ЗМІСТ

[ВСТУП 2](#_Toc41061749)

[1. РОБОЧИЙ ПРОЦЕС ПРОГРАМИ 4](#_Toc41061750)

[2. ПОСТАНОВКА ЗАДАЧІ 5](#_Toc41061751)

[3. АЛГОРИТМІЧНА МОДЕЛЬ ВІДОБРАЖЕННЯ ОСНОВНИХ ГЕОМЕТРИЧНИХ ЕЛЕМЕНТІВ 6](#_Toc41061752)

[4. АЛГОРИТМІЧНА МОДЕЛЬ ВІДОБРАЖЕННЯ ФРАКТАЛЬНИХ ФІГУР 9](#_Toc41061753)

[4.1 UML діаграма класів 12](#_Toc41061754)

[5. ОПИС ПРОГРАМИ 13](#_Toc41061755)

[6. КЕРІВНИЦТВО РОЗРОБНИКА 19](#_Toc41061756)

[7. КЕРІВНИЦТВО КОРИСТУВАЧА 21](#_Toc41061757)

[ВИСНОВОК 22](#_Toc41061758)

[ВИКОРИСТАНІ ДЖЕРЕЛЛА 23](#_Toc41061759)

[ДОДАТОК А 24](#_Toc41061760)

[ДОДАТОК Б 28](#_Toc41061761)

# ВСТУП

На сьогоднішній день комп’ютерні технології стрімко розвиваються. Дедалі більше інструментів, переходять в комп’ютерний простір. Письмо, малювання, креслення, моделювання – усе це можна робити в комп’ютері за допомогою відповідних програм.

*Актуальність теми.* Сьогодні існує багато різноманітних графічних редакторів. Перший графічний редактор було створено в 1963 році Айваном Сазерлендом. Це був примітивний редактор на базі програмно-апаратного комплексу Sketchpad. В ньому була можливість створення найпростіших геометричних фігур, таких як точки лінії, круги за допомого спеціально оптичного пера. Далі технології створення графічних зображень почали стрімко розвиватися і на сьогоднішній день маємо велике різноманіття таких програмних засобів.

Не дивлячись на великий вибір редакторів, кожен з них має свій унікальний функціонал, особливості, які можуть використовуватися в різних цілях: редагування зображень, створення картин та логотипів. Актуальність теми полягає в тому, що не існує графічного редактора, який би підійшов для будь-яких цілей, і якщо для одного художника влаштовує один редактор, то іншого він може не влаштовувати. З цього виходить те, що не існує універсальних графічних редакторів, тому створення нових програм збільшує множину можливих редакторів, яких може кожен обрати, для виконання своїх завдань.

*Мета роботи.* Мета курсової роботи полягає в розробці простого графічного редактору з використання принципу об’єктно орієнтованого програмування, для вдосконалення навичок роботи з ним та отримання досвіду практичного застосування даної концепції.

Під час роботи над цим проектом метою є:

1. Навчитися працювати з концепцією об’єктно-орієнтованого програмування, та здобути практичні навички роботи з нею.
2. Дослідити принцип роботи середовища розробки Microsoft Visual Studio.
3. Детально вивчити інструменти створення програм на основі Windows Form з використанням мови С#.
4. За допомогою вибраної технології створити графічний редактор, на головному вікні якого буде робоча панель, над якою буде знаходитись панель інструментів.
5. Реалізувати можливість збереження та відкриття створених зображень.
6. Розробити механізм створення робочого полотна заданого розміру.
7. Розробити інструменти створення геометричних примітивів з можливістю вибору кольору та товщини кісті.
8. Розробити інструменти для створення найпростіших, двовимірних, фрактальних фігур, таких як трикутник та килим Серпинського.

# 1. РОБОЧИЙ ПРОЦЕС ПРОГРАМИ

Графічний редактор – це комп’ютерна програма, яка дає змогу редагувати та створювати графічні зображення. Процес малювання відбувається на робочому полі, заданих розмірів, за допомогою комп'ютерної миші. Графічний редактор зазвичай має панель інструментів, яка має інструменти створення геометричних фігур, таких як: ліній, точок, кривих, тощо. Редактор повинен мати функції створення, збереження та відкриття рисунка. Функціонал, програми зазвичай залежить від призначення програми.

# 2. ПОСТАНОВКА ЗАДАЧІ

Головною метою курсової роботи є розробка програми графічного редактора, з можливістю створення геометричних фігур. Для створення програми потрібно виконати наступні кроки:

1. Розробити відображення робочого поля редактора
2. Розробити панель інструментів
3. Розробити інструмент для створення та налаштування фрактальних фігур
4. Розробити функції збереження створеного зображення

Для досягнення поставленої мети потрібно вирішити наступні проблеми.

1. Розробити наступні алгоритми:
   1. Коректне відображення вибраних фігур на робочому полі
   2. Вибір інструментів відображення в панелі інструментів
   3. Алгоритм створення фрактальних фігур
   4. Збереження та завантаження зображення
2. Реалізувати зв’язки між формами програми
3. Реалізувати візуальний інтерфейс редактора, використовуючи
4. Реалізувати клас фракталів, з методами відтворення відповідних зображень

# 3. АЛГОРИТМІЧНА МОДЕЛЬ ВІДОБРАЖЕННЯ ОСНОВНИХ ГЕОМЕТРИЧНИХ ЕЛЕМЕНТІВ

Програма реалізована на об’єктно-орієнтованій мові програмування С#. Проект складається з трьох класів, що відповідають за форми, та класу що відповідає за відображення фрактальних фігур.

Оскільки мова C# є об’єктно орієнтованою мовою, тому кожне вікно буде представляти собою окремий клас. Так код буде займати менше місця та буде більш структурованим, оскільки методи кожного форми будуть знаходитися у окремому класі. Для спрощення задачі побудови інтерфейсу та загального функціоналу програми, було використано модульну платформу .Net core.

