**Зміст**

[Вступ 3](#_Toc8672727)

[Розділ 1. Огляд методів та засобів реалізації поставленого завдання 4](#_Toc8672728)

[1.1. Огляд та вибір інструментарю для розробки програми 4](#_Toc8672729)

[1.2. Вибір мови програмування 4](#_Toc8672730)

[1.2.1. Мова програмування C# 4](#_Toc8672731)

[1.2.2. Мова програмування Java 5](#_Toc8672732)

[1.3. Графічний інтерфейс 7](#_Toc8672733)

[1.4. Алгоритм «Решето Ератосфена» 7](#_Toc8672734)

[1.5. Висновок до першого розділу 8](#_Toc8672735)

[Розділ 2. Проектування програмного продукту 9](#_Toc8672736)

[2.1. Робота алгоритму Ератосфена 9](#_Toc8672737)

[2.2. Опис функціоналу інтерфейсу програмного продукту 10](#_Toc8672738)

[2.3. Метод замальовування клітинок з кратними числами 11](#_Toc8672739)

[2.4. Висновок до другого розділу 12](#_Toc8672740)

[Розділ 3. Реалізація програмного продукту 13](#_Toc8672741)

[3.1. Інтерфейс програми 13](#_Toc8672742)

[3.2. Заповнення DataGridView 14](#_Toc8672743)

[3.3. Елементи управління додатком 15](#_Toc8672744)

[3.4. Елемент управління Timer 19](#_Toc8672745)

[3.5. Методи замальовування клітинок, та графічне зображення алгоритму 20](#_Toc8672746)

[3.6. Висновок до третього розділу 26](#_Toc8672747)

[Висновок 27](#_Toc8672748)

[Список використаних джерел 28](#_Toc8672749)

[Додаток А 29](#_Toc8672750)

[Додаток Б 31](#_Toc8672751)

[Додаток В 33](#_Toc8672752)

Вступ

**Актуальність:** зараз прості числа використовуються в різних областях: шифруванні, нанотехнологіях, програмуванні і в багатьох інших технічних науках, а саме технічні науки є провідними науками сьогодення, тому хто збирається, або вже вивчає їх, потрібно знати даний алгоритм пошуку простих чисел.

**Мета курсової роботи:** дослідити множину простих чисел алгоритмом “Решето Ератосфена”, визначити всі прості числа, які менші за N.

Задачі:

1. Розгляд інформаційних джерел;
2. Огляд алгоритмів роботи програм;
3. Розробити структурну і функціональну схему;
4. Реалізувати створений програмний продукт;
5. Провести тестування та виправлення помилок у разі їх виявлення;

Розроблений продукт являє собою програмний додаток який, отримує на вхід число N (число по яке шукати прості числа), і виводить усі прості числа в діапазоні 2 < N.

**Просте число** – це натуральне число, яке має рівно два різних натуральних [дільники](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%96%D0%BB%D1%8C%D0%BD%D1%96%D1%81%D1%82%D1%8C) (лише [1](https://uk.wikipedia.org/wiki/1_(%D1%87%D0%B8%D1%81%D0%BB%D0%BE)) і саме число). Решту чисел, окрім одиниці, називають [складеними](https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D0%BB%D0%B0%D0%B4%D0%B5%D0%BD%D0%B5_%D1%87%D0%B8%D1%81%D0%BB%D0%BE). Таким чином, всі натуральні числа, більші від одиниці, розбивають на прості і складені. [Теорія чисел](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D1%96%D1%8F_%D1%87%D0%B8%D1%81%D0%B5%D0%BB) вивчає властивості простих чисел. В [теорії кілець](https://uk.wikipedia.org/wiki/%D0%9A%D1%96%D0%BB%D1%8C%D1%86%D0%B5_(%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0)) простим числам відповідають [незвідні елементи](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B7%D0%B2%D1%96%D0%B4%D0%BD%D0%B8%D0%B9_%D0%B5%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82)

**Решето Ератосфена –** алгоритм знаходження всіх простих чисел до деякого цілого числа n, який приписують давньогрецькому математику Ератосфену Киренського. Як і в багатьох випадках, тут назва алгоритму говорить про принцип його роботи, тобто решето на увазі фільтрацію, в даному випадку фільтрацію всіх чисел за винятком простих. У міру проходження списку потрібні числа залишаються, а непотрібні (вони називаються складовими) виключаються.

# **Розділ 1.** **Огляд методів та засобів реалізації поставленого завдання**

## **Огляд та вибір інструментарю для розробки програми**

Спираючись на поставлену мету та алгоритм роботи програми необхідно обрати найоптимальніший інструментарій для його реалізації. Вибір постав перед двома мовами програмування C# та Java. Майбутня мова програмування повинна мати великий набір різноманітних бібліотек для роботи з графікою та зручної реалізації інтерфейсу.

## **Вибір мови програмування**

* + 1. **Мова програмування C#**

C# (вимовляється Сі-шарп) — [об'єктно-орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%E2%80%99%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [мова програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) з безпечною [системою типізації](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%82%D0%B8%D0%BF%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%97) для платформи [.NET](https://uk.wikipedia.org/wiki/.NET). Розроблена [Андерсом Гейлсбергом](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B4%D0%B5%D1%80%D1%81_%D0%93%D0%B5%D0%B9%D0%BB%D1%81%D0%B1%D0%B5%D1%80%D0%B3), [Скотом Вілтамутом](https://uk.wikipedia.org/w/index.php?title=%D0%92%D1%96%D0%BB%D1%82%D0%B0%D0%BC%D1%83%D1%82_%D0%A1%D0%BA%D0%BE%D1%82&action=edit&redlink=1) та [Пітером Гольде](https://uk.wikipedia.org/w/index.php?title=%D0%93%D0%BE%D0%BB%D1%8C%D0%B4%D0%B5_%D0%9F%D1%96%D1%82%D0%B5%D1%80&action=edit&redlink=1) під егідою [Microsoft Research](https://uk.wikipedia.org/wiki/Microsoft_Research) (при фірмі [Microsoft](https://uk.wikipedia.org/wiki/Microsoft)).

[Синтаксис](https://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81) C# близький до [С++](https://uk.wikipedia.org/wiki/%D0%A1%2B%2B) і [Java](https://uk.wikipedia.org/wiki/Java). [Мова](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0) має строгу статичну типізацію, підтримує [поліморфізм](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%96%D0%BC%D0%BE%D1%80%D1%84%D1%96%D0%B7%D0%BC_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), перевантаження операторів, вказівники на функції-члени класів, атрибути, події, властивості, [винятки](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%B2%D0%B8%D0%BD%D1%8F%D1%82%D0%BA%D1%96%D0%B2), коментарі у форматі [XML](https://uk.wikipedia.org/wiki/XML). Перейнявши багато що від своїх попередників — мов [С++](https://uk.wikipedia.org/wiki/%D0%A1%2B%2B), [Delphi](https://uk.wikipedia.org/wiki/Delphi_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), [Модула](https://uk.wikipedia.org/w/index.php?title=%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0&action=edit&redlink=1)і [Smalltalk](https://uk.wikipedia.org/wiki/Smalltalk) — С#, спираючись на практику їхнього [використання](https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D0%BD%D1%8F), виключає деякі моделі, що зарекомендували себе як проблематичні при розробці програмних систем, наприклад [множинне спадкування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B6%D0%B8%D0%BD%D0%BD%D0%B5_%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) класів (на відміну від C++)

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](https://uk.wikipedia.org/wiki/%D0%94%D1%96%D1%94%D0%B7), разом з буквою C, що становить в англійській мові назву ноти [до-дієз](https://uk.wikipedia.org/w/index.php?title=%D0%94%D0%BE-%D0%B4%D1%96%D1%94%D0%B7&action=edit&redlink=1). Останнє й дало назву мові. Попри те, що [символ #](https://uk.wikipedia.org/wiki/%D0%9E%D0%BA%D1%82%D0%BE%D1%82%D0%BE%D1%80%D0%BF) (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.

C# розроблялась як мова програмування прикладного рівня для [CLR](https://uk.wikipedia.org/wiki/CLR) і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#. Присутність або відсутність тих або інших виразних особливостей мови диктується тим, чи може конкретна мовна особливість бути трансльована у відповідні конструкції CLR. Так, з розвитком CLR від версії 1.1 до 2.0 значно збагатився і сам C#; подібної взаємодії слід чекати і надалі. (Проте ця закономірність буде порушена з виходом C# 3.0, що є розширеннями мови, що не спираються на розширення платформи .NET.) CLR надає C#, як і всім іншим .NET-орієнтованим мовам, багато можливостей, яких позбавлені «класичні» мови програмування. Наприклад, збірка сміття не реалізована в самому C#, а проводиться CLR для програм, написаних на C# точно так, як і це робиться для програм на VB.NET, J# тощо [4].

* + 1. **Мова програмування Java**

**Java** (вимовляється Джава) — [об'єктно-орієнтована](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) [мова програмування](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), випущена 1995 року компанією [«Sun Microsystems»](https://uk.wikipedia.org/wiki/Sun_Microsystems) як основний компонент платформи Java. З 2009 року мовою займається компанія [«Oracle»](https://uk.wikipedia.org/wiki/Oracle_Corporation), яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми [компілюються](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%82%D0%BE%D1%80) у [байт-код](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B9%D1%82-%D0%BA%D0%BE%D0%B4), який при виконанні інтерпретується [віртуальною машиною](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java) для конкретної платформи.

«Oracle» надає компілятор Java та [віртуальну машину Java](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D1%80%D1%82%D1%83%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0_%D0%BC%D0%B0%D1%88%D0%B8%D0%BD%D0%B0_Java), які задовольняють специфікації [Java Community Process](https://uk.wikipedia.org/wiki/Java_Community_Process), під ліцензією [GNU General Public License](https://uk.wikipedia.org/wiki/GNU_General_Public_License).

Мова значно запозичила синтаксис із [C](https://uk.wikipedia.org/wiki/C_(%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і [C++](https://uk.wikipedia.org/wiki/C%2B%2B). Зокрема, взято за основу об'єктну модель С++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розробки об'єктно-орієнтованих програм. Ряд дій, які в С/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

На противагу [C++](https://uk.wikipedia.org/wiki/C%2B%2B), Java є більш об'єктно-орієнтованою. Всі дані і дії групуються в класи [об'єктів](https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)). Виключенням з повної об'єктності (як скажімо в [Smalltalk](https://uk.wikipedia.org/wiki/Smalltalk)) є примітивні типи (int, float тощо). Це було свідомим рішенням проектувальників мови задля збільшення швидкості. Через це Java не вважається повністю об'єктно-орієнтовною мовою.

У Java всі об'єкти є похідними від головного об'єкта (він називається просто Object), з якого вони успадковують базову поведінку і властивості.

Хоча у C++ вперше стало доступне множинне [успадкування](https://uk.wikipedia.org/wiki/%D0%A3%D1%81%D0%BF%D0%B0%D0%B4%D0%BA%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), але у Java можливе тільки одинарне успадкування, завдяки чому виключається можливість конфліктів між членами класу ([методи](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)) і [змінні](https://uk.wikipedia.org/wiki/%D0%97%D0%BC%D1%96%D0%BD%D0%BD%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))), які успадковуються від базових класів [3].

Співставивши всі плюси та мінуси цих мов, вибір упав на мову програмування C#, через великий набір різноманітних бібліотек для роботи з графікою та зручної реалізації інтерфейсу. Для графічної реалізація вибрано інтерфейс програмування додатків Windows Forms, який є простим в освоєнні, та з великим функціоналом.

## **Графічний інтерфейс**

Windows Forms - інтерфейс програмування додатків (API), що відповідає за графічний інтерфейс користувача і є частиною Microsoft .NET Framework. Даний інтерфейс спрощує доступ до елементів інтерфейсу Microsoft Windows за рахунок створення обгортки для існуючого Win32 API в керованому коді. Причому керований код - класи, що реалізують API для Windows Forms, що не залежать від мови розробки. Тобто програміст однаково може використовувати Windows Forms як при написанні ПЗ на C #, С ++, так і на VB.Net, J # і ін [5].

* 1. **Алгоритм «Решето Ератосфена»**

1. Вводимо число **N**.
2. Перше просте число - два. Викреслимо всі числа більші двох, які діляться на два (4, 6, 8 …)
3. Наступне число, яке залишилося незакресленим (три), є простим. Викреслюємо всі числа більші трьох та кратні трьом (6, 9 …).
4. Наступне незакреслене число (п'ять) є простим. Викреслимо всі числа більші п'яти та кратні п'яти (10, 15, 20, 25 …).
5. Повторюємо операцію поки не буде досягнуто число N:
6. Наступне не закреслене число є простим. Виписуємо прості числа. Викреслимо всі числа більші нього та кратні йому [1].

Числа, які залишилися незакресленими після цієї процедури - прості.

Невелика оптимізація алгоритму:

Користуючись решетом Ератосфена викреслювання можна припинити, як тільки ми дійдемо до простого числа, яке більше ніж √N.

До цього моменту всі не викреслені числа будуть простими.

У нашому випадку при N = 155, після того, як ми викреслили числа кратні 11, подальше викреслювання можна не проводити.

Ще трохи операцій можна заощадити, якщо починати викреслювати кратні k (де k просте число), починаючи не з 2k, а з номера k в квадраті (k2) [2].

## **1.5. Висновок до першого розділу**

В даному розділі було розглянуто алгоритм, та вибрано мову програмування з графічним інтерфейсом для реалізації програмного продукту.

# **Розділ 2. Проектування програмного продукту**

* 1. **Робота алгоритму Ератосфена**

Нижче зображено схему роботи головного алгоритму пошуку простих чисел (рис. 2.1).

Початок

Замальовуємо всі числа, які ще не замальовані і кратні простому.

Перебираємо числа від 2 до (де N число по яке шукати прості числа).

Перевіряємо чи є число простим

Кінець

+

+

-

-

Рисунок 2.1 – Блок - схема основного алгоритму

1. Перший цикл перебирає числа від двох до (кореня з N).
2. Якщо число просте то переходимо до наступного пункту, інакше повертаємось до пункту 1.
3. Замальовуємо всі числа, які ще не замальовані і кратні простому числу.
4. В кінці роботи алгоритму всі числа, які виявились не замальованими є простими.
   1. **Опис функціоналу інтерфейсу програмного продукту**

Запустивши програму перед користувачем з’являється комп’ютерний додаток з графічним інтерфейсом .

В додатку буде:

* чотири кнопки «Auto», «Steps», «Clear», «Info»;
* основна частина, таблиця, яка і буде слугуввати для показу алгоритму;
* Поле на яке будуть виводитись прості числа;
* Для покрокового показу алгоритму поле в якому буде показуватись номер числа на якому ми зараз знаходимось;

Після натискання кнопки «Info», повинно з’явитися випадаюче меню в якому буде два поля, “Інформація про функціонал додатку”, та “Інформація про алгоритм”.

Кнопка «Auto» запускатиме автоматичний показ алгоритму, після її натискання кнопки «Steps» і «Clear» стають не активними, на місці кнопки «Auto» з'являється кнопка «Stop», яка призупиняє показ алгоритму, після її натискання стає активною кнопка «Clear», показ можна відновити натиснувши кнопку «Continue», яка з’явиться після натискання на кнопку «Stop».

Кнопка «Steps» дозволятиме зообразити покроковий показ алгоритму, після її натискання кнопка «Auto» стає не активною, при натисканні кнопки «Steps» автоматично переходимо до наступного числа, в порядку зростання від 2 до 11.

Кнопка «Clear» дозволяє очистити таблицю, після її натисканння всі інші кнопки стають активними і ми можемо знову показувати роботу алгоритму. Під час автоматичного показу кнопка стає активною лише тоді, коли натиснута кнопка «Stop», а при покроковому показі вона завжди активна.

* 1. **Метод замальовування клітинок з кратними числами**

Методом який замальовує клітинки - підпрограма, яка замальовує клітинки таблиці з кратними числами, блок-схема (див. додаток А).

Спершу потрібно заповнити таблицю відповідним чином (рис. 2.2)

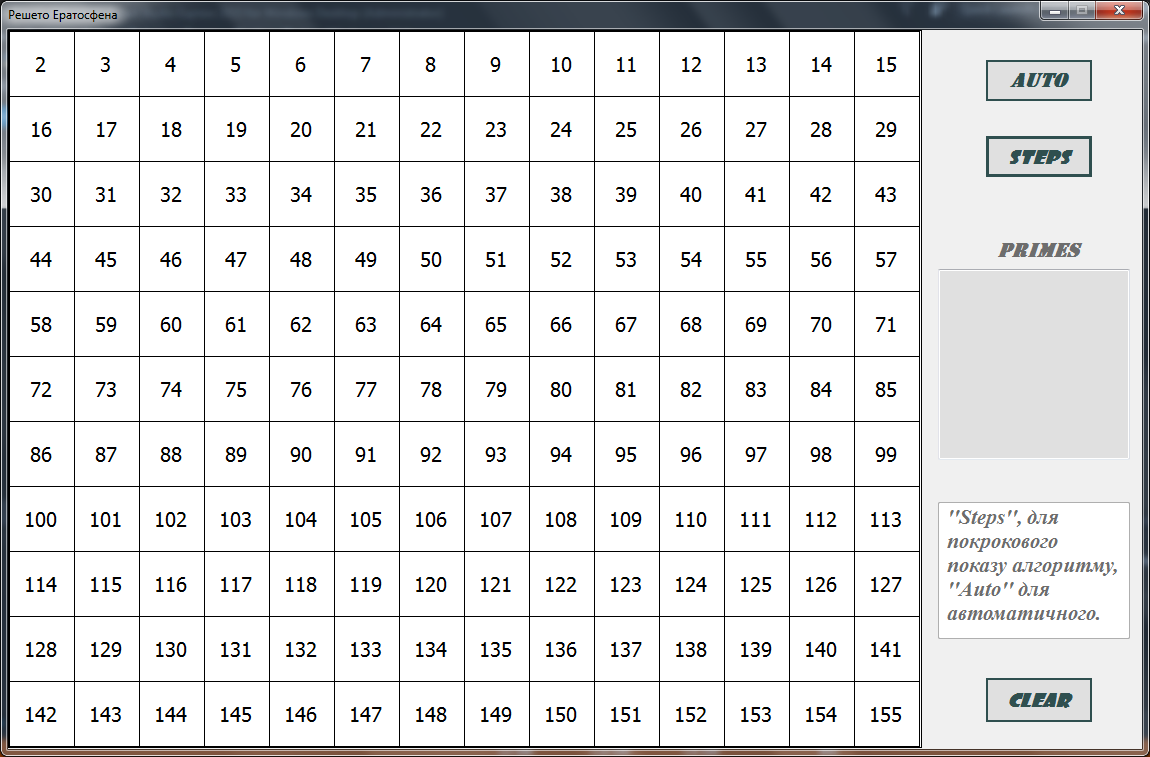


Рисунок 2.2 – Вигляд заповненої таблиці.

Для того, щоб програма сама знаходила кратні числа і замальовувала ці клітинки відповідним кольором, нам потрібні вкладені цикли де:

* перший цикл перебирає номера рядків.
* другий цикл перебирає номера стовпчиків.

Якщо умова першої перевірки виконується то клітинка замальовується відповідним кольором, а саме:

* числа кратні 2, жовто – зеленим;
* числа кратні 3, синім;
* числа кратні 5, жовтим;
* числа кратні 7, темно – зеленим;
* числа кратні 11, фіолетовим;

Для того, щоб ті клітинки, які уже замальовані не замальовувались ще раз, створимо глобальний двовимірний масив, який буде зберігати число 1 (один), якщо клітинка замальована і 0 (нуль), якщо ні.

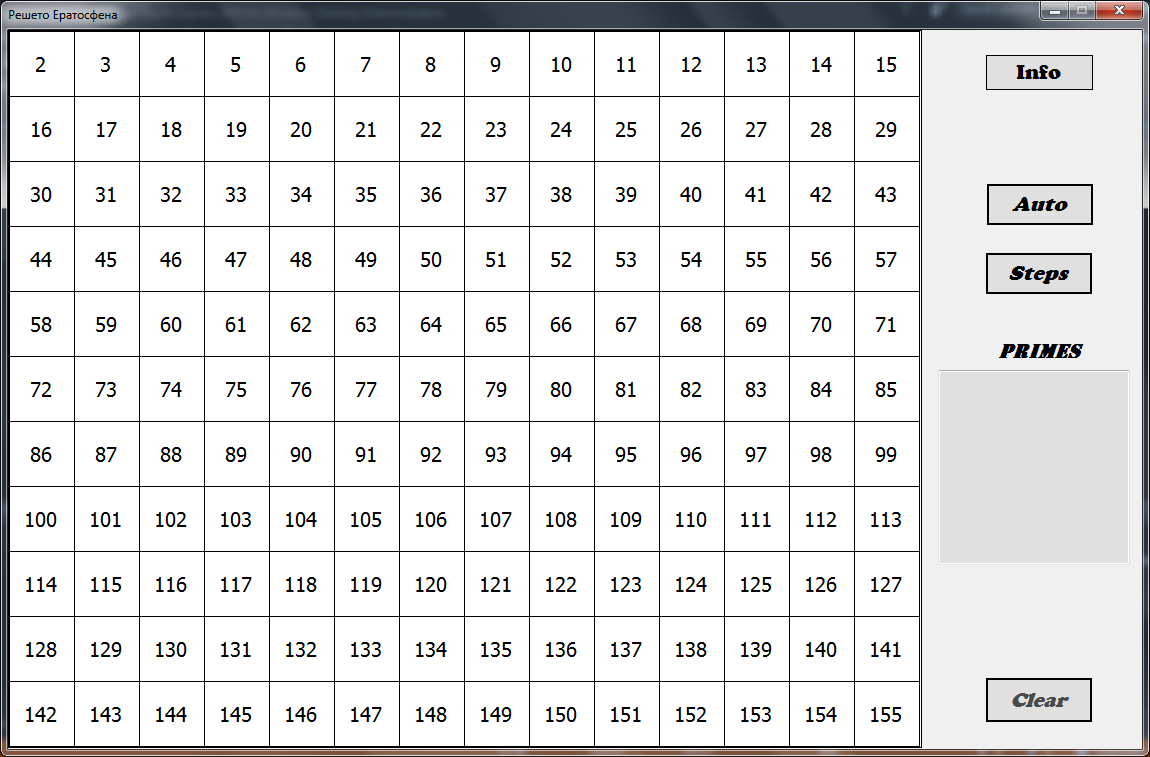
## **2.4. Висновок до другого розділу**

На етапі проектування програмного продукту було визначено схему й опис роботи додатку, було створено, описано та відображено використовуючи блок-схеми, роботу головного та побічних алгоритмів, які допоможуть створити добре функціонуючу програму.

**Розділ 3. Реалізація програмного продукту**

**Інтерфейс програми**

За допомогою Windows Forms створюємо інтерфейс (рис. 3.1).



Рисинук 3.1 – Створений інтерфейс за допогою Windows Forms

Лістинг 3.1 Ініціалізація елементів форми перед початком роботи програми.

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

{

ClearButton.Enabled = false;

dataGridView1.Enabled = false;

dataGridView1.Select();

// блокує форму від змін

this.FormBorderStyle = FormBorderStyle.FixedDialog;

dataGridView1.RowCount = 11;

dataGridView1.ColumnCount = 14;

// задає шрифт тексу в dataGridView1

this.dataGridView1.DefaultCellStyle.Font = new Font("Tahoma", 15);

// робить ширину клітинок рівну 65

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

{

dataGridView1.Columns[i].Width = 65;

}

arrayOfNumbersPaintedCells = new int[dataGridView1.RowCount, dataGridView1.ColumnCount];

PrintArrayWihtNumbers();

}

## **3.2. Заповнення DataGridView**

Перед заповненням самої таблиці її потрібно помістити на форму, в панелі управління (ToolBox), знаходимо елемент **DataGridView**, і зажавши ліву кнопку миші перетягуємо його на форму, після цього потрібно налаштувати саму таблицю по необхідним параметрам [6]:

* кількість рядків 11, кількість стовпчиків 14;
* ширина і висота клітинки рівна 65;
* робимо так, щоб в додатку ми не могли внести ніякі зміни до таблиці;
* задаємо шрифт елементів таблиці "Tahoma", кегель 15;

Всі ці дії можна зробити за допомогою властивостей (Properties), або самого коду програми.

Сам метод заповнення **DataGridView** елементами (див. Додаток Б).

Змінна z слугує наступним числом в таблиці, тобто коли j (номер стовпчика) стає більшим, за кількість стовпчиків, то ми переходимо до наступного рядка, j (номер стовпчика) знову стає нуль, і число z продовжує збільшуватись поки не буде досягнуто останнього рядка і останнього стовпчика.

В результаті це буде мати наступний вигляд (рис 3.2).

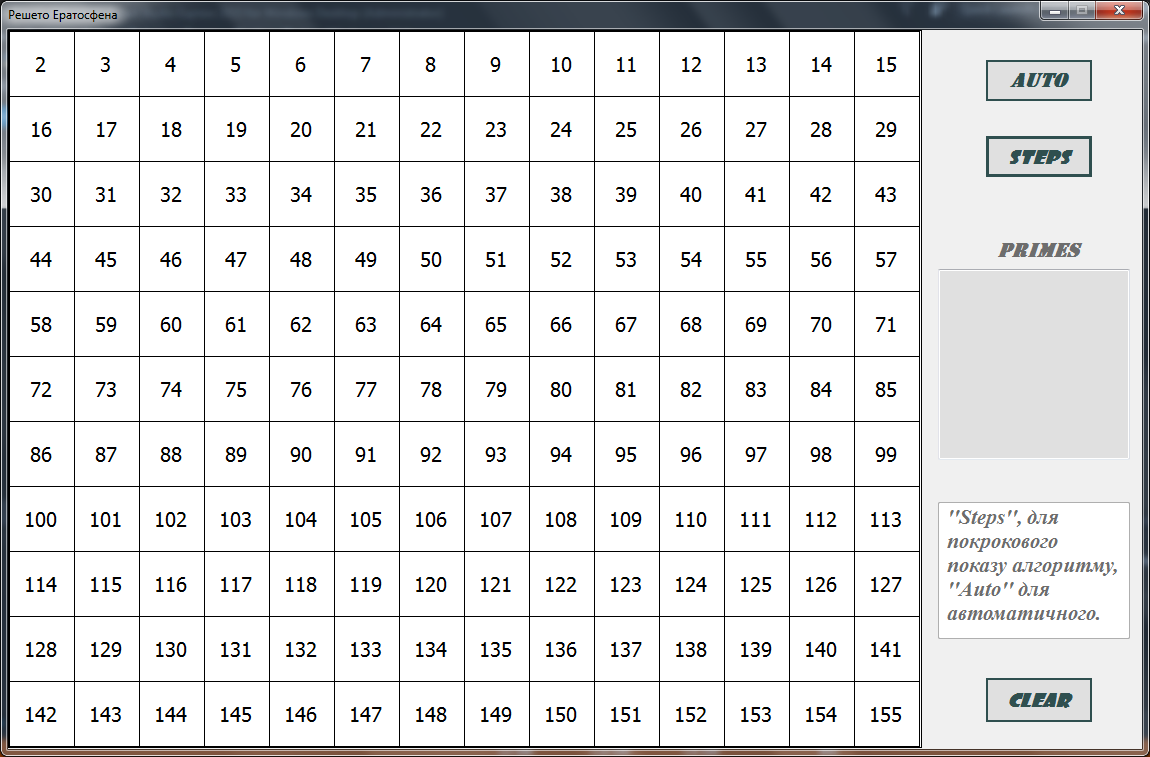


Рисунок 3.2 – Результат заповненої таблиці.

**3.3. Елементи управління додатком**

Основними елементами управління програми є кнопки.

Всі кнопки знаходяться в окремому файлі Button.cs

Кнопка «Info» − кнопка після натискання на яку з’являється випадаюче меню (рис. 3.3), його можна також приховати після повторного натиску.

Лістинг 3.2 Кнопка виводить випадаюче меню

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

{

// якщо кількість натисків парне число показуємо випадаюче меню

if (clickCount % 2 == 0)

{

contextMenuStrip1.Show(ButtonInfo, new Point(0, ButtonInfo.Height));

}

else if (clickCount % 2 != 0) // якщо не парне приховуємо

{

contextMenuStrip1.Close();

}

clickCount++; // збільшуємо кількість натисків на кнопку ButtonInfo

}

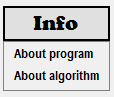


Рисунок. 3.3 − Випадаюче меню

Змінна clickCount задається, як глобальна для класу.

Якщо значення змінної clickCount парне число, після натиску на кнопку «Info» з’являється випадаюче меню, якщо значення змінної clickCount непарне число меню приховується.

У випадаючому меню є два поля:

* About program
* About algorithm

Після натиску на поле «About program» з’являється нове вікно, в якому знаходиться інформація про функціонал додатку, ця інформація зчитується з текстового файлу about\_program.txt, і виводиться в MessageBox, змінна clickCount збільшується на 1 (один).

Лістинг 3.3 Елемент випадаючого меню «Аbout program»

private void toolStripMenuItem1\_Click(object sender, EventArgs e)

{

// якщо натиснутий елемент з випадаючого меню "about program", зчитуємо дані з файла "about\_program.txt" і виводимо їх на MessageBox

string lines = File.ReadAllText("about\_program.txt", Encoding.UTF8);

MessageBox.Show(lines);

// збільшуємо кількість натисків

clickCount++;

}

Після натиску на поле «About algorithm» з’являється нове вікно, в якому знаходиться інформація про алгоритм, ця інформація зчитується з текстового файлу about\_algorithm.txt, і виводиться в MessageBox, змінна clickCount збільшується на 1 (один).

Лістинг 3.4 Елемент випадаючого меню «Аbout algorithm»

// елемент випадаючого меню about algorithm

private void toolStripMenuItem2\_Click(object sender, EventArgs e)

{

// якщо натиснутий елемент з випадаючого меню "about algorithm", зчитуємо дані з файла "about\_algorithm.txt" і виводимо їх на MessageBox

string lines = File.ReadAllText("about\_algorithm.txt", Encoding.UTF8);

MessageBox.Show(lines);

clickCount++; // збільшуємо кількість натисків

}

Кнопка «Auto» − запускає таймер.

Лістинг 3.5 Кнопка для запуску автоматичного показу алгоритму

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

{

// вмикається таймер, і всі інші кнопки стають не активними, крім кнопки Stop і Continue

timer1.Start();

StepsButton.Enabled = false;

AutoButton.Enabled = false;

ClearButton.Enabled = false;

// переносимо кнопку AutoButton\_Click назад

AutoButton.SendToBack();

}

Після натиску на кнопку «Auto» з’являється кнопка «Stop», яка зупиняє роботу таймера.

Лістинг 3.6 Кнопка, яка зупиняє роботу таймера

// кнопка для зупинки таймера

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

{

// зупиняє таймер, кнопка для очистки таблиці стає активною

timer1.Stop();

ClearButton.Enabled = true;

// переносимо кнопку StopButton\_Click назад

StopButton.SendToBack();

}

Після натиску на кнопку «Stop» з’являється кнопка «Continue», яка включає таймер.

Лістинг 3.7 Кнопка, яка продовжує роботу таймера

// кнопка для продоження роботи таймера

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

{

// запускає (продовжує) роботу таймера, кнопка для очистки таблиці стає не активною

timer1.Start();

ClearButton.Enabled = false;

// переносимо кнопку StopButton\_Click вперед

StopButton.BringToFront();

}

Кнопка «Steps» − дозволяє покроково показати основну роботу алгоритму (див. Додаток Б).

Коли змінна nextNumber стає рівною (число по яке шукати прості числа), тобто 12, то кнопка «Steps», стає не активною і запускається таймер.

Кнопка «Clear» − викликає метод, який очищає (замальовує білим кольором) всі клітинки таблиці і приводить елементи масиву статусів до 0 (нуля), ще після натискання ця кнопка повертає всі значення змінних до початкових (див. Додаток Б).

## **3.4. Елемент управління Timer**

Елемент управління таймер є одним з головних елементів в програмі, його можна добавити на форму наступним чином: перейти в панель інструментів (ToolBox), знайти елемент «Timer», і зажавши ліву клавішу миші перетянути його на форму (рис. 3.4).

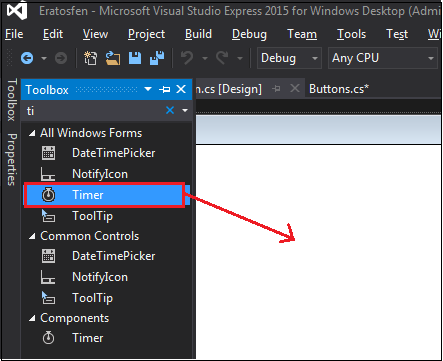
****

Рисунок 3.4 − Вставка таймера на форму

Далі налаштовуємо таймер під нашу програму (див. Додаток Б).

**3.5. Методи замальовування клітинок, та графічне зображення алгоритму**

Умова при якій будуть замальовуватись клітинки.

Лістинг 3.8 Умова при якій замальовуються числа кратні простому

if ((int)dataGridView1.Rows[i].Cells[j].Value % nextNumber == 0 && (int)dataGridView1.Rows[i].Cells[j].Value >= nextNumber \* nextNumber && arrayOfNumbersPaintedCells[i, j] == 0)

Якщо елемент таблиці ділиться націло на число, яке обробляється в певний момент часу, і цей елемент більший-рівний ніж це число в квадраті, та статус клітинки рівний нулю, ми можемо замальовувати таку клітинку і переходимо до наступного етапу, а саме вибору кольору.

Вибір кольору зроблено за допомогою оператора вибору, який в мові програмування C# називається switch.

Оператор switch використовується для того, щоб замальовувати клітинки таблиці з кратними числами відповідними кольорами, для кожного простого числа від 2 до кореня з N.

В нашому випадку в таблиці число N рівне 155, відповідно корінь з цього числа приблизно рівний 12, знаючи прості числа від 2 до 12, можна кожній клітинці з кратними числами присвоїти свій колір, тоді такі клітинки будуть замальовуватись відповідним кольором.

Елемент масива arrayOfNumbersPaintedCells, стає рівним одиниці, це означfє, що клітинка уже замальована, і вона більше уже не буде замальовуватись іншим кольором.

Весь вище зазначений код (в пункті 3.5.) знаходиться у вкладених циклах, це потрібно так як ми користуємось таблицею, і нам потрібно мати доступ до кожного елемента (див. Додаток В).

Цей метод замальовує лише клітинки з кратними числами, нам ще потрібний один метод для замальовування клітинок в яких прості числа, і виводитиме їх на текстбокс (див. Додаток В).

Змінні indexI та indexJ зберігають індекси останньої обробленої клітинки, це необхідно для того, щоб не починати пошук завжди з першого елемента.

Замальовування проводиться з числа nextNumber2 (в квадраті).

Так як було вище сказно, що клітинки замальовуються відповідними кольорами в залежності яке зараз число обробляється, можна поетапно це зообразити зображеннями і побачити всю роботу програми.

Коли змінна nextNumber (змінна яка зберігає в собі наступне число в порядку зростання від 2 до N) рівна двійці то клітинки в яких знаходяться числа кратні їй замальовуються світло-зеленим кольором, а сама клітинка з простим числом червоним, і в масив статусів arrayOfNumbersPaintedCells, записується число 1 (один), що означає ця клітинка уже замальована (рис. 3.5).

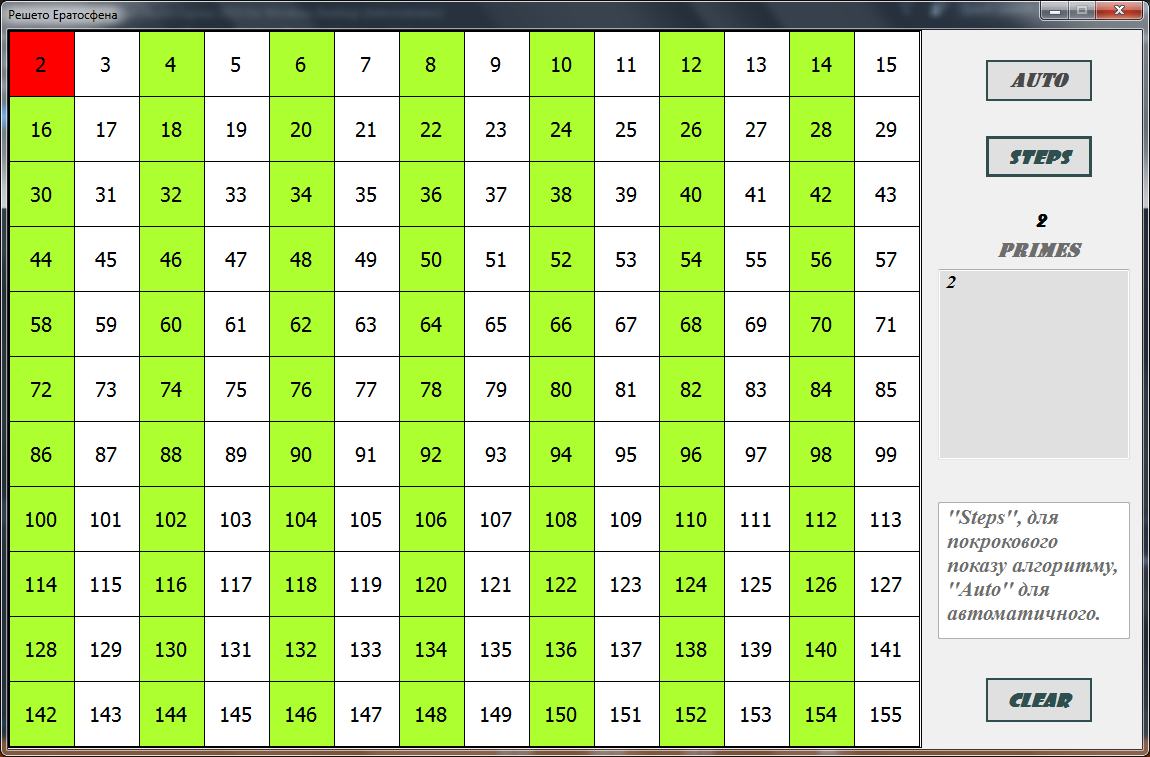


Рисунок 3.5 – Перший етап, коли nextNumber рівна 2 (два)

Лістинг 3.9 Замальовування кратних чисел, двом, зелено - жовтим

case 2: dataGridView1.Rows[i].Cells[j].Style.BackColor = Color.GreenYellow;

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Наступне число 3 (три) воно також є простим, всі числа які кратні йому починаючи з 9 (дев’яти) замальовуються синім кольором, а клітинка в якій число три, замальовується червоним, в масив статусів arrayOfNumbersPaintedCells записується число 1 (один), що означає ця клітинка уже замальована і не будуть повторно замальовуватись (рис. 3.6)

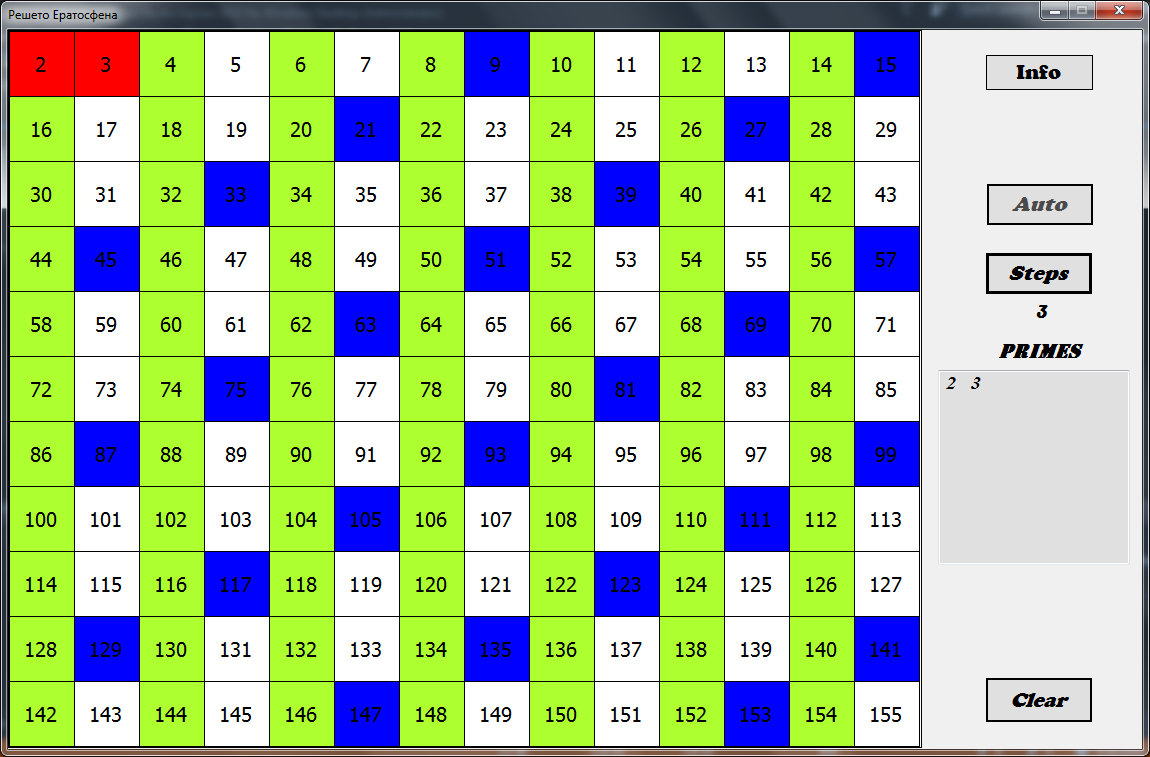


Рисунок 3.6 – Другий етап, коли nextNumber рівна 3 (три)

Лістинг 3.10 Замальовування кратних чисел, числу 3 (три), синім

case 3:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Наступне число 4 (чотири) воно вже замальоване і не є простим, пропускаємо його і переходимо до 5-ти. П’ять є простим, всі числа які кратні йому починаючи з 25 (двадцяти п’яти) замальовуються жовтим кольором, а клітинка в якій число п’ять, замальовується червоним, в масив статусів arrayOfNumbersPaintedCells записується число 1 (один), що означає ця клітинка уже замальована і не будуть повторно замальовуватись (рис. 3.7).

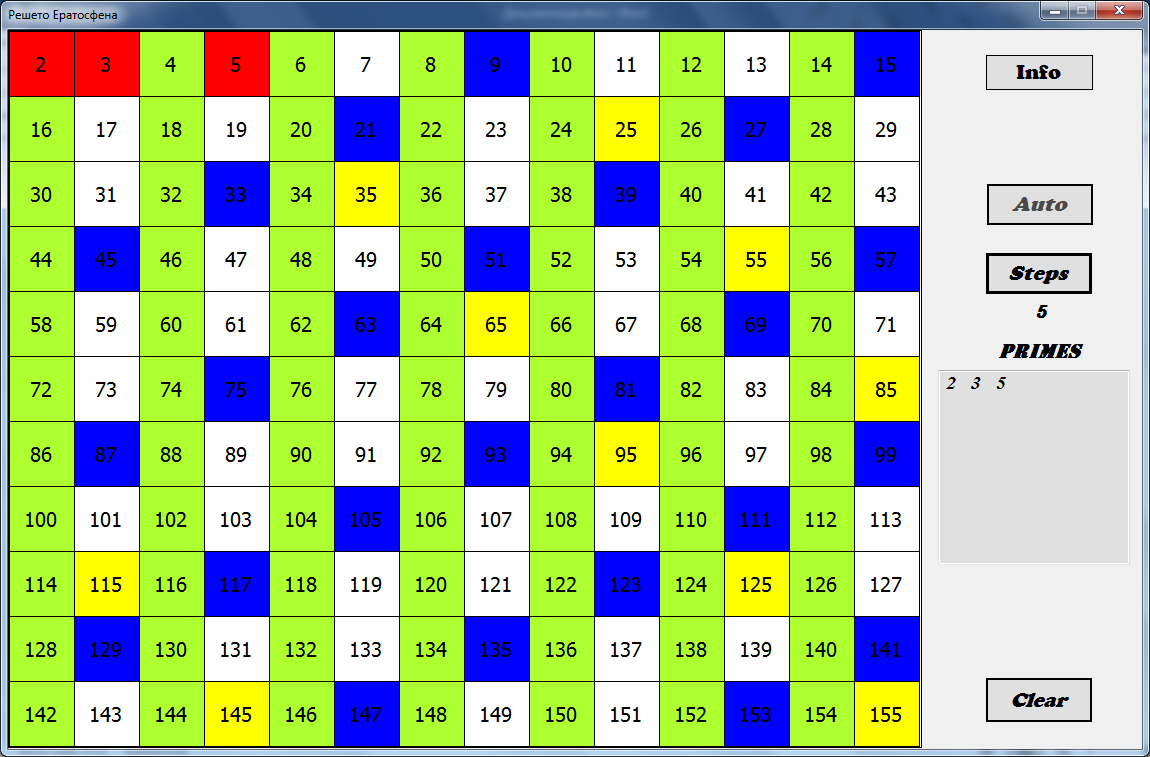


Рисунок 3.7 – Третій етап, коли nextNumber рівна 5 (п’ять)

Лістинг 3.11 Замальовування кратних чисел, числу 5 (п’ять), жовтим

case 5:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Наступне число 6 (шість) воно вже замальоване і не є простим, пропускаємо його і переходимо до 7-ми. Сім є простим, всі числа які кратні йому починаючи з 49 (сорока дев’яти) замальовуються темно-зеленим кольором, а клітинка в якій число сім, замальовується червоним, в масив статусів arrayOfNumbersPaintedCells, записується число 1 (один), що означає ця клітинка уже замальована і не будуть повторно замальовуватись (рис. 3.8).

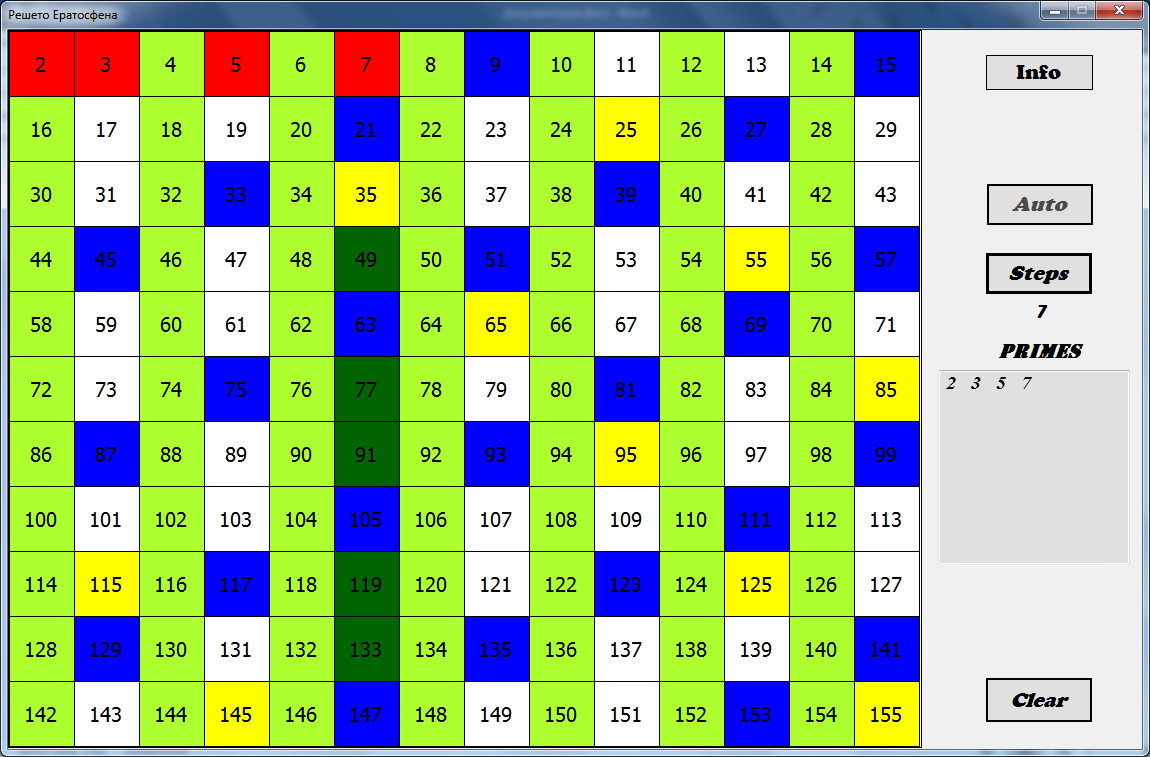


Рисунок 3.8 – Четвертий етап, коли nextNumber рівна 7 (сім)

Лістинг 3.12 Замальовування кратних чисел, числу 7 (сім), темно - зеленим

case 7:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Наступні числа 8 (вісім), 9 (дев’ять), 10 (десять), вже замальовані і не є простими, пропускаємо їх переходимо до 11-ти. Одинадцять є простим, всі числа які кратні йому починаючи з 121 (сто двадцяти одного) замальовуються фіолетовим кольором, а клітинка в якій число одинадцять, замальовується червоним, ті клітинки, які вже замальовані не будуть повторно замальовуватись (рис. 3.9).

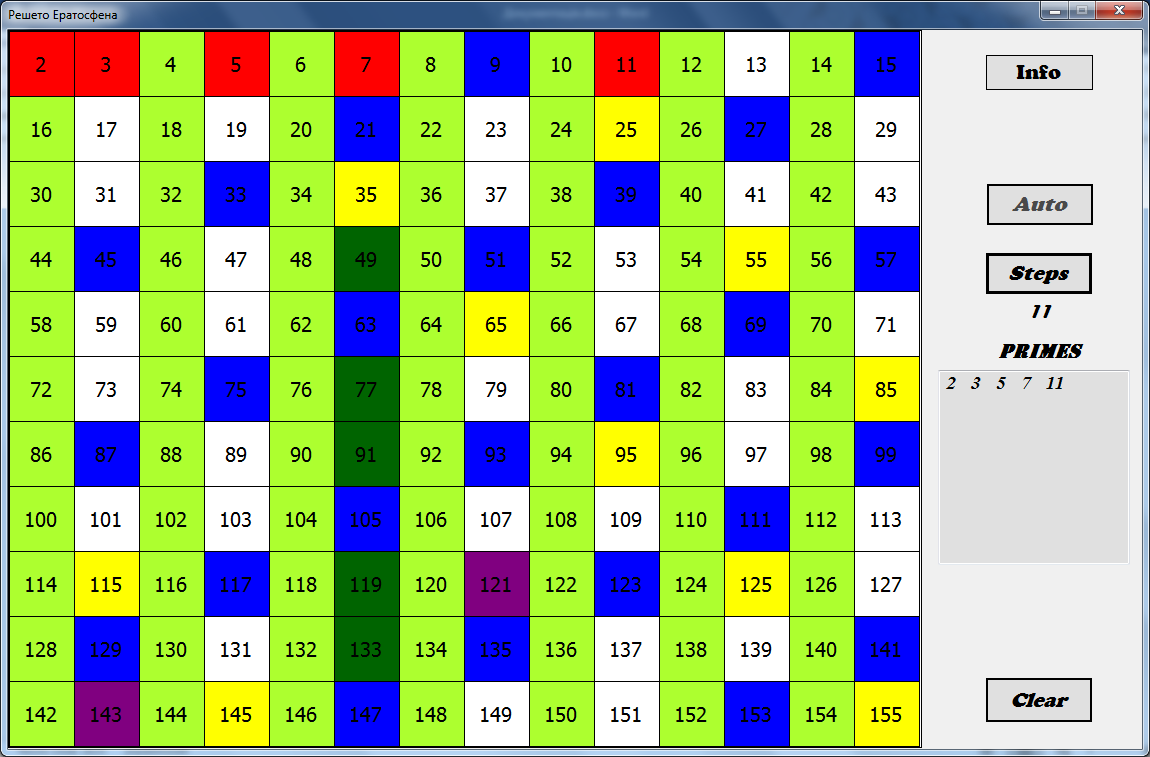


Рисунок 3.9 – П’ятий етап, коли nextNumber рівна 11 (одинадцять)

Лістинг 3.12 Замальовування кратних чисел, числу 11 (одинадцять), фіолетовим

case 11:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Наступне число 12 (дванадцять) воно вже замальоване і не є простим, на цьому робота методу який замальовував клітинки з кратними числами закінчується, так як ми дійшли числа яке вже рівне , метод який замальовує клітинки з простими числами продовжує свою роботу.

На даному етапі усі клітинки з кратними числами уже замальовані, залишилось замалювати лише клітинки з простими числами (рис. 3.10).

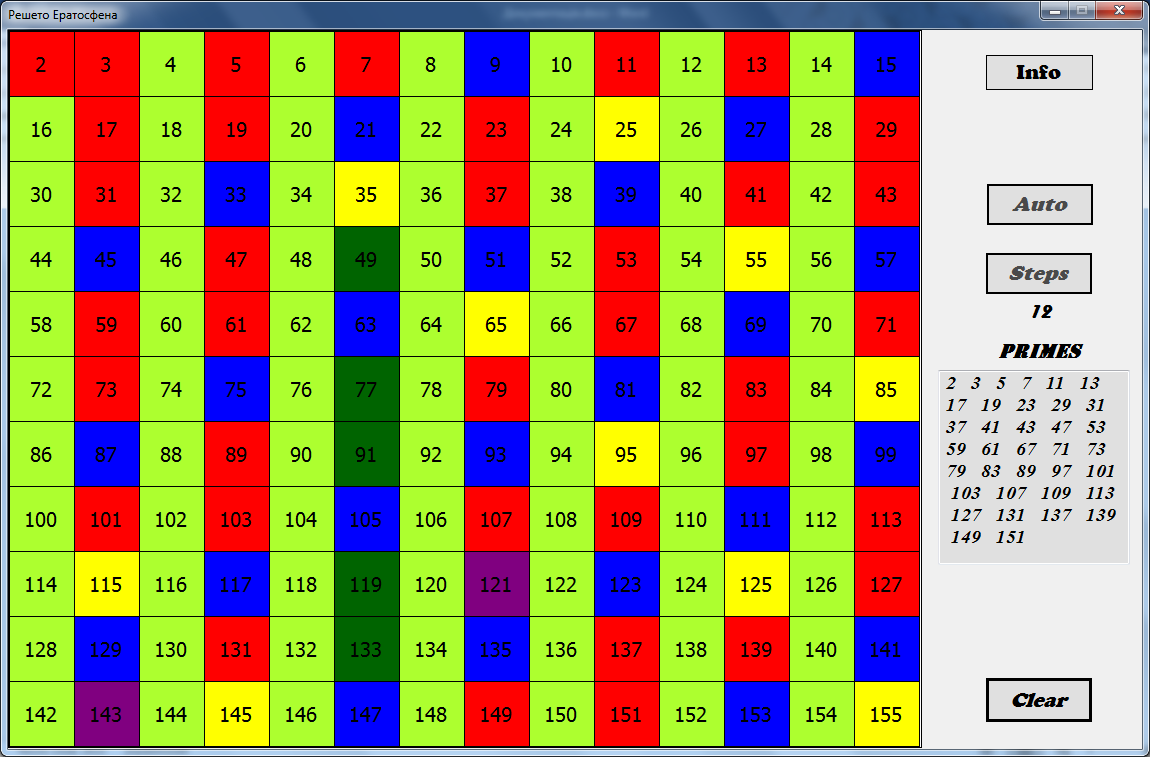


Рисунок 3.10 – Кінець пошуку

## **3.6. Висновок до третього розділу**

Програмний продукт успішно розроблено за всима необхідними критеріями, тестування пройдено успішно. Під час виконання програмного продукту поглибив знання в мові програмування C#, а також роботі з графічним інтерфейсом Windows Forms.

# **Висновок**

Основною функціональною можливістю даної програми є візуальний показ алгоритму пошуку простих чисел «Решето Ератосфена». Найбільш практичним застосуванням цієї програми є використання її у навчанні, як в школах під час вивчення простих чисел так і в університетат здебільшого на технічних спеціальностях.

Дана програма була створена з використанням інтерфейсу Windows Forms. Програма була написана на мові програмування C#. Вона практична у застосуванні і готова до використання. Інтерфейс програми зроблено простим і зрозумілим, щоб у користувача не було зайвих питань по роботі з програмою.

Перевагами розробки є те, що вона проста в експлуатації і має невисокі системні вимоги. Але як і будь-який інший програмний продукт, вона може бути покращена за рахунок допрацювання.

Написання курсової дало багато досвіду в розробці програмного забезпечення, вивчення нового матеріалу, зокрема, робота з Windows Forms у мові програмування С#. Також було закріплено навички з використання всих вивчених матеріалів за курс навчання.

# **Список використаних джерел**

1. Портал uk.wikipedia.org [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Решето_Ератосфена>. – Дата перегляду: 20.04.2019
2. Еще раз о поиске простых чисел [Електронний ресурс]. – Режим доступу: <https://habr.com/ru/post/133037/>. – Дата перегляду: 19.04.2019
3. Портал uk.wikipedia.org [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Java>. – Дата перегляду: 20.04.2019
4. Портал ru.wikipedia.org [Електронний ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/C_Sharp>. – Дата перегляду: 25.04.2019
5. Работа с формами [Електронний ресурс]. – Режим доступу: <https://metanit.com/sharp/windowsforms/2.1.php>. – Дата перегляду: 25.04.2019
6. Елемент управління DataGridView [Електронний ресурс]. – Режим доступу: <https://www.bestprog.net/uk/2018/02/17/the-datagridview-control_ua/>. – Дата перегляду: 15.03.2019
7. Гребенович Ю.Є., Онищенко Б.О.,Супруненко О.О. Методичні рекомендації до виконання та оформлення курсових робіт з дисципліни «Основи програмування», «Програмування та алгоритмічні мови», «Алгоритмізація та програмування» для студентів, які навчаються за напрямом підготовки 050101 – «Комп’ютерні науки», 050103 –«Програмна інженерія» та 040303 – «Системний аналіз» усіх форм навчання. – Черкаси: Вид. від. ЧНУ імені Богдана Хмельницького, 2015. – 32 с.

**Додаток А**

FillingCellsByColore

for (int i = indexI; i < dataGridView1.RowCount;i++ i++)

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

(int)dataGridView1.Rows[i].Cells[j].Value % nextNumber == 0 && (int)dataGridView1.Rows[i].Cells[j].Value >= nextNumber \* nextNumber && arrayOfNumbersPaintedCells[i, j] == 0

nextNumber

Color.GreenYellow

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Color.Yellow

arrayOfNumbersPaintedCells[i, j] = 1;

break;

Color.Blue

arrayOfNumbersPaintedCells[i, j] = 1;

break;

case 5:

case 3:

case 2:

1

2

4

5

3

6

5

Перед початком роботи ініціалізуємо глобальний

(для класу), двовимірний масив для збереження статусів клітинок.

+

+

+

-

-

-

1

case 11:

Color.BlueViolet arrayOfNumbersPaintedCells[i, j] = 1;

break;

Кінець

2

4

3

6

case 7:

Color.BlueViolet

arrayOfNumbersPaintedCells[i, j] = 1;

break;

**Додаток Б**

1. **Лістинг коду який заповнює таблицю числами**

void PrintArrayWihtNumbers()

{

int z = 2;

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

{

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

{

dataGridView1.Rows[i].Cells[j].Value = z;

z++;

}

}

}

1. **Лістинг коду кнопки, яка очищає таблицю**

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

{

// очищає (заммальовує) клітинки білим кольором

ClearAllCells();

// обнуляємо індекси

indexI = 0;

indexJ = 0;

// змінна nextNumber стає рівна двом

nextNumber = 2;

SetDoubleBuffer(dataGridView1, false);

// кнопки стають активними

StepsButton.Enabled = true;

AutoButton.Enabled = true;

StopButton.Enabled = true;

ContinueButton.Enabled = true;

AutoButton.BringToFront();

ContinueButton.SendToBack();

// зупиняємо таймер

timer1.Stop();

// змінюємо тік таймера

timer1.Interval = 1400;

// очищяємо textBox-си

textBox1.Clear();

textBox2.Clear();

// кнопка ClearButton стає не активною

ClearButton.Enabled = false;

}

1. **Лістинг коду кнопки, яка показує алгоритм покроково**

// кнопка для покрокового показу алгоритму

private void StepsButton\_Click\_1(object sender, EventArgs e)

{

// додає прості числа до textBox-са

// замальовує клітинки з простими числами

ClearButton.Enabled = true;

StopButton.Enabled = false;

ContinueButton.Enabled = false;

AutoButton.Enabled = false;

// якщо змінна nextNumber > 11, кнопка StepsButton стає не активною, і запускається робота таймера

FillingCellsByColore();

FillingCellsWithPrimes();

// виводить номер числа яке зараз обробляється в textBox

if (nextNumber < (int)Math.Sqrt(155))

{

if (arrayOfNumbersPaintedCells[indexI, indexJ] == 0 && (int)dataGridView1.Rows[indexI].Cells[indexJ].Value == nextNumber)

{

textBox1.Text = nextNumber.ToString();

}

}

else if (nextNumber == (int)Math.Sqrt(155))

{

StepsButton.Enabled = false;

timer1.Start();

}

nextNumber++;

}

# **Додаток В**

1. **Лістинг коду роботи таймера**

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

{

if (nextNumber >= 152) // якщо nextNumber більше-рівне 152 зупиняємо таймер

{

timer1.Stop();

ClearButton.Enabled = true;

StopButton.Enabled = false;

ContinueButton.Enabled = false;

MessageBox.Show("Клiтинки червоного кольору - простi числа.");

}

if (nextNumber > 6 && nextNumber < 11)

{

timer1.Interval = 700;

}

if (nextNumber == (int)Math.Sqrt(155) + 1)

{

SetDoubleBuffer(dataGridView1, true);

SetDoubleBuffer(textBox2, true);

timer1.Interval = 70;

}

if (nextNumber < (int)Math.Sqrt(155))

{

FillingCellsByColore();

}

FillingCellsWithPrimes();

nextNumber++;

}

1. **Лістинг коду, який замальовує клітинки з кратними числами**

for (int i = indexI; i < dataGridView1.RowCount; i++)

{

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

{

// якщо елемент таблиці ділиться націло на число, яке обробляється в певний момент часу, і цей елемент більший- рівний ніж це число в квадраті, і статус клітинки рівний нулю

if ((int)dataGridView1.Rows[i].Cells[j].Value % nextNumber == 0 && (int)dataGridView1.Rows[i].Cells[j].Value >= nextNumber \* nextNumber && arrayOfNumbersPaintedCells[i, j] == 0)

{

switch (nextNumber)

{

case 2:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

case 3:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

case 5:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

case 7:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

case 11:

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

arrayOfNumbersPaintedCells[i, j] = 1;

break;

default:

break;

}

}

}

}

1. **Лістинг коду, який замальовує клітинки з простими числами**

void FillingCellsWithPrimes()

{

dataGridView1.ClearSelection();

bool b = false;

for (int i = indexI; i < dataGridView1.RowCount; i++)

{

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

{

if (arrayOfNumbersPaintedCells[i, j] == 0 && (int)dataGridView1.Rows[i].Cells[j].Value == nextNumber)

{

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

textBox2.Text += nextNumber.ToString() + " ";

indexI = i;

indexJ = j;

b = true;

break;

}

else if (arrayOfNumbersPaintedCells[i, j] == 1 && (int)dataGridView1.Rows[i].Cells[j].Value == nextNumber)

{

indexI = i;

indexJ = j;

b = true;

break;

}

}

if (b)

{

break;

}

}

ChooseNeededIndexj();

}