**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»**

**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ**

**ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І**

**СИСТЕМ**

**КУРСОВА РОБОТА**

**з дисципліни**

**«Об’єктно-орієнтоване програмування»**

**спеціальність 123 «Комп’ютерна інженерія »**

**Студента групи 201-ТК**

Керівник роботи

старший викладач кафедри комп’ютерних та інформаційних технологій і систем  
Демиденко М.І.

Полтава – 2024

**ЗМІСТ**

[ВСТУП 3](#_Toc165818424)

[РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ 4](#_Toc165818425)

[1.1 Мова програмування C# 4](#_Toc165818426)

[1.2 Поняття судоку 5](#_Toc165818427)

[1.3 Основна логіка гри 6](#_Toc165818428)

[1.3.1 Створення початкової схеми гри 6](#_Toc165818429)

[1.3.2 Перемішування судоку. 7](#_Toc165818430)

[1.3.3 Випадкове приховування значень. 7](#_Toc165818431)

[1.3.3 Заповнення пустих комірок 8](#_Toc165818432)

[1.3.4 Здійснення перевірки введених значень. 9](#_Toc165818433)

[РОЗДІЛ 2 ОПИС ФУНКЦІОНАЛУ КЛАСІВ 11](#_Toc165818434)

[2.1 Класи основного проекту Windows Forms 11](#_Toc165818435)

[РОЗДІЛ 3 ДИЗАЙН КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ 12](#_Toc165818436)

[РОЗДІЛ 4 ВИХІДНИЙ КОД ПРОЕКТУ 13](#_Toc165818437)

[4.1 Код основної форми 13](#_Toc165818438)

[ВИСНОВКИ 17](#_Toc165818439)

# ВСТУП

Об'єктно-орієнтоване програмування (ООП, іноді також об'єктно-зорієнтоване програмування) – це парадигма програмування, яка розглядає програму як множину "об'єктів", що взаємодіють між собою.

Хоча ця парадигма з'явилась ще у 1960-х роках, вона не отримала широкого застосування до 1990-х. З розвитком комп'ютерів та мереж програмування набуло нового значення. Сьогодні багато мов програмування підтримують ООП або навіть є цілком об'єктно-орієнтованими.

Основні принципи об'єктно-орієнтованого програмування базуються на інкапсуляції, наслідуванні, абстракції та поліморфізмі.

* **Інкапсуляція** – приховування деталей роботи класів від об'єктів, які їх використовують.
* **Наслідування** – можливість класу мати "підкласи", що розширюють функціональність батьківського класу.
* **Абстракція** – спрощення складної дійсності шляхом моделювання класів.
* **Поліморфізм** – залежність поведінки від класу, що викликає дію.

Об'єктно-орієнтоване програмування визначає програму як набір об'єктів, які взаємодіють за допомогою повідомлень. Кожен об'єкт є незалежним автоматом з власним станом та поведінкою.

За словами Алана Кея, одного з "батьків-засновників" ООП, основні принципи включають у себе те, що всі об'єкти є представниками класів і взаємодіють за допомогою обміну даними.

Отже, об'єктно-орієнтоване програмування дозволяє створювати програми, що складаються з об'єктів, які мають свій стан та поведінку, і взаємодіють за допомогою повідомлень.

# РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ

Розробити гру “Судоку” мовою програмування C#.

## Мова програмування C#

C# (вимовляється "Сі-шарп") — це об'єктно-орієнтована мова програмування з безпечною системою типізації, розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research для платформи .NET.

Синтаксис C# подібний до мов програмування С++ і Java. Вона має строгу статичну типізацію та підтримує поліморфізм, перевантаження операторів, атрибути, події, властивості, винятки та інші концепції.

Мова С# успадкувала багато концепцій від своїх попередників, таких як С++, Object Pascal, Модула і Smalltalk. Водночас, вона виключає деякі моделі, які вважались проблематичними, наприклад, множинне успадкування класів, що відсутнє у C#.

Станом на 2023 рік, остання стабільна версія мови C# — це 11.0, яка була випущена в 2022 році як частина платформи .NET 7.0.

Символ # у назві мови можна інтерпретувати як дві пари плюсів ++, що вказує на її розвиток від С++, і як музичний символ дієз, що спільно з буквою C утворює назву мови. Це пояснення символу # було запропоноване Microsoft.

Хоча символ # (октоторп) відрізняється від символу дієз ♯, Microsoft рекомендує використовувати саме символ # для назви мови, і цей стандарт прийнято вважати офіційним.

## Поняття судоку

Судоку - це головоломка, яка полягає в заповненні сітки розміром 9x9 клітинок. Сітка поділена на 9 рядків, 9 стовпців та 9 блоків розміром 3x3. Мета гри полягає в тому, щоб заповнити кожну клітинку сітки цифрою від 1 до 9 таким чином, щоб кожен рядок, кожний стовпчик та кожен блок містили всі цифри від 1 до 9 без повторень.

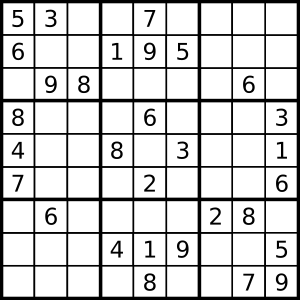


Рисунок 1.1 – Вигляд судоку

На початку гри на сітці вже можуть бути деякі цифри, які відомі. Гравець повинен заповнити пусті клітинки так, щоб дотримуватися правил гри. Головоломку можна розв'язувати методом проб і помилок, присвоюючи цифри клітинкам одну за одною, або використовуючи різні логічні стратегії, такі як "єдина можливість" або "пошук потенційних цифр".

Судоку є популярною головоломкою, яка зазвичай вирішується без необхідності випадкових виборів або використання спеціальних алгоритмів. Вона сприяє розвитку логічного мислення, уваги та витривалості.

## Основна логіка гри

1.3.1 Створення початкової схеми гри. У грі судоку кожен ряд, стовпець і блок 3x3 повинні містити унікальні цифри від 1 до 9 без повторень. Для створення початкової схеми гри, її можна скласти, враховуючи ці правила. Перший ряд записується по порядку від 1 до 9. Далі, другий та третій ряди записуються зі зміщенням на 3 позиції вправо. Послідовні ряди складаються за аналогічним принципом, зі зміщенням на 1 позицію вправо. Таким чином, формується базова схема гри, яку потім можна перемішувати для створення нових головоломок.

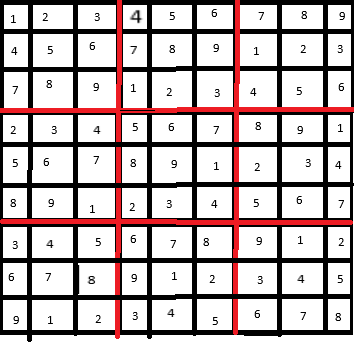


Рисунок 1.2 – початкова схема гри

## 1.3.2 Перемішування судоку.

Перемішування рядків, стовпців та блоків: У цьому методі рядки, стовпці та блоки перемішуються незалежно один від одного, але так, щоб кожна клітинка залишалася в межах свого блоку. Це забезпечує збереження унікальності кожного ряду, стовпця та блоку.

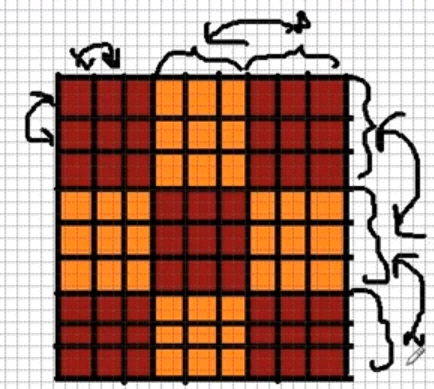


Рисунок 1.3 – Перемішування судоку

## 1.3.3 Випадкове приховування значень.

Випадкове приховування значень після перемішування - це додатковий етап у процесі створення головоломки судоку. Після того, як базова схема судоку була сформована і тасована за допомогою одного з методів, можна випадковим чином вибрати деякі клітинки для приховування значень.



Рисунок 1.4 - Випадкове приховування значень

В результаті цього процесу отримується готова головоломка судоку, яка має випадково розташовані відомі та невідомі значення, готова до розв'язання гравцем.

## 1.3.3 Заповнення пустих комірок

Після натискання на пусту комірку, спочатку вона заповнюється значенням одиниці (або будь-яким іншим початковим значенням, яке відповідає правилам гри). Після цього кожен наступний натиск на цю комірку змінює її значення, додаючи до поточного значення одиницю. Наприклад, якщо початкове значення було одиницею, наступне натискання змінить його на двійку, а наступне - на трійку, і так далі.

Цей процес дозволяє гравцям легко та зручно заповнювати комірки головоломки судоку, додаючи значення послідовно без необхідності вручну вводити кожне значення. Такий метод дозволяє гравцям швидко переглядати та виправляти свої введені значення, якщо потрібно.

Цей процес може бути особливо корисним для мобільних або онлайн версій гри, де введення значень відбувається за допомогою сенсорного екрану або миші. Він додає зручність та ефективність у користуванні, роблячи гру більш приємною для гравців.



Рисунок 1.5 - Заповнення пустих комірок

## 1.3.4 Здійснення перевірки введених значень.

Після того як гравець завершив заповнення головоломки судоку, важливо здійснити перевірку введених значень зі схемою гри, яка була задана перед приховуванням. Цей процес перевірки гарантує, що гравець розв'язав головоломку правильно та без помилок.

Перевірка зіставлення з вихідною схемою: Після того, як кожна клітина була перевірена на відповідність правилам судоку, вона порівнюється зі значенням, яке було задане в початковій схемі перед приховуванням. Це дозволяє перевірити, чи введені гравцем значення співпадають з оригінальними значеннями головоломки.

Відображення результатів: Після завершення перевірки може бути показано повідомлення про те, чи було розв'язано головоломку правильно, чи ні.

Цей процес перевірки допомагає гравцям переконатися, що вони правильно розв'язали головоломку судоку.

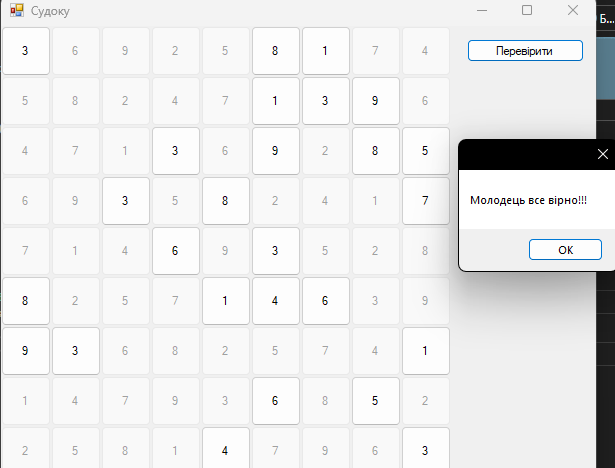


Рисунок 1.6 - перевірка введених значень

Та після перевірки якщо все вірно створюватиметься нова гра.

# РОЗДІЛ 2 ОПИС ФУНКЦІОНАЛУ КЛАСІВ

Дана курсова робота включає в себе один проект в середовищі розробки Visual Studio 2022: проект Windows Forms (.NET Framework 4.8) під назвою “rgr”.

## 2.1 Класи основного проекту Windows Forms

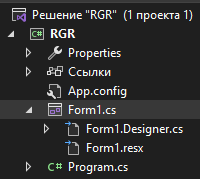


Рисунок 2.1 – Структура проекту Windows Forms

Файл Form1.cs – форма основного вікна програми, клас якої описує події натискання бокових вкладок (кнопок).

Клас Program.cs – точка входу програми.

# РОЗДІЛ 3 ДИЗАЙН КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ

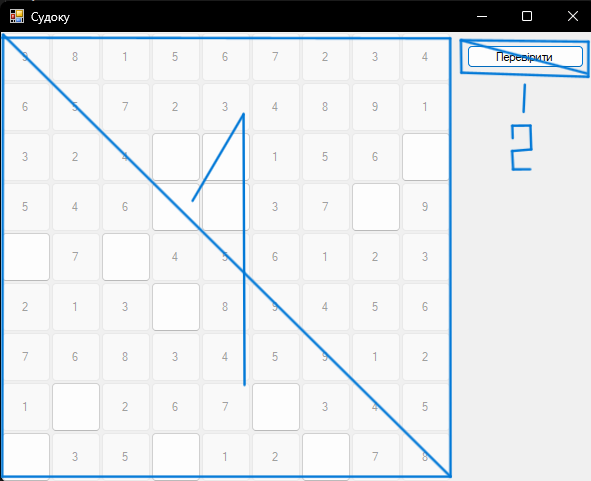


Рисунок 3.1 – Дизайн основної форми програми

1 – Поле гри 2 – кнопка перевірки (рис.3.1)

# РОЗДІЛ 4 ВИХІДНИЙ КОД ПРОЕКТУ

## 4.1 Код основної форми

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 RGR

{

public partial class Form1 : Form

{

const int n = 3;

const int sizeButton = 50;

public int[,] map = new int[n \* n, n \* n];

public Button[,] buttons = new Button[n \* n, n \* n];

public Form1()

{

InitializeComponent();

GenerateMap();

}

public void GenerateMap()

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

map[i, j] = (i \* n + i / n + j) % (n \* n) + 1;

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

}

}

Random r = new Random();

for (int i = 0; i < 10; i++)

{

ShuffleMap(r.Next(0, 5));

}

CreateMap();

HideCells();

}

public void HideCells()

{

int N = 1;

Random r = new Random();

while (N > 0)

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

int a = r.Next(0, 4);

buttons[i, j].Text = a == 0 ? "" : buttons[i, j].Text;

buttons[i, j].Enabled = a == 0 ? true : false;

if (a == 0)

N--;

}

}

}

}

public void ShuffleMap(int i)

{

switch (i)

{

case 0:

MatrixTransposition();

break;

case 1:

SwapRowsInBlock();

break;

case 2:

SwapColumnsInBlock();

break;

case 3:

SwapBlocksInRows();

break;

case 4:

SwapBlocksInColumns();

break;

default:

MatrixTransposition();

break;

}

}

public void SwapBlocksInColumns()

{

Random r = new Random();

var block1 = r.Next(0, n);

var block2 = r.Next(0, n);

while (block1 == block2)

block2 = r.Next(0, n);

block1 \*= n;

block2 \*= n;

for (int i = 0; i < n \* n; i++)

{

var k = block2;

for (int j = block1; j < block1 + n; j++)

{

var temp = map[i, j];

map[i, j] = map[i, k];

map[i, k] = temp;

k++;

}

}

}

public void SwapBlocksInRows()

{

Random r = new Random();

var block1 = r.Next(0, n);

var block2 = r.Next(0, n);

while (block1 == block2)

block2 = r.Next(0, n);

block1 \*= n;

block2 \*= n;

for (int i = 0; i < n \* n; i++)

{

var k = block2;

for (int j = block1; j < block1 + n; j++)

{

var temp = map[j, i];

map[j, i] = map[k, i];

map[k, i] = temp;

k++;

}

}

}

public void SwapRowsInBlock()

{

Random r = new Random();

var block = r.Next(0, n);

var row1 = r.Next(0, n);

var line1 = block \* n + row1;

var row2 = r.Next(0, n);

while (row1 == row2)

row2 = r.Next(0, n);

var line2 = block \* n + row2;

for (int i = 0; i < n \* n; i++)

{

var temp = map[line1, i];

map[line1, i] = map[line2, i];

map[line2, i] = temp;

}

}

public void SwapColumnsInBlock()

{

Random r = new Random();

var block = r.Next(0, n);

var row1 = r.Next(0, n);

var line1 = block \* n + row1;

var row2 = r.Next(0, n);

while (row1 == row2)

row2 = r.Next(0, n);

var line2 = block \* n + row2;

for (int i = 0; i < n \* n; i++)

{

var temp = map[i, line1];

map[i, line1] = map[i, line2];

map[i, line2] = temp;

}

}

public void MatrixTransposition()

{

int[,] tMap = new int[n \* n, n \* n];

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

tMap[i, j] = map[j, i];

}

}

map = tMap;

}

public void CreateMap()

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

Button = new Button();

buttons[i, j] = button;

button.Size = new Size(sizeButton, sizeButton);

button.Text = map[i, j].ToString();

button.Click += OnCellPressed;

button.Location = new Point(j \* sizeButton, i \* sizeButton);

this.Controls.Add(button);

}

}

}

public void OnCellPressed(object sender, EventArgs e)

{

Button pressedButton = sender as Button;

string buttonText = pressedButton.Text;

if (string.IsNullOrEmpty(buttonText))

pressedButton.Text = "1";

else

{

int num = int.Parse(buttonText);

num++;

if (num == 10)

num = 1;

pressedButton.Text = num.ToString();

}

}

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

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

var btnText = buttons[i, j].Text;

if (btnText != map[i, j].ToString())

{

MessageBox.Show("Є помилки");

return;

}

}

}

MessageBox.Show("Молодець все вірно!!!");

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

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

}

}

GenerateMap();

}

}

}

# ВИСНОВКИ

Метою даного проекту була розробка гри "Судоку" з використанням об'єктно-орієнтованого підходу програмування мовою C#. Основною метою гри було створення програми, яка надає користувачам можливість грати в судоку, дотримуючись правил гри та насолоджуючись процесом розв'язання головоломки.

У цій розрахунково графічній роботі було використано об'єктно-орієнтований підхід до програмування, що дозволило зручно моделювати об'єкти та їх взаємодію у грі. Кожна клітина гри, рядок, стовпець та блок були представлені в програмі як окремі об'єкти, які взаємодіяли один з одним відповідно до правил судоку.

За допомогою об'єктно-орієнтованого підходу було досягнуто наступних результатів:

* Моделювання гри "Судоку" у вигляді об'єктів, що дозволило зручно працювати з різними елементами головоломки.
* Реалізація логіки гри з використанням класів та методів, що спростило розв'язання завдань та перевірку правильності ходу гри.
* Створення графічного інтерфейсу користувача, який спрощує взаємодію з програмою та надає зручність в користуванні.

Отже, об'єктно-орієнтований підхід дозволив створити гру "Судоку", яка надає користувачам можливість насолоджуватися процесом гри та розв'язувати цікаві головоломки.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація по мові програмування C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/
2. Об'єктно-орієнтоване програмування [Електронний ресурс]. – Режим доступу: https://shorturl.at/cjzU3
3. Мова програмування C# [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/C\_Sharp
4. Файли проекту курсової роботи [Електронний ресурс]. – Режим доступу: https://github.com/Fristik2004/OOP\_RGR