Для створення інтерфейсу вікон, кожен клас який відповідає за вікно наслідує клас Form платформи .Net core. Для підключення класу було використано прості імен [System.Windows.Forms](https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms?view=netcore-3.1).dll. Відповідно для відображення об’єктів на головній формі було використано наступні класи з вибраної платформи:

* **Panel** – для відображення панелі інструментів та об’єднання об’єктів вибраної платформи
* **MenuStrip** – забезпечує відображення та функціонування випадаючого меню в верхньому лівому кутку форми
* **ToolStripMenuItem** – забезпечує відображення та функціонал кнопок в випадаючого меню
* **ToolStrip** – забезпечує відображення та функціонал панелі з піктограмами в панелі інструментів
* **ToolStripButton** - забезпечує відображення та функціонал піктограм в панелі інструментів
* **TrackBar** – забезпечує функціонал повзунка для налаштування товщини кисті, та коефіцієнту кроку для побудови фрактальних фігур
* **SaveFileDialog** – забезпечує функцію вибору папки для збереження
* **TextBox** – забезпечує відображення тексту з можливістю редагування, використовується для зміни товщини кисті та коефіцієнту кроку для фрактальних фігурю
* **CheckBox** – відображає текст, який можна відмітити галочкою, використовується для вмикання функцій відображення фрактальних фігур
* **Label** – відображає інформативний текст
* **ComboBox** – відображає випадаюче меню на формі створення нового рисунка, для можливості вибору шаблонних розмірів
* **OpenFileDialog** – забезпечує функцію відкриття зображень
* **PictureBox** – виконує функцію відображення поля, на якому буде створюватися рисунок
* **ColorDialog** – відображає вікно з можливістю вибору кольору

Вище зазначені класи використовувалися для спрощення, створення та відображення, графічного інтерфейсу програми, оскільки вони мають в собі методи обробки маніпуляцій з їх об’єктами, тобто натискань тощо.

В програмі реалізовано 7 можливих об’єктів. Для їх відображення було використано клас Bitmap – для зберігання растрового зображення в пам’яті, та клас Graphics, який забезпечує створення геометричних примітивів(прямих, точок, прямокутників, сплайнів, тощо). Ці класи належать до простору імен System.Drawing. Перший об’єкт це звичайна крива лінія, яка відображається при на робочому полотні за допомогою комп’ютерної миші. До уваги бралося два основних алгоритма відображення кривої. Перший – точковий, тобто під час натискання кнопки миші, зчитуються координати курсора та на їх місці ставляться точки відповідної заданої товщини. Недолік цього методу в тому, що при різкій змінні положення курсора, можуть утворюватися пробіли в кривій, що нам не потрібно. Другий спосіб – лінійний, тобто, під час натискання по робочому полю, буде зчитуватися, перша координата та при русі курсору по робочому полі, будуть зчитуватися координати точок і кожні дві сусідні точки з’єднуватися прямою. Еліпс та прямокутник відображатиметься за допомогою вбудованого класу Graphics. Рівнобедрений, прямокутний трикутник та ромб за допомогою задання точок відносно початкової та кінцевої так, щоб під час їх з’єднання прямим ми отримали потрібну фігуру (Додаток А).

Для реалізації фрактальних фігур було вирішено використовувати CheckBox, при натисканні на який, відкривається меню налаштування фрактальних фігур, а саме коефіцієнт від 1 до 1000, який визначає крок нових точок при побудові. Даний вид фігур доступний лише для чотирикутників та трикутників.

# АЛГОРИТМІЧНА МОДЕЛЬ ВІДОБРАЖЕННЯ ФРАКТАЛЬНИХ ФІГУР

Фрактал — нерегулярна, самоподібна структура. В широкому розумінні фрактал означає фігуру, малі частини якої в довільному збільшенні є подібними до неї самої. Термін фрактал увів [1975](https://uk.wikipedia.org/wiki/1975) року [Бенуа Мандельброт](https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%BD%D1%83%D0%B0_%D0%9C%D0%B0%D0%BD%D0%B4%D0%B5%D0%BB%D1%8C%D0%B1%D1%80%D0%BE%D1%82" \o "Бенуа Мандельброт).

Фрактал – це доволі широке поняття, на сьогоднішній день фрактали широко використовуються в різноманітних напрямках. За допомогою фракталів описують складні графічні структури, які мають самоподібну структуру. Фрактали найчастіше використовуються в комп’ютерній графіці, оскільки дають можливість описати складні об’єкти простимим відносно невеликими формулами.

В даній роботі реалізовано найпростіші фрактальні фігури такі, килим та серветка Серпінського.

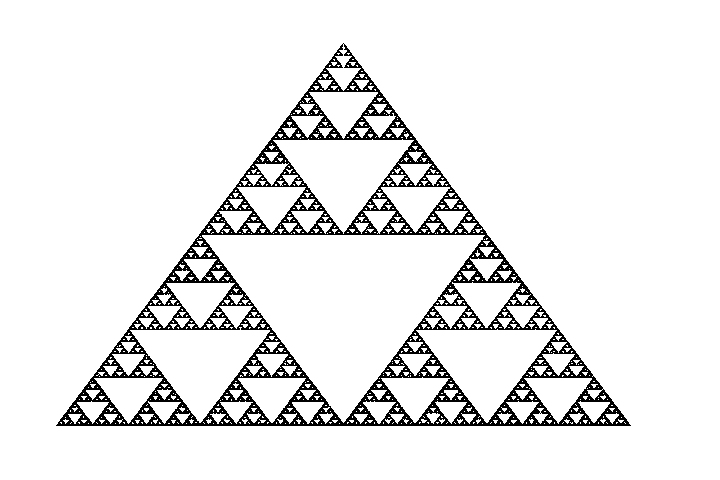


Рис 4.1 Серветка або трикутник Серпінського

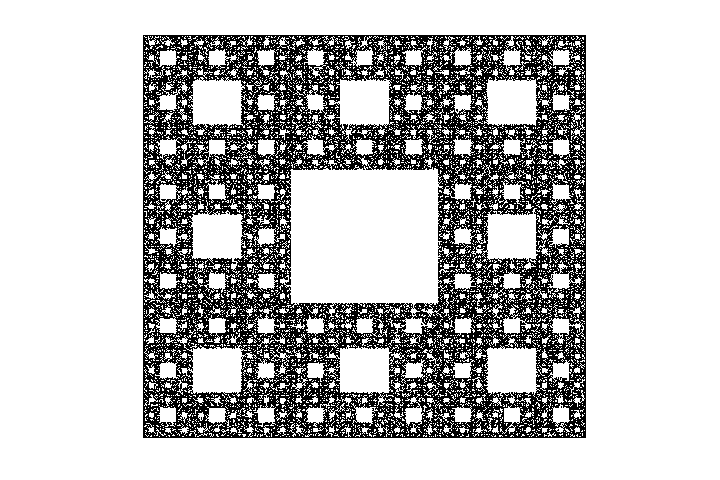


Рис 4.2 Килим Серпінського

Для відображення даних фігур було обрано метод хаоса, оскільки він відносно простий для розуміння та найголовніше демонструє, як випадковість генерує впорядкованість.

В роботі для побудови даних фігур метод хаосу виглядає так:

1. Задаються координати точок атракторів:
   1. Для трикутника вершини
   2. Для квадрата – вершини та середини сторін
2. Обирається деяка початкова, випадкова точка(може бути одна з вершин)
3. Далі починається цикл побудови фігури:
   1. За допомогою генерації випадкових чисел обирається довільна точка атрактор
   2. Ставиться нова точка по середині між початковою та обраною точкою атрактором(тобто координати такої точки будуть

Де – координата останньої точки, – координата обраної точки атрактора, коефіцієнт кроку.

Такий цикл повторюється довільну кількість раз, і чим більша кількість ітерацій тим чіткішим вийде зображення. У програмі алгоритм реалізується таким чином, що якщо було увімкнено режим побудови фрактальних фігур і якщо обрана фігура чотирикутник або трикутник, то запам’ятовуються координати атракторів відповідної фігури і викликається метод, який запускає цикл побудови фрактала методом хаоса, який реалізований в класі Fractal (Додаток Б). Формула координат нової точки реалізованому методі дещо відрізняється і має вигляд:

Де С – коефіцієнт кроку, який користувач має змогу налаштувати в панелі інструментів.

## 4.1 UML діаграма класів

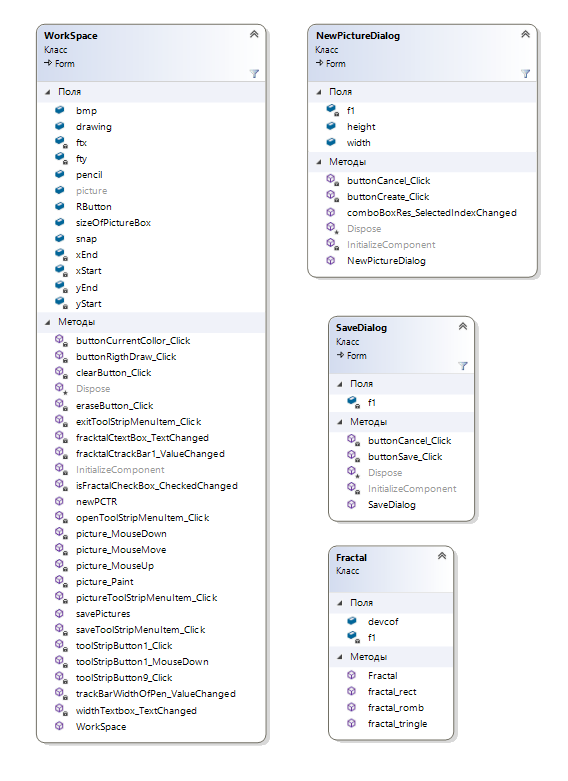


Рис. 4.1.1 Діаграма класів програми

# ОПИС ПРОГРАМИ

Після запуску програми повинно з’являтися робоче поле, на якому буде відображатися зображення. Також наявна кнопка очищення робочого поля, та кнопка гумки. Програма матиме режим відтворення фрактальних фігур, тобто трикутників та прямокутників. Програма має можливість зберігати та відкривати створені зображення. Головне вікно редактора представлено на (Рис. 5.1)

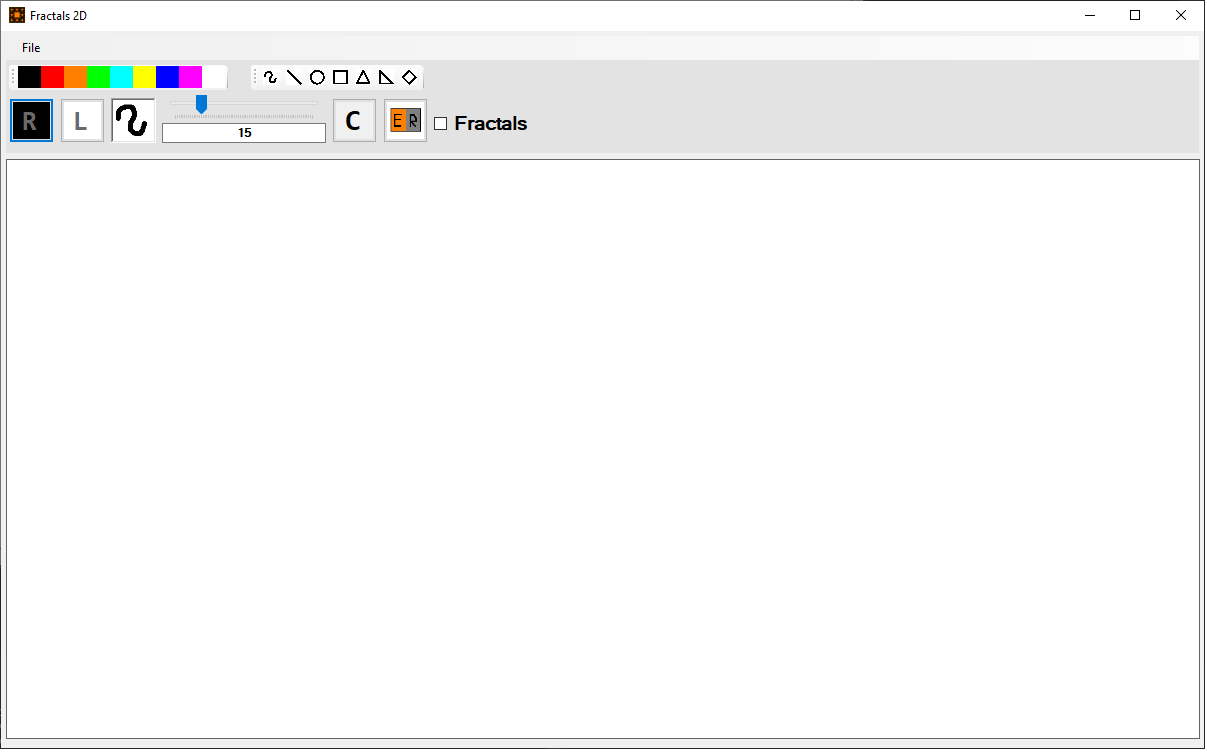


Рис 5.1 Головне вікно редактора

Над робочою частиною програми знаходиться панель інструментів, а саме вибір фігури, кольору та товщини ліній.

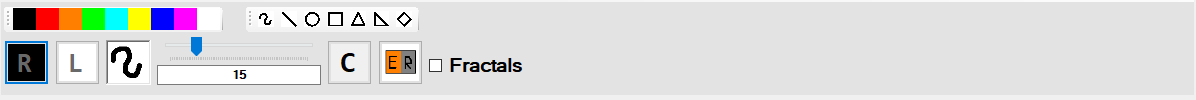


Рис. 5.2 Панель інструментів

Для створення графічного зображення потрібно обрати колір, вибрати товщину лінії та фігуру. Також наявна кнопка очищення робочого поля, та кнопка гумки. Пояснення панелі інструментів (Рис. 5.3).

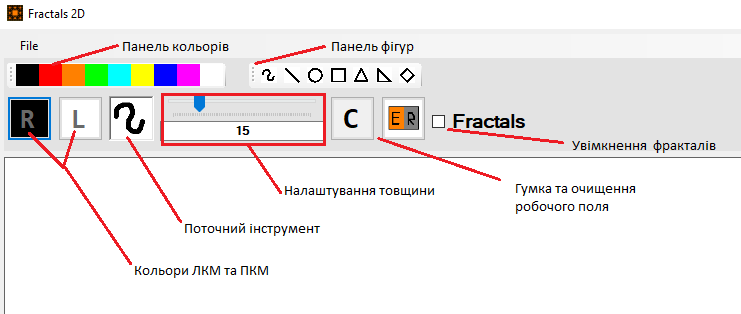


Рис. 5.3 Підпис пояснення панелі

Далі відповідно натисканням та перетягуванням курсора по робочому полю відтворювати потрібне зображення. Кисть має два кольори, один – основний, при натисканні лівої кнопки миші, другий – допоміжний при натисканні лівої кнопки миші. Користувач має можливість обрати один із стандартних кольорів з панелі, або натиснувши на кнопки відображення поточного кольору (ПКМ - R та ЛКМ – L) , обрати будь-який інший колір (Рис 5.4). Варто зазначити що усі іконки, які використовуються в редакторі було створено в ньому ж.

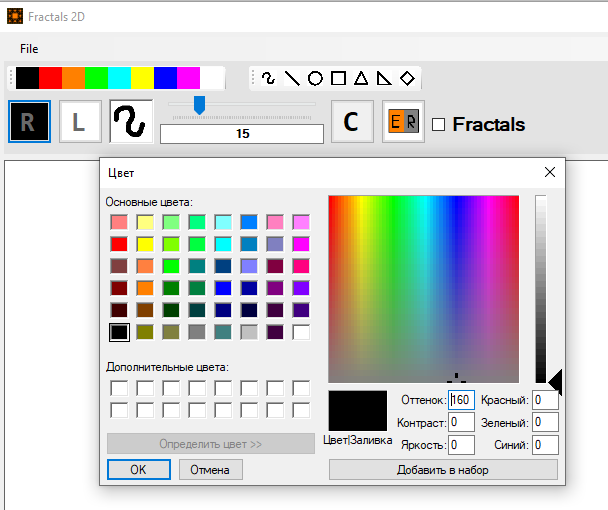


Рис. 5.4 Вікно вибору кольору

Програма має режим відтворення фрактальних фігур, тобто трикутників та прямокутників, для того що його увімкнути потрібно поставити галочку Fractals (Рис. 5.5) після чого з’явиться панель налаштування коефіцієнту кроку побудови. Також має можливість зберігати та відкривати створені зображення (Рис. 5.6).

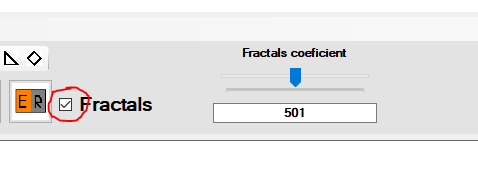


Рис. 5.5

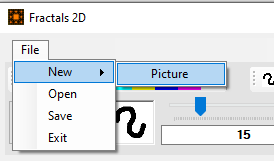


Рис. 5.6 Меню

Приклади побудованих зображень при використанні звичайних фігур, тобто з вимкненим режимом фракталів (Рис 5.7).

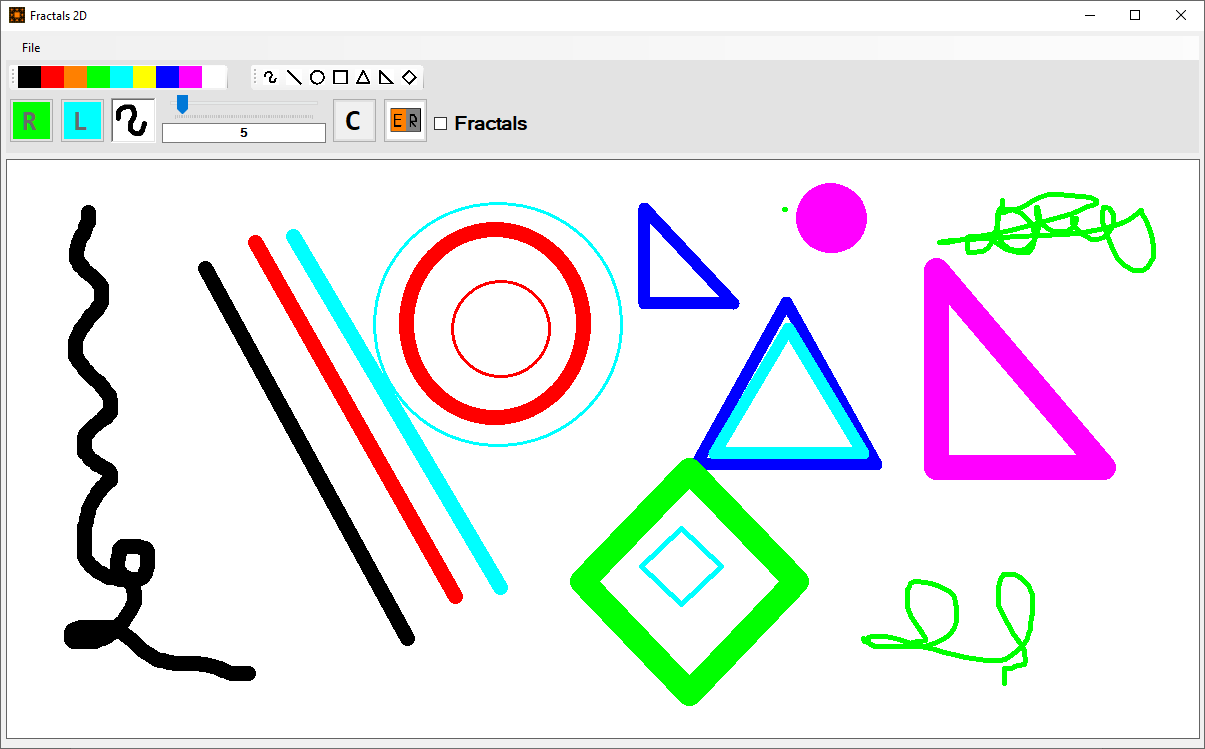


Рис. 5.7 Приклад роботи програми в звичайному режимі

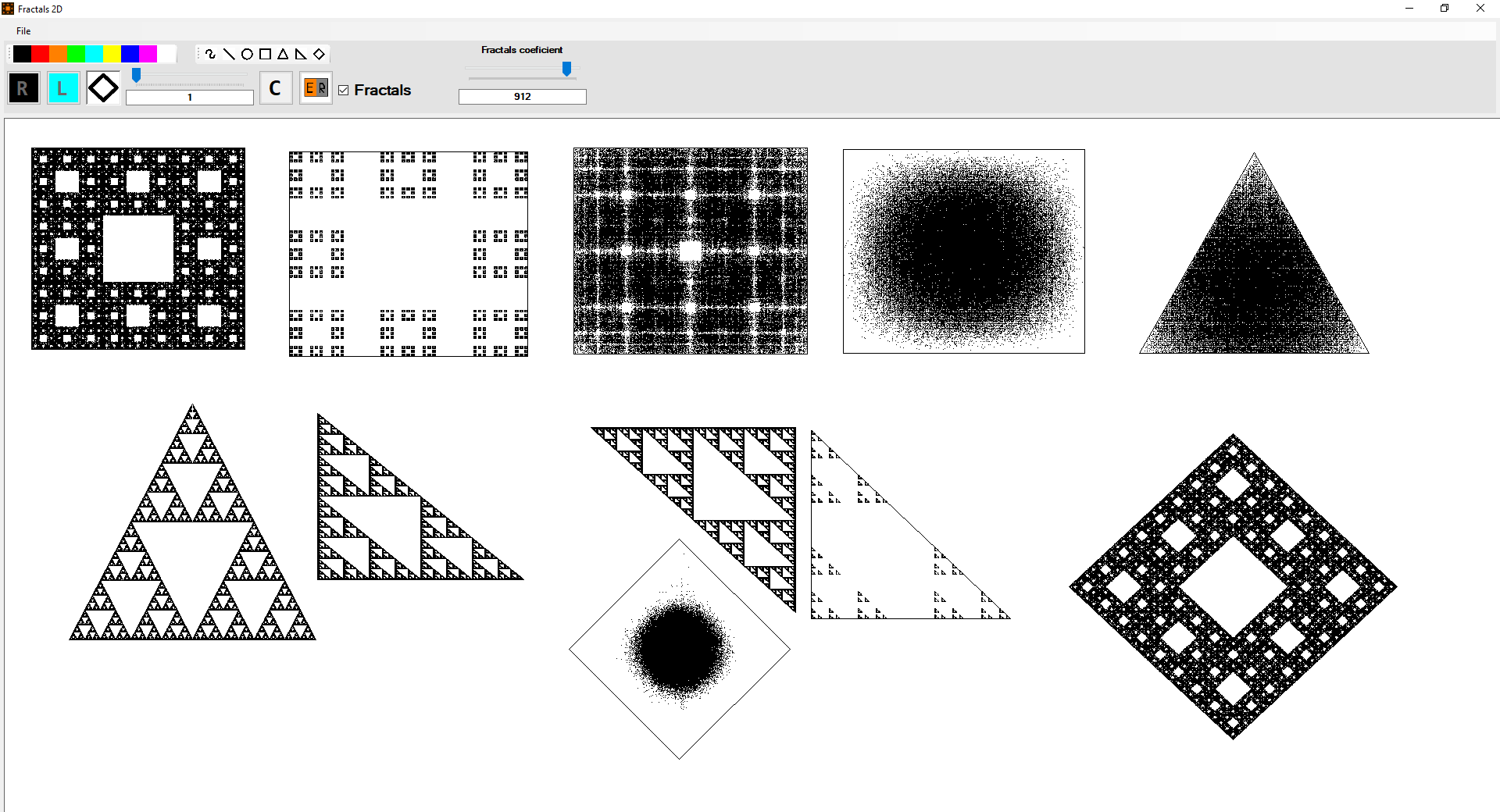


Рис 5.8 Приклад роботи програми в режимі відтворення фракталів

Як можна побачити, типові фігури Серпінського виходять тільки при певних коефіцієнтах. Для трикутника , для квадрата :

В інших випадках ми отримаємо інше, проте також доволі цікаве зображення.

* Якщо ми отримаємо розмите зображення, при чому чим більший коефіцієнт тим більш точки збираються до центру фігури.
* Якщо ж фрактальна структура зберігається, проте вона стає більш дрібною.

Також програма має вікно створення нового рисунка. Після натискання кнопки New->Picture в меню File (Рис. 5.6), з’являється діалогове вікно (Рис. 5.9), в якому запитується чи хоче користувач зберегти поточне зображення. Якщо користувач погодиться то відбудиться збереження файлу в обраній папці.

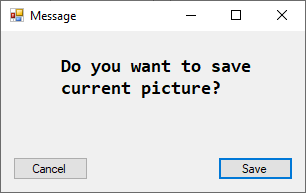


Рис. 5.9 Діалогове вікно збереження рисунка перед створенням нового

Далі після збереження відкривається вікно створення нового рисунку, в якому є можливість обрати розширення в пікселях (Рис. 5.10). Користувач має можливість обрати одне із запропонованих розширень, або вписати своє. Після чого створиться нове робоче поле заданого розміру.

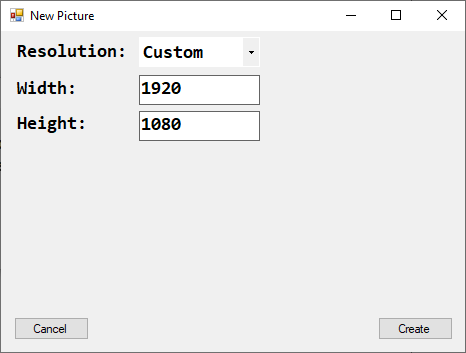


Рис. 6.10 вікно створення нового зображення

# КЕРІВНИЦТВО РОЗРОБНИКА

Даний графічний редактор написаний на мові програмування С#. Концепція програмування програми – об'єктно–орієнтована. В якості середовища виступає Microsoft Visual Studio Community 2019. Проект має тип Windows Forms і залежить від платформи .Net Framework. Дана платформа забезпечує роботу інтерфейсу програми. Офіційне середовище MS Visual Studio Community є безкоштовним та доступне для завантаження, на офіційному сайті [*Microsoft*](https://visualstudio.microsoft.com/ru/thank-you-downloading-visual-studio/?sku=Community&rel=16).

Програма складається з трьох класів, які відповідають за форми, та класу, який відповідає за обрахунок фракталів, та файлу Program , який запускає роботу основної форми. На рис. 7.1 зображена архітектура проекту.

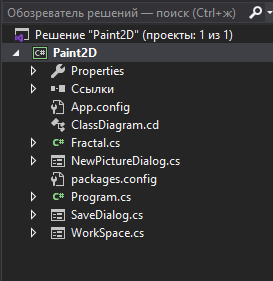


Рис. 6.1

Клас WorkSpace – відображає робоче вікно, у ньому знаходяться усі методи обробки подій, що відбуваються на формі. NewPicturesDialog – клас, що відповідає за діалогове вікно, створення нового рисунка та SaveDialog – відповідає за діалогове вікно, що запитує користувача про збереження. Клас Fractal містить методи побудови фракталів. Для простоти розуміння коду функція кожного методу підписана коментарем.

Програма реалізована за допомогою платформи .Net Core 3.1, яка містить в собі потрібні для роботи бібліотеки класів, такий, як клас Form, від якого унаслідуються класи вікон програми.

# КЕРІВНИЦТВО КОРИСТУВАЧА

Графічний редактор “Fractals 2D” доступний усім віковим категоріям, оскільки має прости та інтуїтивно зрозумілий інтерфейс.

Для роботи програми потрібна операційна система Windows 10. Програма запускається за допомогою файлу з розширенням (.exe). Для повноцінної роботи з програмою потрібно мати такі периферійні пристрої, як комп’ютерна мишка, та клавіатура.

Можливості редактора:

* Створювати найпростіші графічні об’єкти лінії, прямокутники, трикутники, криві
* Обирати колір фігур та ліній
* Налаштовувати товщину ліній
* Створення фрактальних, двухвимірні об’єкти
* Створення, завантаження та збереження проекту

В розділі Опис програми функціонал програми розписано детальніше, та продемонстровано приклади роботи.

# ВИСНОВОК

В ході виконання курсової роботи було реалізовано програма графічний редактор. Під час виконання роботи було освоєно основні принципи побудови програм за допомогою використання принципів об’єктно-орієнтованого програмування, та будовою класів. Ознайомився та виробив основні навички роботи з мовою програмування С# в середовищі розробки програмного забезпечення Microsoft Visual Studio 2019. Ознайомився з платформою .NET, та класами що вона надає. Навчився працювати з конструктором інтерфейсу, та створювати віконну оболонку програми. Розроблений графічний редактор працює коректно. Інтерфейс програми інтуїтивно зрозумілий. Програма має базові можливості зміни кольору та товщини ліній. Також досліджено та реалізовано побудову найпростіших фрактальних фігур, такий як серветка та килим Серпінського. Досліджено принцип роботи методу хаоса для побудови найпростіших фракталів.

# ВИКОРИСТАНІ ДЖЕРЕЛЛА

1. Работа с формами. Основи форм. Посилання на ресурс: <https://metanit.com/sharp/windowsforms/1.1.php>
2. Технічна документація Microsoft Build. Посилання на ресурс: <https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.form?view=netcore-3.1>
3. Stack overflow. Поисилання на ресурс: <https://ru.stackoverflow.com/>
4. Варіації теми (прямокутні) треми. Посилання на ресурс: <http://www.cut-the-knot.org/Curriculum/Geometry/SqStrFSM.shtml>
5. Фрактал. Посилання на ресурс: <https://uk.wikipedia.org/wiki/%D0%A4%D1%80%D0%B0%D0%BA%D1%82%D0%B0%D0%BB>
6. Chaos Game – Numberphile. Посилання на ресурс: <https://www.youtube.com/watch?v=kbKtFN71Lfs>

# ДОДАТОК А

Код методів що відповідають за малювання(*picture\_Paint, picture\_MouseMove,* *picture\_MouseUp*) Клас *WorkSpace*

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

{

if (drawing)

{

Graphics g = Graphics.FromImage(bmp);

Pen penL = new Pen(buttonCurrentCollor.BackColor, trackBarWidthOfPen.Value);

Pen penR = new Pen(buttonRigthDraw.BackColor, trackBarWidthOfPen.Value);

penL.StartCap = System.Drawing.Drawing2D.LineCap.Round;

penL.EndCap = System.Drawing.Drawing2D.LineCap.Round;

penR.StartCap = System.Drawing.Drawing2D.LineCap.Round;

penR.EndCap = System.Drawing.Drawing2D.LineCap.Round;

if (bmp != null)

{

switch (pencil)

{

//Крива

case 1:

if (RButton) g.DrawLine(penL, xStart, yStart, xEnd, yEnd);

else if (!RButton) g.DrawLine(penR, xStart, yStart, xEnd, yEnd);

xStart = xEnd;

yStart = yEnd;

break;

//Прямий відрізок

case 2:

if (RButton) g.DrawLine(penL, xStart, yStart, xEnd, yEnd);

else if (!RButton) g.DrawLine(penR, xStart, yStart, xEnd, yEnd);

break;

//Коло

case 3:

if (RButton) g.DrawEllipse(penL, xStart, yStart, xEnd - xStart, yEnd - yStart);

else if (!RButton) g.DrawEllipse(penR, xStart, yStart, xEnd - xStart, yEnd - yStart);

break;

//Прямокутник

case 4:

if (xStart < xEnd && yStart < yEnd)

{

if (RButton) g.DrawRectangle(penL, xStart, yStart, xEnd - xStart, yEnd - yStart);

else if (!RButton) g.DrawRectangle(penR, xStart, yStart, xEnd - xStart, yEnd - yStart);

ftx[1] = xStart;

fty[1] = yStart;

ftx[2] = xEnd;

fty[2] = yEnd;

}

else if (xStart < xEnd && yStart > yEnd)

{

if (RButton) g.DrawRectangle(penL, xStart, yEnd, xEnd - xStart, yStart - yEnd);

else if (!RButton) g.DrawRectangle(penR, xStart, yEnd, xEnd - xStart, yStart - yEnd);

ftx[1] = xStart;

fty[1] = yEnd;

ftx[2] = xEnd;

fty[2] = yStart;

}

else if (xStart > xEnd && yStart < yEnd)

{

if (RButton) g.DrawRectangle(penL, xEnd, yStart, xStart - xEnd, yEnd - yStart);

else if (!RButton) g.DrawRectangle(penR, xEnd, yStart, xStart - xEnd, yEnd - yStart);

ftx[1] = xEnd;

fty[1] = yStart;

ftx[2] = xStart;

fty[2] = yEnd;

}

else

{

if (RButton) g.DrawRectangle(penL, xEnd, yEnd, xStart - xEnd, yStart - yEnd);

else if (!RButton) g.DrawRectangle(penR, xEnd, yEnd, xStart - xEnd, yStart - yEnd);

ftx[1] = xEnd;

fty[1] = yEnd;

ftx[2] = xStart;

fty[2] = yStart;

}

break;

// Трикутник рівнобедренний

case 5:

if (RButton)

{

g.DrawLine(penL, xStart, yEnd, xEnd, yEnd);

g.DrawLine(penL, xStart, yEnd, xStart + ((xEnd - xStart) / 2), yStart);

g.DrawLine(penL, xStart + ((xEnd - xStart) / 2), yStart, xEnd, yEnd);

}

else if (!RButton)

{

g.DrawLine(penR, xStart, yEnd, xEnd, yEnd);

g.DrawLine(penR, xStart, yEnd, xStart + ((xEnd - xStart) / 2), yStart);

g.DrawLine(penR, xStart + ((xEnd - xStart) / 2), yStart, xEnd, yEnd);

}

ftx[1] = xStart;

fty[1] = yEnd;

ftx[2] = xStart + ((xEnd - xStart) / 2);

fty[2] = yStart;

ftx[3] = xEnd;

fty[3] = yEnd;

break;

// Трикутник прямокутний

case 6:

if (RButton)

{

g.DrawLine(penL, xStart, yStart, xEnd, yEnd);

g.DrawLine(penL, xStart, yEnd, xEnd, yEnd);

g.DrawLine(penL, xStart, yStart, xStart, yEnd);

}

else if (!RButton)

{

g.DrawLine(penR, xStart, yStart, xEnd, yEnd);

g.DrawLine(penR, xStart, yEnd, xEnd, yEnd);

g.DrawLine(penR, xStart, yStart, xStart, yEnd);

}

ftx[1] = xStart;

fty[1] = yStart;

ftx[2] = xStart;

fty[2] = yEnd;

ftx[3] = xEnd;

fty[3] = yEnd;

break;

// Ромб

case 7:

if (RButton)

{

g.DrawLine(penL, xStart + (xEnd - xStart) / 2, yStart, xEnd, yStart + (yEnd - yStart) / 2);

g.DrawLine(penL, xStart + (xEnd - xStart) / 2, yStart, xStart, yStart + (yEnd - yStart) / 2);

g.DrawLine(penL, xStart, yStart + (yEnd - yStart) / 2, xStart + (xEnd - xStart) / 2, yEnd);

g.DrawLine(penL, xStart + (xEnd - xStart) / 2, yEnd, xEnd, yStart + (yEnd - yStart) / 2);

}

else if (!RButton)

{

g.DrawLine(penR, xStart + (xEnd - xStart) / 2, yStart, xEnd, yStart + (yEnd - yStart) / 2);

g.DrawLine(penR, xStart + (xEnd - xStart) / 2, yStart, xStart, yStart + (yEnd - yStart) / 2);

g.DrawLine(penR, xStart, yStart + (yEnd - yStart) / 2, xStart + (xEnd - xStart) / 2, yEnd);

g.DrawLine(penR, xStart + (xEnd - xStart) / 2, yEnd, xEnd, yStart + (yEnd - yStart) / 2);

}

ftx[1] = xStart + (xEnd - xStart) / 2;

fty[1] = yStart;

ftx[2] = xEnd;

fty[2] = yStart + (yEnd - yStart) / 2;

ftx[3] = xStart;

fty[3] = yStart + (yEnd - yStart) / 2;

ftx[4] = xStart + (xEnd - xStart) / 2;

fty[4] = yEnd;

break;

case 101:

penL.Color = Color.White;

g.DrawLine(penL, xStart, yStart, xEnd, yEnd);

xStart = xEnd;

yStart = yEnd;

break;

default:

break;

}

}

penR.Dispose();

penL.Dispose();

e.Graphics.DrawImageUnscaled(bmp, 0, 0);

g.Dispose();

}

}

//Обробка в момент відпускання клавіші

private void picture\_MouseUp(object sender, MouseEventArgs e)

{

Fractal fractal = new Fractal(Convert.ToDouble(fracktalCtextBox.Text) / 1000, this);

drawing = false;

snap = (Bitmap)bmp.Clone();

if (isFractalCheckBox.Checked)

{

switch (pencil)

{

case 4:

fractal.fractal\_rect(buttonCurrentCollor.BackColor, ftx[1], fty[1], ftx[2], fty[2]);

picture.Image = bmp;

snap = bmp;

break;

case 5:

case 6:

fractal.fractal\_tringle(buttonCurrentCollor.BackColor, ftx[1], fty[1], ftx[2], fty[2], ftx[3], fty[3]);

picture.Image = bmp;

snap = bmp;

break;

case 7:

fractal.fractal\_romb(buttonCurrentCollor.BackColor, ftx[1], fty[1], ftx[2], fty[2], ftx[3], fty[3], ftx[4], fty[4]);

picture.Image = bmp;

snap = bmp;

break;

default:

break;

}

}

}

//Обробка натискань та координат під час малювання

private void picture\_MouseMove(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left) RButton = true;

if (e.Button == MouseButtons.Right) RButton = false;

if (drawing)

{

if (pencil != 1 && pencil != 101) bmp = (Bitmap)snap.Clone();

xEnd = e.X;

yEnd = e.Y;

picture.Invalidate();

picture.Update();

}

}

# ДОДАТОК Б

Код класу Fractal

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

//using System.Windows.Forms;

namespace Paint2D

{

class Fractal

{

public double devcof = 0.501;

WorkSpace f1;

public Fractal(double val, WorkSpace f)

{

devcof = val;

f1 = f;

}

// Фрактальний трикутник

public void fractal\_tringle(Color color, int x1, int y1, int x2, int y2, int x3, int y3)

{

Random rand = new Random();

int point;

int xcp = (x1 + x2) / 2;

int ycp = y3;

for (int i = 0; i < 200000; i++)

{

point = rand.Next(0, 4);

try

{

switch (point)

{

case 1:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x1 + (xcp - x1) \* devcof));

ycp = Convert.ToInt32((y1 + (ycp - y1) \* devcof));

break;

case 2:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x2 + (xcp - x2) \* devcof));

ycp = Convert.ToInt32((y2 + (ycp - y2) \* devcof));

break;

case 3:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x3 + (xcp - x3) \* devcof));

ycp = Convert.ToInt32((y3 + (ycp - y3) \* devcof));

break;

default:

break;

}

}

catch (Exception)

{

return;

}

}

//picture.Image = bmp;

}

// Фрактальний прямокутник

public void fractal\_rect(Color color, int x1, int y1, int x2, int y2)

{

Random rand = new Random();

int point;

int xcp = x1;

int ycp = y1;

int x3 = x1;

int y3 = y2;

int x4 = x2;

int y4 = y1;

int x5 = x1 + (x2 - x1) / 2;

int y5 = y1;

int x6 = x1;

int y6 = y1 + (y2 - y1) / 2;

int x7 = x1 + (x2 - x1) / 2; ;

int y7 = y2;

int x8 = x2;

int y8 = y1 + (y2 - y1) / 2;

for (int i = 0; i < 200000; i++)

{

point = rand.Next(0, 9);

try

{

switch (point)

{

case 1:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x1 + (xcp - x1) \* devcof));

ycp = Convert.ToInt32((y1 + (ycp - y1) \* devcof));

break;

case 2:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x2 + (xcp - x2) \* devcof));

ycp = Convert.ToInt32((y2 + (ycp - y2) \* devcof));

break;

case 3:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x3 + (xcp - x3) \* devcof));

ycp = Convert.ToInt32((y3 + (ycp - y3) \* devcof));

break;

case 4:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x4 + (xcp - x4) \* devcof));

ycp = Convert.ToInt32((y4 + (ycp - y4) \* devcof));

break;

case 5:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x5 + (xcp - x5) \* devcof));

ycp = Convert.ToInt32((y5 + (ycp - y5) \* devcof));

break;

case 6:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x6 + (xcp - x6) \* devcof));

ycp = Convert.ToInt32((y6 + (ycp - y6) \* devcof));

break;

case 7:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x7 + (xcp - x7) \* devcof));

ycp = Convert.ToInt32((y7 + (ycp - y7) \* devcof));

break;

case 8:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x8 + (xcp - x8) \* devcof));

ycp = Convert.ToInt32((y8 + (ycp - y8) \* devcof));

break;

default:

break;

}

}

catch (Exception)

{

return;

}

}

//picture.Image = bmp;

}

// Фрактальний ромб

public void fractal\_romb(Color color, int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4)

{

Random rand = new Random();

int point;

int xcp = x1;

int ycp = y1;

int x5 = x1 + (x2 - x1) / 2;

int y5 = y1 + (y2 - y1) / 2;

int x6 = x1 - (x2 - x1) / 2;

int y6 = y1 + (y2 - y1) / 2;

int x7 = x3 + (x4 - x3) / 2;

int y7 = y3 + (y4 - y3) / 2;

int x8 = x4 + (x2 - x4) / 2;

int y8 = y2 + (y4 - y2) / 2;

for (int i = 0; i < 200000; i++)

{

point = rand.Next(0, 9);

try

{

switch (point)

{

case 1:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x1 + (xcp - x1) \* devcof));

ycp = Convert.ToInt32((y1 + (ycp - y1) \* devcof));

break;

case 2:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x2 + (xcp - x2) \* devcof));

ycp = Convert.ToInt32((y2 + (ycp - y2) \* devcof));

break;

case 3:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x3 + (xcp - x3) \* devcof));

ycp = Convert.ToInt32((y3 + (ycp - y3) \* devcof));

break;

case 4:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x4 + (xcp - x4) \* devcof));

ycp = Convert.ToInt32((y4 + (ycp - y4) \* devcof));

break;

case 5:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x5 + (xcp - x5) \* devcof));

ycp = Convert.ToInt32((y5 + (ycp - y5) \* devcof));

break;

case 6:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x6 + (xcp - x6) \* devcof));

ycp = Convert.ToInt32((y6 + (ycp - y6) \* devcof));

break;

case 7:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x7 + (xcp - x7) \* devcof));

ycp = Convert.ToInt32((y7 + (ycp - y7) \* devcof));

break;

case 8:

f1.bmp.SetPixel(xcp, ycp, color);

xcp = Convert.ToInt32((x8 + (xcp - x8) \* devcof));

ycp = Convert.ToInt32((y8 + (ycp - y8) \* devcof));

break;

default:

break;

}

}

catch (Exception)

{

return;

}

}

//picture.Image = bmp;

}

}

}