**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 122 «Комп’ютерні науки »**

**Студента групи 201-ТН Білоконя Валентина Андрійовича**

Керівник роботи

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

Полтава – 2023

**Зміст**

[**Вступ 3**](#_Toc136207838)

[**РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ 5**](#_Toc136207839)

[**1.1 Мова програмування C# 5**](#_Toc136207840)

[**1.2 Етіологія стригучого лишаю 6**](#_Toc136207841)

[**1.3 Побудова незараженої матриці 7**](#_Toc136207842)

[**РОЗДІЛ 2 ОПИС ФУНКЦІОНАЛУ КЛАСІВ 9**](#_Toc136207843)

[**2.1 Класи основного проекту Windows Forms 9**](#_Toc136207844)

[**РОЗДІЛ 3 ДИЗАЙН КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ 13**](#_Toc136207845)

[**РОЗДІЛ 4 ВИХІДНИЙ КОД ПРОЕКТУ 15**](#_Toc136207846)

[**4.1 Код основної форми 15**](#_Toc136207847)

[**4.2 Код класу Ringworm 23**](#_Toc136207848)

[**Висновки 25**](#_Toc136207849)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 26**](#_Toc136207850)

# **Вступ**

Об'єктно-орієнтоване програмування (ООП, іноді об'єктно-зорієнтоване програмування; від англ. Object-oriented programming, OOP) — одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою.

Попри те, що ця парадигма з'явилась в 1960-х роках, вона не мала широкого застосування до 1990-х, коли розвиток комп'ютерів та комп'ютерних мереж дав змогу писати надзвичайно об'ємне і складне програмне забезпечення, що змусило переглянути підходи до написання програм. Сьогодні багато мов програмування або підтримують ООП (PHP, Lua), або ж є цілком об'єкто-орієнтованими (зокрема, Java, C#, C++, Python, Ruby, Swift).

Об'єктно-орієнтоване програмування сягає своїм корінням до створення мови програмування Симула в 1960-х роках, одночасно з посиленням дискусій про кризу програмного забезпечення. Через ускладнення апаратного та програмного забезпечення було дуже важко зберегти якість програм. Об'єкто-орієнтоване програмування частково розв'язує цю проблему шляхом наголошення на модульності програми. [1]

Дана парадигма програмування базується на чотирьох основних принципах: інкапсуляція, наслідування, абстракція та поліморфізм.

Інкапсуляція – це приховування деталей про роботу класів від об'єктів, що їх використовують або надсилають їм повідомлення. Інкапсуляція досягається шляхом вказування, які класи можуть звертатися до членів об'єкта. Як наслідок, кожен об'єкт надає кожному іншому класу певний інтерфейс — члени, доступні іншим класам. Інкапсуляція потрібна для того, аби запобігти використанню користувачами інтерфейсу тих частин реалізації, які, швидше за все, будуть змінюватись. Це дасть змогу полегшити внесення змін без потреби змінювати й користувачів інтерфейсу. [1]

Наслідування означає, що клас може мати «підкласи», спеціалізовані, розширені версії надкласу. Можуть також утворюватись цілі дерева успадкування. Успадкування може бути одиничне (один безпосередній батьківський клас) та множинне (кілька батьківських класів). Це залежить від вибору програміста, який реалізовує клас та мови програмування. Так, наприклад, в мові C# дозволене лише одинарне успадкування, а в С++ і те, й інше. [1]

Абстракція – це спрощення складної дійсності шляхом моделювання класів, що відповідають проблемі, та використання найприйнятнішого рівня деталізації окремих аспектів проблеми. [1]

Поліморфізм означає залежність поведінки від класу, в якому ця поведінка викликається, тобто, два або більше класів можуть реагувати по-різному на однакові повідомлення. На практиці це реалізовується шляхом реалізації ряду підпрограм (функцій, процедур, методи тощо) з однаковими іменами, але з різними параметрами. Залежно від того, що передається, вибирається відповідна підпрограма.

На відміну від традиційних поглядів, коли програму розглядали як набір підпрограм або як перелік інструкцій комп'ютеру, об’єктно-орієнтовані програми можна вважати сукупністю об'єктів. Відповідно до парадигми об'єктно-орієнтованого програмування, кожен об'єкт здатний отримувати повідомлення, обробляти дані, та надсилати повідомлення іншим об'єктам. Кожен об'єкт — своєрідний незалежний автомат з окремим призначенням та відповідальністю. [1]

# **РОЗДІЛ 1**

# **ПОСТАНОВКА ЗАДАЧІ**

Завдання про інфекцію стригучого лишаю (Ван Тассел Д. Стиль, розробка, ефективність, налагодження й випробування програм. - М.:Мир, 1981). Промоделювати процес поширення інфекції — стригучого лишаю по ділянці шкіри розміром n х n (n — непарне) клітинок. Передбачається, що вихідною зараженою клітинкою шкіри є центральна. У кожний інтервал часу уражена інфекцією клітинка може з імовірністю 0,5 заразити кожну із сусідніх здорових клітинок. По закінченні шести одиниць часу заражена клітинка стає несприйнятливою до інфекції імунітет, що виник, діє протягом наступних чотирьох одиниць часу, а потім клітинка виявляється здоровою. У ході моделювання описаного процесу видавати поточний стан ділянки шкіри, що моделюється, в кожному інтервалі часу, відзначаючи заражені, несприйнятливі до інфекції й здорові клітинки. (120балів)

## **Мова програмування C#**

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

Синтаксис C# близький до С++ і Java. Мова має строгу статичну типізацію, підтримує поліморфізм, перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, винятки, коментарі у форматі XML. Перейнявши багато від своїх попередників — мов С++, Object Pascal, Модула і Smalltalk — С#, спираючись на практику їхнього використання, виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад, мова С#, на відміну від C++, не передбачає множинне успадкування класів. [2]

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

Символ # у назві мови можна інтерпретувати і як дві пари плюсів ++, що натякають на новий крок у розвитку мови порівняно з C++ (подібно до кроку від C до C++), і як музичний символ дієз, разом з буквою C, що становить в англійській мові назву ноти до-дієз. Останнє й дало назву мові. Попри те, що символ # (октоторп) насправді є символом для позначення номера на більшості клавіатур і відрізняється від символу дієз ♯ (Unicode U+266F), Microsoft, як автор мови, неодноразово зверталася до своїх клієнтів з проханням прийняти таку стилізацію. [2]

## **1.2 Етіологія стригучого лишаю**

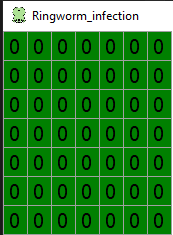
Стригучий (трихофітія, мікроспорія).

Належить до заразних, провокується грибками роду Trichophyton або Microsporum canis, ferrugineum або audoin, які часто розносяться бездомними тваринами: кішками, собаками, кроликами, мавпами та ін. Ризик зараження зростає при частому контакті з джерелом інфекції: на роботі у ветеринарних клініках і тваринницьких фермах, під час годування бродячих кішок і собак тощо.

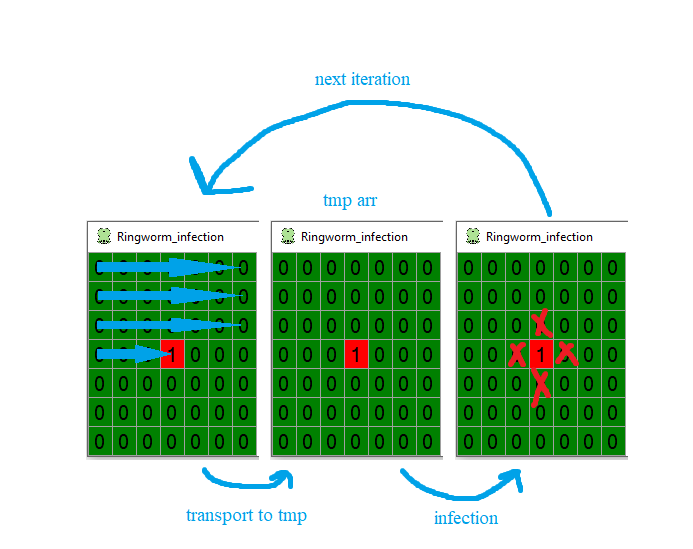
Цей лишай у людини може з'являтися на будь-якій ділянці гладкої шкіри або волосистій частині голови, рідше — на нігтях. На ураженій ділянці виникають рожеві плями з рівними обрисами, які складаються з дрібних бульбашок. На початковому етапі вони не викликають свербіж, і він іноді з'являється тільки з часом.

Життєдіяльність грибків призводить до випадання волосся в зоні плями. При локалізації на волосистій частині голови утворюються помітні лисини, на яких видно «пеньки волосся». Інфекція може поширюватися шляхом самозараження по всьому тілу. Іноді вогнища лишаю нагноюються.

## **Побудова незараженої матриці**

Для побудови матриці використовується двовимірний масив в якому, за означенням завдання, знаходяться лише нулі. Матриця будується із внесенням до неї непарного числа за принципом **(n x n)**, щоб мати змогу заразити центральну клітинуц матриці, приводячи її до стартового положення.

**1.4 Основний принцип роботи алгоритму**

Основним принципом є встановлення зараженої клітинки у центр, за допомогою натискання на кнопку **Set**, вона встановлює у центр заражену клітинку, далі для початку роботи застосунку потрібно натиснути **Start**. Після натискання відбувається зараження, яке працює таким чином: запускається цикл, який перевіряє чи є ця клітинка зараженою, якщо ні, то відбувається простий **return**, а якщо ж вона заражена програма заходить у блок перевірки **if** і заносить у додатковий масив значення координат зараженої клітики, після зчитування усіх заражених клітинок починає свою роботу нова функція, яка ще одним циклом йде вже по додатковому масиву і при знаходженні зараженої клітинки вона виконує алгоритм зараження прилеглих клітинок до зараженої у основному масиві. Одночасно з цим відбувається обробка третього масиву **immuneArr**, який зберігає у собі інформацію про імунітет клітикон, адже за постановкою задачі сказано, що клітинка є зараженою протягом 6 ітерацій циклу, після цього вона отримує імунітет на 4 ітерації і тільки після 10 ітерацій сумарно вона знову повертається до вихідного положення, тобто стає незараженою.

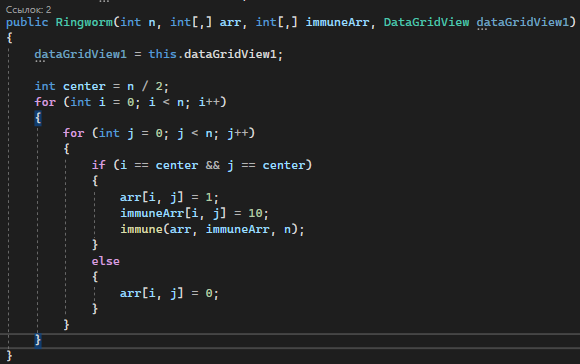
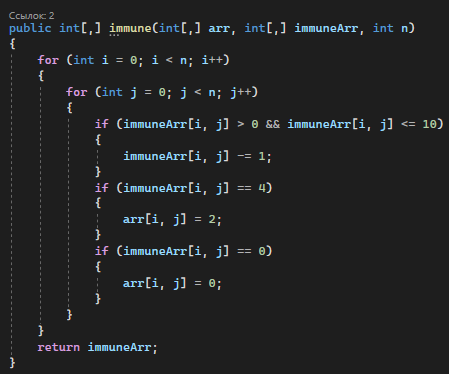
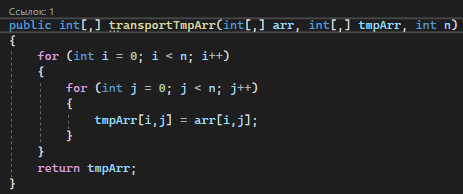
# **РОЗДІЛ 2 ОПИС ФУНКЦІОНАЛУ КЛАСІВ**

Розроблений додаток був створений у середовищі розробки **Visual Studio Comunity 2022**:Windows Forms (.NET Framework 4.8).

## **2.1 Класи основного проекту Windows Forms**

Рисунок 1

**Ringworm**

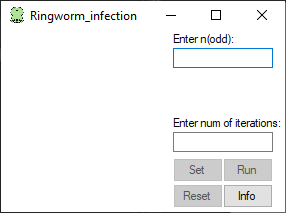
Додаток розроблено з використанням одного додаткового класу **Ringworm**, в ньому прописаний основний функціонал, створений конструктор та прописані усі допоміжні методи для коректної роботи. Перше поле, що знаходиться у класі є для того, щоб мати можливість передавати **dataGridView1** параметром у конструктор та методи, які працюють саме з **DataGridView** у проекті і встановлюють значення цих клітинок та змініють їх колір при зміні значень у масиві. Далі йде конструткор, який приймає декілька параметрів, такі як: **n**, **arr**, **immuneArr**, **dataGridView1**. В конструкторі відбуваються базові дії, такі як зараження центральної клітинки, для початку зараження всієї ділянки. Для її зараження знаходиться центр масиву і штучно змінюється значення з 0 на 1, тобто заражається. Першим методом класу є метод **immune**, який приймає параметри **n**, **arr**, **immuneArr**, які він в подальшому обробляє. Принцип дії цього методу такий, що два цикли йдуть по масивах, один по основному (**arr**), інший по масиву імунітету, одночасно роблячи перевірку стадії зараження клітинки. Цей метод відповідає за відстежуванням імунітету заражених клітинок, при зараженні клітинка одразу підпадає під блок перевірки **if**, який починає відлік, після проходження 6 одиниць часу клітинка здобуває імунітет і не піддається зараженню протягом наступних 4 одиниць часу. Повертає цей метот масив **immuneArr**. Наступний і останній метод **transportTmpArr** класу, допоміжний метод для роботи головного методу класу **Form1**. Він виконує перенос основного масиву до тимчасового масиву, для коректності роботи методу зараження.

**Form1**

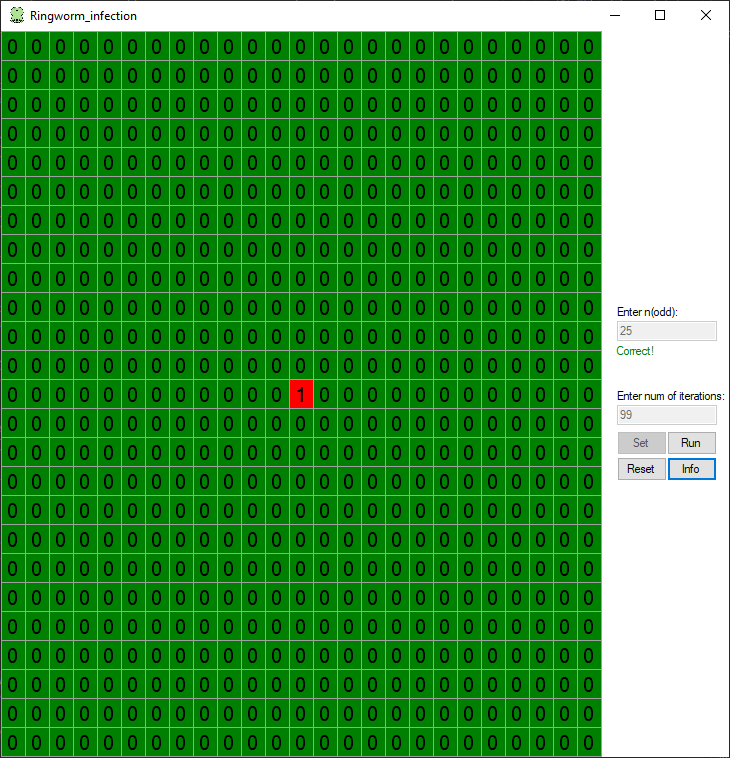
Цей клас містить у собі такі поля: **n** – для зберігання значення розміру матриці, має тип данних **int** оскільки приймає у себе лише непарні числі від 3 до 25 і не може містити дробних або нецілих чисел. **k –** виступає у ролі ітератора у методі **timer1\_Tick**, для того щоб відображати кількість пройдених ітерацій часу під час виконання алгоритму і роботи застосунку. **Iteratio –** відповідає за зберігання значення, яке задає користувач, для визначення кількості одиниць часу, для проведення досліду із зараження ділянки інфекцією. Далі створюються два основні масиви, на яких лежить уся програма і які зберігають у собі усі головні значення роботи програми, це **arr**, **immuneArr**. Наступним є абсолютно дефолтний конструктор класу **Form1**, який створюється за замовчуванням і не був ніяк змінений. Першим метод це **Form1\_Load**, в якому прописані **properties** для таких об’єктів як **label**, **button**,  **textBox** та **dataGridView**. Цей метод має функціонал забезпечення базувий функіонал для захисту від потрапляння у ексепшени, тому деякі кнопки одразу ж після запуску программи е заблокованими, щоб користувач не мав змоги доступитися до них. Це зроблено завдяку методу, властивому об’єкту **button**, .**Enabled**, який блокує кнопку і не дає користувачеві натиснути на неї. Головним методом цього додатку є метод **infection2** (2 тільки тому що це його друга модифікацція, в якій доданий набагато більший функціонал від його першої модифікації). Цей метод приймає такі параметри **n**, **arr**, **immuneArr**, в якому використовується поле рандом та змінна **rand**  для обробки функціоналу зараження. Головні функції цього методу це зараження, що працює за принципом перевірки клітинок навколо зараженої клітинки, якщо метод знаходить незаражену клітинку, біля зараженої клітинки, метод з’ясовує де саме знаходиться клітинка і який шаблон зараження до неї примінити, таких шаблонів існує декілька (основні – це **center**, **corners**, **bars**. Які потім діляться на додаткові підгрупи). Після з’ясування який саме шаблон потрібно застосувати ми потрапляємо до блоку перевірки **if**, в якому спрацьовує метод рандомного зараження, за принципом генерації рандомного числа від 0 до 1, а потім перевіркою встановлюється значення змінної **rand**, якщо 1 то спрацьовує зараження, якщо ж 0, зараження не спрацьову, така генерація рандомного числа дає нам відсоток генерації одиниці чи нуля складає 50%, тому цей пункт завдання виконано. Цей метод виконує одну ітерацію циклу зараження та повертає основний масив після відпрацювання методу. Метод **runArr** виконує поєднання програмної частини із візуальною частиною, який виводить масив **arr** у **dataGridView1**, метод використовує вкладений цикл, для того щоб вивести кожне значення масиву у відповідну позицію **dataGridView1**, також цей метод виконує функціонал покраски клітинок 0 – зелений, 1 – червоний, 2 – помаранчевий. Далі йдуть кнопки, які забезпечують користувача можливістю управління формою. У методі **setBtn** виконується встановлення дефолтних значень для розміру форми, який встановлюється в залежності від розміру **dataGridView**, також ця кнопка має превірку на те, яке число було введено для встановлення розміру матриці (можна ввести лише число від 3 до 25 включно і лише непарне число), якщо усі значення були прийняті, вони записуються у відповідні поля, а **textBox1** та **textBox2 блокуються**. У цьому методі також викликається конструктор класу **Ringworm** для задання усіх необхідних значень для початку зараження. **infoBtn\_Click** викликає **MessageBox**, який виводить повідомлення на екран із інфографікою про те що означають кольори у **dataGridView1**. **runBtn\_Click** цей метод відповідає за запуск таймеру, який заміняє цикл, в якому мали б відбуватися усі ітерації часу, встановлені користувачем. Також на кнопку прив’язані логічні блокування кнопок та функціоналу, який на данний момент не потрібен користувачеві. **resetBtn\_Click** виконує функцію скидування усіх налаштувань додатку для роботи з новими значеннями, тут скидується розмір форми до дефолтного (300px, 220px). **timer1\_Tick** один із головних методів, який відповідає за поступове проходження одиниць часу, для можливості наглядного спостереження зараження, цей метод має перевірку на кількість ітерацій, якщо кількість дорівнює нулю, то таймер зупиняється у поточному стані і в додатку відображається останнє положення масиву. Останніми методами є методи захисту від дурня **textBox1\_KeyPress**, **textBox2\_KeyPress**, **textBox2\_TextChanged**, які слугують для унеможливлювання вводу нічого окрім цифр до текст боксів, та робить превірку на те чи є **textBox2** є порожнім чи ні.

# **РОЗДІЛ 3 ДИЗАЙН КОРИСТУВАЦЬКОГО ІНТЕРФЕЙСУ**

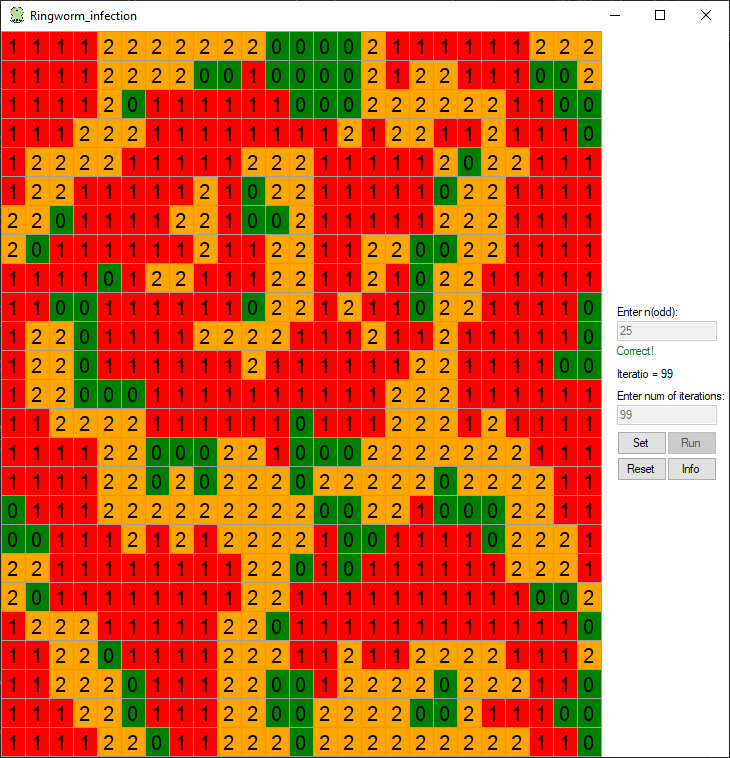
**3.1 Основна форма**



Початковий вигляд форми.



Вигляд форми після встановлення значень для головних полів.



Вигляд програми на останньому етапі її роботи.

# **РОЗДІЛ 4 ВИХІДНИЙ КОД ПРОЕКТУ**

## **4.1 Код основної форми**

using System;

using System.Drawing;

using System.Windows.Forms;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

static int n;

int k = 0;

int iteratio = 0;

int[,] arr;

int[,] immuneArr;

private Ringworm ringworm;

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

label1.Text = "";

label2.Text = "Enter n(odd):";

label3.Text = "";

label3.Visible = false;

label4.Text = "Enter num of iterations:";

setBtn.Enabled = false;

runBtn.Enabled = false;

resetBtn.Enabled = false;

textBox1.MaxLength = 2;

textBox2.MaxLength = 2;

dataGridView1.Visible = false;

}

public static int[,] transportTmpArr(int[,] arr, int[,] tmpArr, int n)

{

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

{

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

{

tmpArr[i, j] = arr[i, j];

}

}

return tmpArr;

}

public static int[,] infection2(int[,] arr, int[,] immuneArr, int n)

{

Random random = new Random();

int rand;

int[,] tmpArr = new int[n, n];

transportTmpArr(arr, tmpArr, n);

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

{

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

{

if (tmpArr[i, j] == 1)

{

//CENTER

if (i > 0 && j > 0 && i < n - 1 && j < n - 1)

{

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i - 1, j] == 0)

{

arr[i - 1, j] = 1;

immuneArr[i - 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j + 1] == 0)

{

arr[i, j + 1] = 1;

immuneArr[i, j + 1] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i + 1, j] == 0)

{

arr[i + 1, j] = 1;

immuneArr[i + 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j - 1] == 0)

{ arr[i, j - 1] = 1;

immuneArr[i, j - 1] = 10;

}

}

}

//CORNERS

if (i == 0 && j == 0)

{ //left-up

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i + 1, j] == 0)

{

arr[i + 1, j] = 1;

immuneArr[i + 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j + 1] == 0)

{

arr[i, j + 1] = 1;

immuneArr[i, j + 1] = 10;

}

}

}

if (i == n - 1 && j == 0)

{ //left-down

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i - 1, j] == 0)

{

arr[i - 1, j] = 1;

immuneArr[i - 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j + 1] == 0)

{

arr[i, j + 1] = 1;

immuneArr[i, j + 1] = 10;

}

}

}

if (i == n - 1 && j == n - 1)

{ //right-down

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j - 1] == 0)

{

arr[i, j - 1] = 1;

immuneArr[i, j - 1] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i - 1, j] == 0)

{

arr[i - 1, j] = 1;

immuneArr[i - 1, j] = 10;

}

}

}

/\*00000000000000000000000000000000000000000000000000000000000\*/

if (i == 0 && j == n - 1)

{ //right-up

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i + 1, j] == 0)

{

arr[i + 1, j] = 1;

immuneArr[i + 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j - 1] == 0)

{

arr[i, j - 1] = 1;

immuneArr[i, j - 1] = 10;

}

}

}

//BARS

if (i > 0 && i < n - 1 && j == 0)

{ // left-bar

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i - 1, j] == 0)

{ arr[i - 1, j] = 1;

immuneArr[i - 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j + 1] == 0)

{

arr[i, j + 1] = 1;

immuneArr[i, j + 1] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i + 1, j] == 0)

{

arr[i + 1, j] = 1;

immuneArr[i + 1, j] = 10;

}

}

}

if (j > 0 && j < n - 1 && i == 0)

{ // up-bar

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j - 1] == 0)

{

arr[i, j - 1] = 1;

immuneArr[i, j - 1] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i + 1, j] == 0)

{

arr[i + 1, j] = 1;

immuneArr[i + 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j + 1] == 0)

{

arr[i, j + 1] = 1;

immuneArr[i, j + 1] = 10;

}

}

}

if (i > 0 && i < n - 1 && j == n - 1)

{ // right-bar

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i - 1, j] == 0)

{

arr[i - 1, j] = 1;

immuneArr[i - 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j - 1] == 0)

{

arr[i, j - 1] = 1;

immuneArr[i, j - 1] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i + 1, j] == 0)

{

arr[i + 1, j] = 1;

immuneArr[i + 1, j] = 10;

}

}

}

if (j > 0 && j < n - 1 && i == n - 1)

{ // down-bar

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j - 1] == 0)

{

arr[i, j - 1] = 1;

immuneArr[i, j - 1] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i - 1, j] == 0)

{

arr[i - 1, j] = 1;

immuneArr[i - 1, j] = 10;

}

}

rand = random.Next(0, 2);

if (rand == 1)

{

if (arr[i, j + 1] == 0)

{

arr[i, j + 1] = 1;

immuneArr[i, j + 1] = 10;

}

}

}

}

}

}

return arr;

}

public void runArr(int[,] arr, int n)

{

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

{

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

{

if(arr[i, j] == 0)

{

dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.Green;

dataGridView1.Rows[i].Cells[j].Style.SelectionBackColor = Color.Green;

}else if (arr[i, j] == 2)

{

dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.Orange;

dataGridView1.Rows[i].Cells[j].Style.SelectionBackColor = Color.Orange;

}

else

{

dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.Red;

dataGridView1.Rows[i].Cells[j].Style.SelectionBackColor = Color.Red;

}

dataGridView1.Rows[i].Cells[j].Value = arr[i, j].ToString();

}

}

}

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

{

if (textBox1.Text == "")

{

label1.ForeColor = Color.Red;

label1.Text = "Enter something!";

}

else if (Convert.ToInt32(textBox1.Text) > 25 || Convert.ToInt32(textBox1.Text) < 3)

{

label1.ForeColor = Color.Red;

label1.Text = "Enter value in between\n 3 to 25!";

}

else

{

setBtn.Enabled = false;

runBtn.Enabled = true;

resetBtn.Enabled = true;

n = Convert.ToInt32(textBox1.Text);

if (n % 2 == 0)

{

textBox1.Text = "";

label1.ForeColor = Color.Red;

label1.Text = "Not an odd number,\n enter one more!";

}

else

{

dataGridView1.Visible = true;

n = Convert.ToInt32(textBox1.Text);

textBox1.Enabled = false;

textBox2.Enabled = false;

label1.ForeColor = Color.Green;

label1.Text = "Correct!";

switch (n)

{

case 5:

Size = new Size(300, 220);//+

MinimumSize = new Size(300, 220);

break;

case 7:

Size = new Size(312, 243);//+

MinimumSize = new Size(312, 243);

break;

case 9:

Size = new Size(360, 301);//+

MinimumSize = new Size(360, 301);

break;

case 11:

Size = new Size(408, 359);//+

MinimumSize = new Size(408, 359);

break;

case 13: Size = new Size(456, 417);//+

MinimumSize = new Size(456, 417);

break;

case 15:

Size = new Size(504, 475);//+

MinimumSize = new Size(504, 475);

break;

case 17:

Size = new Size(552, 533);//+

MinimumSize = new Size(552, 533);

break;

case 19:

Size = new Size(600, 591);//+

MinimumSize = new Size(600, 591);

break;

case 21:

Size = new Size(648, 649);//+

MinimumSize = new Size(648, 649);

break;

case 23:

Size = new Size(696, 707);//+

MinimumSize = new Size(696, 707);

break;

case 25:

Size = new Size(744, 765);//+

MinimumSize = new Size(744, 765);

break;

}

int[,] arr = new int[n, n];

int[,] immuneArr = new int[n, n];

Ringworm ringworm = new Ringworm(n, arr, immuneArr, dataGridView1);

dataGridView1.ColumnCount = n;

dataGridView1.RowCount = n;

runArr(arr, n);

}

}

}

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

{

MessageBox.Show("0 - neutral \n1 - infected \n2 - protected", "Info", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

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

{

dataGridView1.Visible = true;

label3.Visible = true;

setBtn.Enabled = false;

infoBtn.Enabled = false;

runBtn.Enabled = false;

resetBtn.Enabled = false;

timer1.Enabled = true;

arr = new int[n, n];

immuneArr = new int[n, n];

ringworm = new Ringworm(n, arr, immuneArr, dataGridView1);

dataGridView1.ColumnCount = n;

dataGridView1.RowCount = n;

iteratio = Convert.ToInt32(textBox2.Text);

}

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

{

textBox1.Enabled = false;

textBox1.Text = "";

textBox2.Enabled = false;

textBox2.Text = "";

setBtn.Enabled = true;

setBtn.Enabled = false;

runBtn.Enabled = false;

resetBtn.Enabled = false;

label1.Text = "";

label3.Visible = false;

textBox1.Enabled = true;

textBox2.Enabled = true;

dataGridView1.Visible = false;

MinimumSize = new Size(300, 220);

Size = new Size(300, 220);

}

private void timer1\_Tick(object sender, EventArgs e)

{

k++;

if (iteratio == 0)

{

k = 0;

timer1.Stop();

setBtn.Enabled = true;

infoBtn.Enabled = true;

resetBtn.Enabled = true;

}

else

{

runArr(infection2(arr, immuneArr, n), n);

ringworm.immune(arr, immuneArr, n);

label3.Visible = true;

label3.Text = "Iteratio = " + (k);

iteratio -= 1;

}

}

private void textBox1\_KeyPress(object sender, KeyPressEventArgs e)

{

if (char.IsLetter(e.KeyChar))

{

e.Handled = true;

}

if (!(char.IsLetter(e.KeyChar)) && !(char.IsNumber(e.KeyChar)) && !(char.IsControl(e.KeyChar)) && !(char.IsWhiteSpace(e.KeyChar)))

{

e.Handled = true;

}

}

private void textBox2\_KeyPress(object sender, KeyPressEventArgs e)

{

if (char.IsLetter(e.KeyChar))

{

e.Handled = true;

}

if (!(char.IsLetter(e.KeyChar)) && !(char.IsNumber(e.KeyChar)) && !(char.IsControl(e.KeyChar)) && !(char.IsWhiteSpace(e.KeyChar)))

{

e.Handled = true;

}

}

private void textBox2\_TextChanged(object sender, EventArgs e)

{

if (textBox2.Text == "")

{

setBtn.Enabled = false;

}

else

{

setBtn.Enabled = true;

}

}

}

}

## **4.2 Код класу Ringworm**

using System.Windows.Forms;

namespace WindowsFormsApp1

{

public class Ringworm

{

DataGridView dataGridView1 = null;

public Ringworm(int n, int[,] arr, int[,] immuneArr, DataGridView dataGridView1)

{

dataGridView1 = this.dataGridView1;

int center = n / 2;

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

{

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

{

if (i == center && j == center)

{

arr[i, j] = 1;

immuneArr[i, j] = 10;

immune(arr, immuneArr, n);

}

else

{

arr[i, j] = 0;

}

}

}

}

public int[,] immune(int[,] arr, int[,] immuneArr, int n)

{

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

{

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

{

if (immuneArr[i, j] > 0 && immuneArr[i, j] <= 10)

{

immuneArr[i, j] -= 1;

}

if (immuneArr[i, j] == 4)

{

arr[i, j] = 2;

}

if (immuneArr[i, j] == 0)

{

arr[i, j] = 0;

}

}

}

return immuneArr;

}

public int[,] transportTmpArr(int[,] arr, int[,] tmpArr, int n)

{

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

{

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

{

tmpArr[i,j] = arr[i,j];

}

}

return tmpArr;

}

}

}

# **Висновки**

Отже, метою даної курсової роботи була розробка десктоп-додатка мовою програмування C# об’єктно-орієнтованим підходом програмування, використовуючи різні елементи графічного інтерфейсу та власну динамічну бібліотеку.

Загалом, суть об’єктно-орієнтованого програмування – це сприйняття предметної області у вигляді об’єктів, які в свою чергу мають власний стан та поведінку. Дана парадигма програмування дозволяє розробникам простіше створювати програми, використовуючи такий підхід, адже людині дуже легко мислити в категоріях об’єктів. На відміну від інших парадигм програмування, які вимагають перебудови мислення для представлення задачі у вигляді функцій, об’єктно-орієнтоване програмування відмінно сприймається звичним мисленням людини.

Також варто зазначити, що об’єктно-орієнтоване програмування будується на чотирьох основних принципах: інкапсуляція (приховування інформації), успадкування (наслідування), абстракція (спрощення складності шляхом моделювання класів, що відповідають конкретній проблемі) та поліморфізм (два або більше класів можуть реагувати по-різному на однакове повідомлення).

Відповідно до поставленої задачі, була виконана курсова робота згідно усіх вимог завдання та усіх поставленних задач. Після виконання роботи я засвоїв усі використовувані інструменти для створення форм у середовищі розробки Visual Studio 2022. На базі отриманих знань, було розроблено дизайн форм програми для інтуїтивної роботи користувача з інтерфейсом. Також було написано програмний код двох проектів: основного графічного інтерфейсу програми.

Під час виконання курсової роботи було використано офіційну документацію до мови програмування C# та платформи Windows Forms від Microsoft для ознайомлення з використанням різних елементів управління – кнопок, таблиць тощо.

# **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/cjzU>
3. Мова програмування C# [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/C_Sharp>
4. Допомога у виконанні роботи: <https://metanit.com/sharp/>
5. Допомога у виконанні роботи: <https://stackoverflow.com/questions/tagged/c%23>
6. Допомога у виконанні роботи: <https://learn.microsoft.com/en-us/dotnet/core/diagnostics/debug-stackoverflow>

Git Hub: