parse(Convert.ToString(\_solvingProcess.Progress[index]));

}

}

}

name = \_crossword.Name;

// размер готовой числовой матрицы

readyMatrix = new int[\_width, \_height];

// Заполняем готовую матрицу

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

int index = i \* \_height + j;

if (index < \_crossword.Matrix.Length)

{

readyMatrix[i, j] = int.Parse(Convert.ToString(\_crossword.Matrix[index]));

}

}

}

// создаем матрицу(игровое поле) из Label

createGameMatrix(\_width, \_height);

if (\_solvingProcess.StatusOfCrossword == true)

{

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

// делаем некликабельными пиксели

pixel.MouseDown -= Pixel\_MouseDown;

}

}

}

// вычисление количества черных клеток у готового кроссворда

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

if (readyMatrix[i, j] == valueOfBlack)

{

\_countOfBlack++;

}

}

}

checkProgressPercent();

// ставим таймер

var leadTime = (int)\_solvingProcess.LeadTime;

// Часы

int hours = leadTime / 3600;

// Десятки часов

A = hours / 10;

// Единицы часов

B = hours % 10;

// Минуты

int minutes = (leadTime % 3600) / 60;

// Десятки минут

C = minutes / 10;

// Единицы минут

D = minutes % 10;

// Секунды

int seconds = leadTime % 60;

// Десятки секунд

E = seconds / 10;

// Единицы секунд

F = seconds % 10;

labelTime.Text = $"{A}{B}:{C}{D}:{E}{F}";

if (\_solvingProcess.StatusOfCrossword == false)

{

timer.Enabled = true;

}

// подсказки

hintsUsed = (int)\_solvingProcess.HintsUsed;

buttonHint.Text = $"Подсказать {hintsUsed}/{maxOfHints}";

}

private void Pixel\_MouseDown(object sender, MouseEventArgs e)

{

if (!hintOrNot)

{

// проверка нажатия левой кнопки по пикселю

if (e.Button == MouseButtons.Left)

{

Label pixel = (Label)sender;

pixel.Tag = null;

if (pixel.BackColor == Color.LightBlue)

{

pixel.BackColor = Color.Black; // Меняем цвет пикселя

}

// проверка повторного нажатия по пикселю

else if (pixel.BackColor == Color.Black)

{

pixel.BackColor = Color.LightBlue;

}

}

// проверка нажатия правой кнопки по пикселю

if (e.Button == MouseButtons.Right)

{

Label pixel = (Label)sender;

if (pixel.Tag == null)

{

pixel.BackColor = Color.LightBlue;

// Для показа крестика:

pixel.Tag = "cross";

pixel.Invalidate(); // Перерисовываем

}

// проверка повторного нажатия по пикселю

else if (pixel.Tag == "cross")

{

pixel.BackColor = Color.LightBlue;

//сброс крестика

pixel.Tag = null;

pixel.Invalidate();

}

}

}

else

{

Label pixel = (Label)sender;

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

if (pixel.Name == pixelName)

{

if (readyMatrix[i, j] == valueOfBlack && pixel.BackColor != Color.Black)

{

pixel.BackColor = Color.Black;

pixel.Tag = null;

pixel.Invalidate();

hintsUsed++;

}

// значение закрашенной клетки

else if ((readyMatrix[i, j] == valueOfWhite || readyMatrix[i, j] == valueOfX) && pixel.BackColor == Color.Black || pixel.BackColor == Color.LightBlue && pixel.Tag == null)

{

pixel.BackColor = Color.LightBlue;

pixel.Tag = "cross";

pixel.Invalidate();

hintsUsed++;

}

if (hintsUsed == maxOfHints)

{

buttonHint.BackColor = SystemColors.Window;

hintOrNot = false;

}

}

}

}

}

if (\_solvingProcess == null)

{

UpdateMatrixForCrossword();

}

else

{

UpdateMatrixForSP();

}

}

/// <summary>

/// обновление числовой матрицы после взаимодействия с пользователем

/// </summary>

///

private void checkProgressPercent()

{

int countOfScores = 0; // подсчет правильных совпадений черных клеток с готовым кроссвордом

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

{

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

{

if (matrix[i, j] == valueOfBlack && readyMatrix[i, j] == valueOfBlack)

{

countOfScores++;

}

else if (matrix[i, j] == valueOfBlack && (readyMatrix[i, j] == valueOfWhite || readyMatrix[i, j] == valueOfX))

{

countOfScores--;

}

}

}

\_readyPercent = Convert.ToInt32(Convert.ToDouble(countOfScores) / \_countOfBlack \* 100);

if (\_readyPercent > 0)

{

progressBar.Value = \_readyPercent;

}

else

{

progressBar.Value = 0;

}

}

/// <summary>

/// функция обновления матрицы после действия НЕавторизованного пользователя (работаем с таблицей Crossword)

/// </summary>

private void UpdateMatrixForCrossword()

{

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

// значение закрашенной клетки

if ((matrix[i, j] == valueOfWhite || matrix[i, j] == valueOfX) && pixel.BackColor == Color.Black)

{

matrix[i, j] = valueOfBlack;

}

// значения пустой клетки

if ((matrix[i, j] == valueOfX && pixel.Tag == null) ||

matrix[i, j] == valueOfBlack && pixel.BackColor == Color.LightBlue)

{

matrix[i, j] = valueOfWhite;

}

// значение крестика

if (matrix[i, j] == valueOfWhite && pixel.Tag == "cross")

{

matrix[i, j] = valueOfX;

}

}

}

// обновление подсказок

buttonHint.Text = $"Подсказать {hintsUsed}/{maxOfHints}";

checkProgressPercent();

// проверка на выигрыш

bool winOrNot = true;

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

if (matrix[i, j] == valueOfBlack && readyMatrix[i, j] != valueOfBlack || matrix[i, j] != valueOfBlack && readyMatrix[i, j] == valueOfBlack)

{

winOrNot = false;

}

}

}

if (winOrNot)

{

timer.Enabled = false;

FormSuccess formSuccess = new FormSuccess(\_crossword);

formSuccess.ShowDialog();

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

// делаем некликабельными пиксели

pixel.MouseDown -= Pixel\_MouseDown;

}

}

}

}

/// <summary>

/// функция обновления матрицы после действия авторизованного пользователя (работаем с таблицей SolvingProcess)

/// </summary>

private void UpdateMatrixForSP()

{

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

// значение закрашенной клетки

if ((matrix[i, j] == valueOfWhite || matrix[i, j] == valueOfX) && pixel.BackColor == Color.Black)

{

matrix[i, j] = valueOfBlack;

}

// значения пустой клетки

if ((matrix[i, j] == valueOfX && pixel.Tag == null) ||

matrix[i, j] == valueOfBlack && pixel.BackColor == Color.LightBlue)

{

matrix[i, j] = valueOfWhite;

}

// значение крестика

if (matrix[i, j] == valueOfWhite && pixel.Tag == "cross")

{

matrix[i, j] = valueOfX;

}

}

}

// обновление подсказок

buttonHint.Text = $"Подсказать {hintsUsed}/{maxOfHints}";

checkProgressPercent();

// проверка на выигрыш

bool winOrNot = true;

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

if (matrix[i, j] == valueOfBlack && readyMatrix[i, j] != valueOfBlack || matrix[i, j] != valueOfBlack && readyMatrix[i, j] == valueOfBlack)

{

winOrNot = false;

}

}

}

if (winOrNot)

{

timer.Enabled = false;

\_solvingProcess.StatusOfCrossword = true;

\_solvingProcess.EndTime = DateTime.Now;

Program.context.SaveChanges();

FormSuccess formSuccess = new FormSuccess(\_crossword);

formSuccess.ShowDialog();

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

// делаем некликабельными пиксели

pixel.MouseDown -= Pixel\_MouseDown;

}

}

}

// 1. Преобразуем текущую матрицу в строку прогресса

StringBuilder progressBuilder = new StringBuilder(\_width \* \_height);

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

progressBuilder.Append(matrix[i, j]);

}

}

// 2. Обновляем прогресс в объекте SolvingProcess

\_solvingProcess.Progress = progressBuilder.ToString();

// 3. Обновляем количество использованных подсказок

\_solvingProcess.HintsUsed = hintsUsed;

Program.context.SaveChanges();

}

/// <summary>

/// Обработчик наведения курсора, для хорошей видимости выбираемой клетки

/// </summary>

private void Pixel\_MouseEnter(object sender, EventArgs e)

{

Label pixel = (Label)sender;

if (pixel.BackColor == Color.White)

{

pixel.BackColor = Color.LightBlue;

}

}

/// <summary>

/// Обработчик выхода курсора, и замены цвета на стандартный

/// </summary>

private void Pixel\_MouseLeave(object sender, EventArgs e)

{

Label pixel = (Label)sender;

if (pixel.BackColor == Color.LightBlue)

{

pixel.BackColor = Color.White;

}

}

// Флаг для отслеживания, была ли форма закрыта через кнопку "Выход"

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

{

timer.Enabled = false; // обязательно выключаем таймер, иначе счет будет идти, и счетчики складываться (как это работает хз).

this.Close();

}

/// <summary>

/// Создаём игровое поле и панели с цифрами

/// </summary>

private void createGameMatrix(int width, int height)

{

panel.Controls.Clear(); // Очищаем панель перед созданием новой матрицы

// Определяем максимальное количество чисел в подсказках

int maxRowDigits = 0;

int maxColDigits = 0;

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

maxRowDigits = Math.Max(maxRowDigits, GetHintsForRow(i).Count);

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

maxColDigits = Math.Max(maxColDigits, GetHintsForColumn(j).Count);

// Максимальный размер отступа под цифры

int digitXSize = Math.Max(maxRowDigits, maxColDigits);

// Вычисляем размер клетки так, чтобы все влезало

int pixelSize;

if (Math.Max(height, width) + digitXSize < 13)

{

pixelSize = Math.Min(panel.Width / (height + digitXSize), panel.Height / (width + digitXSize));

panel.Size = new Size((maxRowDigits + height) \* pixelSize, (maxColDigits + width) \* pixelSize);

CenterPanel();

}

else

{

pixelSize = Math.Min(panel.Width / (13), panel.Height / (13));

panel.AutoScroll = true; // Включаем автопрокрутку

panel.AutoScrollMinSize = new Size(0, 0); // Сброс минимального размера

// Вычисляем общий размер кроссворда

int totalWidth = (maxRowDigits + height) \* pixelSize;

int totalHeight = (maxColDigits + width) \* pixelSize;

// Устанавливаем минимальный размер для прокрутки

panel.AutoScrollMinSize = new Size(totalWidth, totalHeight);

}

// Вычисляем корректные отступы для центрирования

int offsetX = 0;

int offsetY = 0;

// Генерация чисел для столбцов (прижимаем к нижнему краю)

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

{

List<int> columnHints = GetHintsForColumn(j);

int startY = offsetY + (maxColDigits - columnHints.Count) \* pixelSize; // Смещение вниз

for (int k = 0; k < columnHints.Count; k++)

{

Label colHintLabel = new Label

{

Name = $"col{j}\_{k}",

Text = columnHints[k].ToString(),

Size = new Size(pixelSize, pixelSize),

Location = new Point((j + maxRowDigits) \* pixelSize + offsetX, startY + (k \* pixelSize)),

TextAlign = ContentAlignment.MiddleCenter,

BackColor = Color.DarkGray,

BorderStyle = BorderStyle.FixedSingle,

Font = new Font("Calibri", 72, FontStyle.Bold)

};

colHintLabel.Click += digit\_Click;

colHintLabel.Paint += digit\_Paint;

AdjustFontSize(colHintLabel, pixelSize, pixelSize);

panel.Controls.Add(colHintLabel);

}

}

// Генерация чисел для строк (прижимаем к правому краю)

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

{

List<int> rowHints = GetHintsForRow(i);

int startX = offsetX + (maxRowDigits - rowHints.Count) \* pixelSize; // Смещение вправо

for (int k = 0; k < rowHints.Count; k++)

{

Label rowHintLabel = new Label

{

Name = $"row{i}\_{k}",

Text = rowHints[k].ToString(),

Size = new Size(pixelSize, pixelSize),

Location = new Point(startX + (k \* pixelSize), (i + maxColDigits) \* pixelSize + offsetY),

TextAlign = ContentAlignment.MiddleCenter,

BackColor = Color.DarkGray,

BorderStyle = BorderStyle.FixedSingle,

Font = new Font("Calibri", 72, FontStyle.Bold)

};

rowHintLabel.Click += digit\_Click;

rowHintLabel.Paint += digit\_Paint;

AdjustFontSize(rowHintLabel, pixelSize, pixelSize);

panel.Controls.Add(rowHintLabel);

}

}

// поиск минимального размера шрифта

float minSize = 72f;

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

{

List<int> rowHints = GetHintsForRow(i);

for (int k = 0; k < rowHints.Count; k++)

{

string rowName = $"row{i}\_{k}";

Label row = panel.Controls.Find(rowName, true).FirstOrDefault() as Label;

minSize = Math.Min(minSize, row.Font.Size);

}

}

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

{

List<int> columnHints = GetHintsForColumn(i);

for (int k = 0; k < columnHints.Count; k++)

{

string colName = $"col{i}\_{k}";

Label col = panel.Controls.Find(colName, true).FirstOrDefault() as Label;

minSize = Math.Min(minSize, col.Font.Size);

}

}

// выравнивание шрифтов

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

{

List<int> rowHints = GetHintsForRow(i);

for (int k = 0; k < rowHints.Count; k++)

{

string rowName = $"row{i}\_{k}";

Label row = panel.Controls.Find(rowName, true).FirstOrDefault() as Label;

row.Font = new Font(row.Font.FontFamily, minSize, row.Font.Style);

}

}

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

{

List<int> columnHints = GetHintsForColumn(i);

for (int k = 0; k < columnHints.Count; k++)

{

string colName = $"col{i}\_{k}";

Label col = panel.Controls.Find(colName, true).FirstOrDefault() as Label;

col.Font = new Font(col.Font.FontFamily, minSize, col.Font.Style);

}

}

// Генерация игрового поля с учетом прогресса

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

{

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

{

Label pixel = new Label

{

Name = $"pixel{i}{j}",

Size = new Size(pixelSize, pixelSize),

Location = new Point((j + maxRowDigits) \* pixelSize + offsetX,

(i + maxColDigits) \* pixelSize + offsetY),

BackColor = Color.White,

BorderStyle = BorderStyle.FixedSingle

};

// Инициализация пикселей в зависимости от прогресса

if (matrix[i, j] == valueOfBlack)

{

pixel.BackColor = Color.Black;

}

else if (matrix[i, j] == valueOfX)

{

pixel.Tag = "cross";

}

pixel.MouseDown += Pixel\_MouseDown;

pixel.MouseEnter += Pixel\_MouseEnter;

pixel.MouseLeave += Pixel\_MouseLeave;

pixel.Paint += digit\_Paint;

panel.Controls.Add(pixel);

}

}

// Заполняем ВСЁ пространство панели клетками (включая подсказки)

for (int y = 0; y < maxColDigits + width; y++)

{

for (int x = 0; x < maxRowDigits + height; x++)

{

// Пропускаем левый верхний угол (где x < maxRowDigits И y < maxColDigits)

if (x < maxRowDigits && y < maxColDigits)

continue;

// Проверяем, не является ли эта клетка уже созданной подсказкой или игровым полем

bool isHintOrPixel = false;

foreach (Control ctrl in panel.Controls)

{

if (ctrl.Location.X == x \* pixelSize && ctrl.Location.Y == y \* pixelSize)

{

isHintOrPixel = true;

break;

}

}

// Если это пустое место - заполняем серой клеткой

if (!isHintOrPixel)

{

Label backgroundCell = new Label

{

Size = new Size(pixelSize, pixelSize),

Location = new Point(x \* pixelSize, y \* pixelSize),

BackColor = Color.LightGray,

BorderStyle = BorderStyle.FixedSingle

};

panel.Controls.Add(backgroundCell);

}

}

}

}

/// <summary>

/// Получает группы подряд идущих единиц для столбца

/// </summary>

private List<int> GetHintsForColumn(int col)

{

List<int> hints = new List<int>();

int count = 0;

for (int i = 0; i < \_width; i++)

{

if (readyMatrix[i, col] == valueOfBlack)

{

count++;

}

else if (count > 0)

{

hints.Add(count);

count = 0;

}

}

if (count > 0) hints.Add(count); // если последняя клетка закрашена в черный и не сработала функция добавления count

return hints;

}

/// <summary>

/// Получает группы подряд идущих единиц для строки

/// </summary>

private List<int> GetHintsForRow(int row)

{

List<int> hints = new List<int>();

int count = 0;

for (int j = 0; j < \_height; j++)

{

if (readyMatrix[row, j] == valueOfBlack)

{

count++;

}

else if (count > 0)

{

hints.Add(count);

count = 0;

}

}

if (count > 0) hints.Add(count); // если последняя клетка закрашена в черный и не сработала функция добавления count

return hints;

}

/// <summary>

/// Подгоняет размер шрифта, чтобы все цифры поместились в клетках

/// </summary>

void AdjustFontSize(Label label, int maxWidth, int maxHeight)

{

using (Graphics g = label.CreateGraphics())

{

float fontSize = label.Font.Size;

Font testFont = new Font(label.Font.FontFamily, fontSize, label.Font.Style);

while (fontSize > 1)

{

SizeF textSize = g.MeasureString(label.Text, testFont);

if (textSize.Width <= maxWidth - 5 && textSize.Height <= maxHeight - 5)

break; // Если текст влезает, выходим

fontSize--; // Уменьшаем шрифт

testFont = new Font(label.Font.FontFamily, fontSize, label.Font.Style);

}

label.Font = testFont; // Применяем найденный размер шрифта

}

}

/// <summary>

/// клик по цифре подсказки, который меняет флаг для закраски крестиком

/// </summary>

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

{

Label digit = (Label)sender;

if (digit.Tag == null)

{

digit.Tag = "ready"; // или true/false, или новый объект

}

else if (digit.Tag == "ready")

{

digit.Tag = null;

}

digit.Invalidate(); // Перерисовать

}

private void GameForm\_Load\_1(object sender, EventArgs e)

{

}

// выравнивание панели кроссворда

private void CenterPanel()

{

// Получаем размеры формы и панели

int formWidth = panel2.Width; // Ширина клиентской области формы

int formHeight = panel2.Height; // Высота клиентской области формы

int panelWidth = panel.Width; // Ширина панели

int panelHeight = panel.Height; // Высота панели

// Вычисляем координаты для центрирования панели

int x = (formWidth - panelWidth) / 2;

// Устанавливаем положение панели

panel.Location = new Point(x, 0);

// Центрируем все элементы внутри панели (например, игровое поле и подсказки)

foreach (Control ctrl in panel.Controls)

{

// Если нужно центрировать элементы внутри панели:

int elementX = (panel.Width - ctrl.Width) / 2;

int elementY = (panel.Height - ctrl.Height) / 2;

// Устанавливаем координаты для каждого элемента

ctrl.Location = new Point(elementX, elementY);

}

}

/// <summary>

/// закраска клеток крестиком

/// </summary>

private void digit\_Paint(object sender, PaintEventArgs e)

{

var label = (Label)sender;

if (label.Tag?.ToString() == "ready" || label.Tag?.ToString() == "cross")

{

using (Pen pen = new Pen(Color.Black, 2))

{

// Получаем размеры клиентской области метки (без учета границ и отступов)

int clientWidth = label.ClientSize.Width;

int clientHeight = label.ClientSize.Height;

// Центр метки

int centerX = clientWidth / 2;

int centerY = clientHeight / 2;

// Размер крестика, который зависит от меньшего размера стороны

int crossSize = Math.Min(clientWidth, clientHeight) / 2;

// Рисуем крестик

e.Graphics.DrawLine(pen, centerX - crossSize, centerY - crossSize, centerX + crossSize, centerY + crossSize);

e.Graphics.DrawLine(pen, centerX + crossSize, centerY - crossSize, centerX - crossSize, centerY + crossSize);

}

}

}

private void timer\_Tick(object sender, EventArgs e)

{

if (\_solvingProcess != null)

{

// таймер для авторизованного пользвоателя

\_solvingProcess.LeadTime++;

var leadTime = (int)\_solvingProcess.LeadTime;

// Часы

int hours = leadTime / 3600;

// Десятки часов

A = hours / 10;

// Единицы часов

B = hours % 10;

// Минуты

int minutes = (leadTime % 3600) / 60;

// Десятки минут

C = minutes / 10;

// Единицы минут

D = minutes % 10;

// Секунды

int seconds = leadTime % 60;

// Десятки секунд

E = seconds / 10;

// Единицы секунд

F = seconds % 10;

labelTime.Text = $"{A}{B}:{C}{D}:{E}{F}";

Program.context.SaveChanges();

}

else

{

// таймер для неавторизованного пользвоателя

// распределяем таймер по цифрам

F++;

if (F > 9)

{

F = 0;

E++;

if (E > 5)

{

E = 0;

D++;

if (D > 9)

{

D = 0;

C++;

if (C > 5)

{

C = 0;

B++;

if (B > 9)

{

B = 0;

A++;

}

}

}

}

}

labelTime.Text = $"{A}{B}:{C}{D}:{E}{F}";

}

}

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

{

// Подтверждение действия у пользователя

var result = MessageBox.Show("Вы действительно хотите сбросить текущее решение?",

"Подтверждение сброса",

MessageBoxButtons.YesNo,

MessageBoxIcon.Question);

if (result != DialogResult.Yes) return;

if (\_solvingProcess != null)

{

// Сбрасываем статус решения

\_solvingProcess.StatusOfCrossword = false;

// Создаем строку прогресса из нулей

\_solvingProcess.Progress = new string('0', \_width \* \_height);

// Обновляем таймер

\_solvingProcess.StartTime = DateTime.Now;

\_solvingProcess.LeadTime = 0;

\_solvingProcess.HintsUsed = 0;

\_solvingProcess.EndTime = null;

// Обновляем матрицу

}

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

// Сбрасываем значение в матрице

matrix[i, j] = 0;

// Находим соответствующий пиксель

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

if (pixel != null)

{

// Сбрасываем визуальное состояние

pixel.BackColor = Color.White;

pixel.Tag = null;

pixel.Invalidate();

// Обновляем подписки на события (более надежный способ)

pixel.MouseDown -= Pixel\_MouseDown;

pixel.MouseDown += Pixel\_MouseDown;

}

}

}

// сброс таймера для неавторизованного пользователя

if (\_solvingProcess == null)

{

A = 0; // десятки часов таймера

B = 0; // часы таймера

C = 0; // десятки минут таймера

D = 0; // минуты таймер

E = 0; // десятки секунд таймера

F = 0; // секунды таймера

}

hintsUsed = 0;

if (\_solvingProcess != null)

{

// Обновляем данные в базе

UpdateMatrixForSP();

Program.context.SaveChanges();

}

else

{

UpdateMatrixForCrossword();

}

timer.Enabled = true;

labelTime.Text = "00:00:00";

}

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

{

if (hintsUsed < maxOfHints)

{

if (!hintOrNot)

{

buttonHint.BackColor = Color.LightGreen;

hintOrNot = true;

}

else

{

buttonHint.BackColor = SystemColors.Window;

hintOrNot = false;

}

}

}

}

}

* + - **CreateForm.cs:**

using Nonograms\_1.\_1.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.ListView;

namespace Nonograms\_1.\_1.Forms

{

public partial class CreateForm : Form

{

private int \_width;

private int \_height;

private int[,] \_matrix; // заполняемая матрица

private int \_levelDifficult = 0;

private string matrixToString;

private int valueOfX = 2;

private int valueOfBlack = 1;

private int valueOfWhite = 0;

public CreateForm()

{

InitializeComponent();

var difficults = Program.context.Difficults.OrderBy(d=>d.DifficultID).ToList();

foreach( var difficult in difficults )

{

comboBoxDifficult.Items.Add(difficult.DifficultName.ToString());

}

// инициализируем компоненты матрицы

\_width = Convert.ToInt32(numericUpDownLength.Value); // Размер матрицы N x N

\_height = Convert.ToInt32(numericUpDownHeight.Value); // Размер матрицы N x N

// размер заполняемой матрицы

\_matrix = new int[\_width, \_height];

// создаем матрицу(игровое поле) из Label

createGameMatrix(\_width, \_height);

}

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

{

}

private void Pixel\_MouseDown(object sender, MouseEventArgs e)

{

// проверка нажатия левой кнопки по пикселю

if (e.Button == MouseButtons.Left)

{

Label pixel = (Label)sender;

pixel.Tag = null;

if (pixel.BackColor == Color.LightBlue)

{

pixel.BackColor = Color.Black; // Меняем цвет пикселя

}

// проверка повторного нажатия по пикселю

else if (pixel.BackColor == Color.Black)

{

pixel.BackColor = Color.LightBlue;

}

}

// проверка нажатия правой кнопки по пикселю

if (e.Button == MouseButtons.Right)

{

Label pixel = (Label)sender;

if (pixel.Tag == null)

{

pixel.BackColor = Color.LightBlue;

// Для показа крестика:

pixel.Tag = "cross";

pixel.Invalidate(); // Перерисовываем

}

// проверка повторного нажатия по пикселю

else if (pixel.Tag == "cross")

{

pixel.BackColor = Color.LightBlue;

//сброс крестика

pixel.Tag = null;

pixel.Invalidate();

}

}

UpdateMatrix();

}

/// <summary>

/// обновление числовой матрицы после взаимодействия с пользователем

/// </summary>

private void UpdateMatrix()

{

matrixToString = null;

for (int i = 0; i < \_width; i++)

{

for (int j = 0; j < \_height; j++)

{

string pixelName = $"pixel{i}{j}";

Label pixel = panel.Controls.Find(pixelName, true).FirstOrDefault() as Label;

// значение закрашенной клетки

if ((\_matrix[i, j] == valueOfWhite || \_matrix[i, j] == valueOfX) && pixel.BackColor == Color.Black)

{

\_matrix[i, j] = valueOfBlack;

}

// значения пустой клетки

if ((\_matrix[i, j] == valueOfX && pixel.Tag == null) ||

\_matrix[i, j] == valueOfBlack && pixel.BackColor == Color.LightBlue)

{

\_matrix[i, j] = valueOfWhite;

}

// значение крестика

if (\_matrix[i, j] == valueOfWhite && pixel.Tag == "cross")

{

\_matrix[i, j] = valueOfX;

}

matrixToString += \_matrix[i, j];

}

}

}

// Обработчик наведения курсора

private void Pixel\_MouseEnter(object sender, EventArgs e)

{

Label pixel = (Label)sender;

if (pixel.BackColor == Color.White)

{

pixel.BackColor = Color.LightBlue;

}

}

// Обработчик ухода курсора

private void Pixel\_MouseLeave(object sender, EventArgs e)

{

Label pixel = (Label)sender;

if (pixel.BackColor == Color.LightBlue)

{

pixel.BackColor = Color.White;

}

}

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

{

this.Close();

}

/// <summary>

/// создаем матрицу(игровое поле) из Label

/// </summary>

///

private void createGameMatrix(int width, int height)

{

// само игровое поле, с которым взаимодействует пользователь

int pixelSize; // Размер одного "пикселя"

if (width > height)

{

pixelSize = panel.Width / width;

}

else

{

pixelSize = panel.Width / height;

}

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

{

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

{

Label pixel = new Label();

pixel.Name = $"pixel{i}{j}";

pixel.Size = new Size(pixelSize, pixelSize);

pixel.Location = new Point(j \* pixelSize, i \* pixelSize);

pixel.BackColor = Color.White;

pixel.FlatStyle = FlatStyle.Flat;

pixel.BorderStyle = BorderStyle.FixedSingle;

pixel.ForeColor = Color.Black;

// обработчики

pixel.MouseDown += Pixel\_MouseDown; // Обработка клика

pixel.MouseEnter += Pixel\_MouseEnter; // Обработка наведения курсора

pixel.MouseLeave += Pixel\_MouseLeave; // Обработка выведения кусора

pixel.Paint += (sender, e) =>

{

var label = (Label)sender;

if (label.Tag?.ToString() == "cross") // Используем Tag как флаг

{

using (Pen pen = new Pen(Color.Black, 2))

{

e.Graphics.DrawLine(pen, 0, 0, label.Width, label.Height);

e.Graphics.DrawLine(pen, label.Width, 0, 0, label.Height);

}

}

};

panel.Controls.Add(pixel);

}

}

}

private void numericUpDownLength\_ValueChanged(object sender, EventArgs e)

{

panel.Controls.Clear(); // Очищаем панель перед созданием новых кнопок

\_width = Convert.ToInt32(numericUpDownLength.Value); // Размер матрицы N x N

createGameMatrix(\_width, \_height);

\_matrix = new int[\_width, \_height];

}

private void numericUpDownHeight\_ValueChanged(object sender, EventArgs e)

{

;

panel.Controls.Clear(); // Очищаем панель перед созданием новых кнопок

\_height = Convert.ToInt32(numericUpDownHeight.Value); // Размер матрицы N x N

createGameMatrix(\_width, \_height);

\_matrix = new int[\_width, \_height];

}

private void comboBoxDifficult\_SelectedIndexChanged(object sender, EventArgs e)

{

\_levelDifficult = comboBoxDifficult.SelectedIndex + 1;

}

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

{

string name = textBoxName.Text;

string matrix = matrixToString;

int width = \_width;

int height = \_height;

int difficult = \_levelDifficult;

bool isMatrixFilled = false;

foreach (var item in matrix)

{

if (item == '1')

{

isMatrixFilled = true;

}

}

if (isMatrixFilled)

{

if (name != null)

{

if (difficult > 0)

{

Crossword crossword = new Crossword()

{

Name = name,

Matrix = matrix,

Width = width,

Height = height,

DifficultID = difficult,

};

Program.context.Crosswords.Add(crossword);

Program.context.SaveChanges();

MessageBox.Show("Успешное добавление кроссворда!", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

}

}

}

}

* + - **FormSuccess.cs:**

using Nonograms\_1.\_1.Models;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Nonograms\_1.\_1.forms

{

public partial class FormSuccess : Form

{

private int[,] readyMatrix; // готовая матрица

// значения клеток в цифрах

private int valueOfBlack = 1; // черная клетка

public FormSuccess(Crossword crossword)

{

InitializeComponent();

int N = crossword.Width;

int M = crossword.Height;

readyMatrix = new int[N, M];

string matrixString = crossword.Matrix;

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

{

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

{

int index = i \* M + j;

if (index < matrixString.Length)

{

readyMatrix[i, j] = int.Parse(Convert.ToString(crossword.Matrix[index]));

}

}

}

createGameMatrix(N, M);

labelNameOfCrossword.Text = crossword.Name;

}

private void createGameMatrix(int N, int M)

{

panel.Controls.Clear();

int pixelWidth = panel.Width / M;

int pixelHeight = panel.Height / N;

int pixelSize = Math.Min(pixelWidth, pixelHeight);

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

{

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

{

Label pixel = new Label();

pixel.Name = $"pixel{i}{j}";

pixel.Size = new Size(pixelSize, pixelSize);

pixel.Location = new Point(j \* pixelSize, i \* pixelSize);

pixel.BackColor = (readyMatrix[i, j] == valueOfBlack) ? Color.Black : Color.White;

pixel.BorderStyle = BorderStyle.FixedSingle;

panel.Controls.Add(pixel);

}

}

}

private void flowLayoutPanel1\_Paint(object sender, PaintEventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

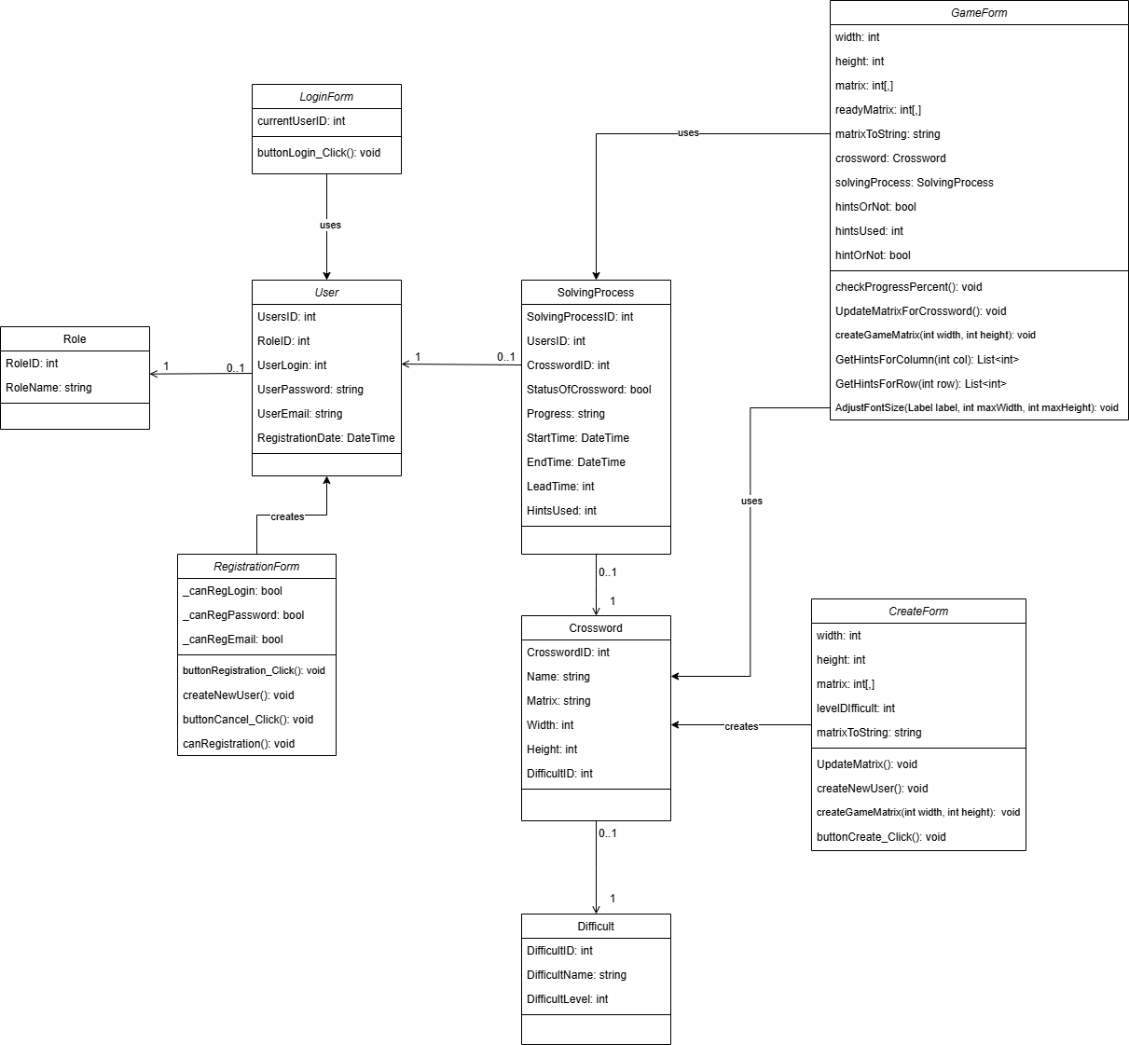
1. **Диаграммы:**

* Диаграмма сущность-связь:

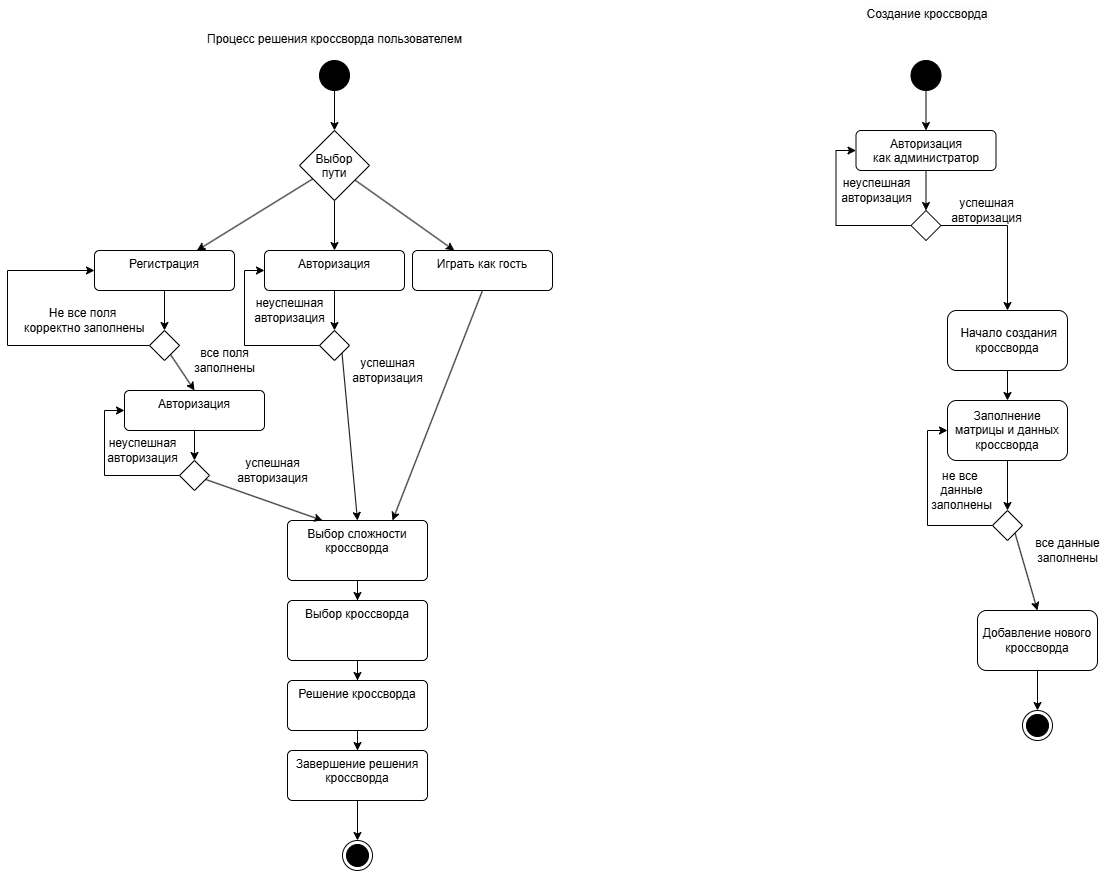
Изображение выглядит как снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

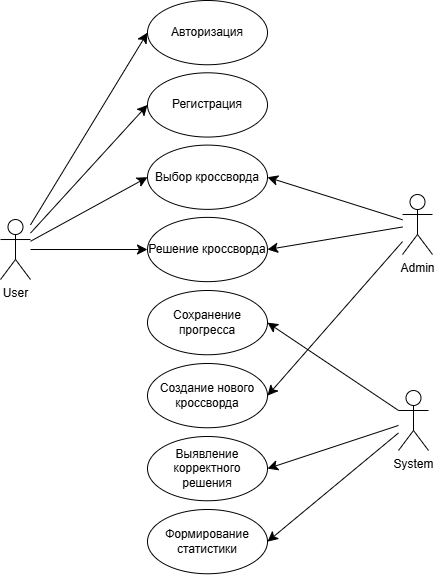
* Диаграмма классов:



* Диаграммы деятельности:



* Диаграмма прецедентов:



## **Источники, используемые при разработке**

1. Документация .NET Framework.
2. MS SQL Server Developer Guide.
3. ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы».