|  |
| --- |
| [Название организации] |
| Факультет компьютерных наук  Департамент программной инженерии |
| Контрольное домашнее задание по дисциплине «программирование» |
|  |
| Графика. Расчёт площади изображения методом «Монте – Карло» |
| Вариант 14 |
| Выполнил студент группы БПИ154(2)  Мачнев Алексей Евгеньевич |
| Тел: +7(916)-272-32-63  e-mail: [aemachnev@edu.hse.ru](mailto:aemachnev@edu.hse.ru)  Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Оглавление

[1. Условия задачи 2](#_Toc436984923)

[2. Функции разрабатываемого приложения 3](#_Toc436984924)

[2.1 Варианты использования 3](#_Toc436984925)

[2.2 Описание интерфейса пользователя 3](#_Toc436984926)

[3. Структура приложения 6](#_Toc436984927)

[3.1 Диаграмма классов 6](#_Toc436984928)

[3.2 Описание классов, их полей и методов 6](#_Toc436984929)

[4. Распределение кода по файлам проекта 8](#_Toc436984930)

[5.Контрольный пример и описание результатов 8](#_Toc436984931)

[6. Текст программы 10](#_Toc436984932)

[7. Список литературы 25](#_Toc436984933)

# Условия задачи

Вариант 14.

Разработать оконное приложение (шаблон **Windows Forms Application**) для определения относительной площади изображения методом “Монте-Карло”.

В клиентской области элемента **PictureBox** изобразить контуром рисунок (рис. 1), рисунок должен масштабироваться при изменении размеров окна. Требуется определить относительную площадь изображения и раскрасить его.

Случайным образом генерируются координаты точек, которые отображаются на рисунке разными цветами в зависимости от попадания или нет в изображение (различные области предпочтительнее раскрашивать разными цветами). Ведется подсчет общего количества точек и количества точек, попавших в изображение. Относительная площадь изображения определяется как отношение количества попавших в него точек к общему количеству брошенных, умноженное на 100%.

В приложении реализовать два режима:

1. Бесконечный. По нажатии на клавишу **P** и кнопку “**Остановить**” на форме процесс останавливается, выводится результат. По нажатии на другую клавишу C и кнопку “**Продолжить**” на форме процесс продолжается.

2. С таймером. Время выбирать с помощью элементов **ComboBox** или **TrackBar**.

Переключение режимов реализовать с помощью любые управляющих элементов. В обоих режимах процесс генерации точек запускать по нажатии на клавишу **S** или кнопку “**Начать**” на форме. При уже запущенном или приостановленном (в первом режиме) процессе, по нажатии на клавишу **S** или кнопку “Начать” должен происходить возврат изображения к исходному состоянию и перезапуск процесса.

При возникновении исключений и/или некорректных действиях пользователя выводить соответствующее сообщения в диалоговых окнах.

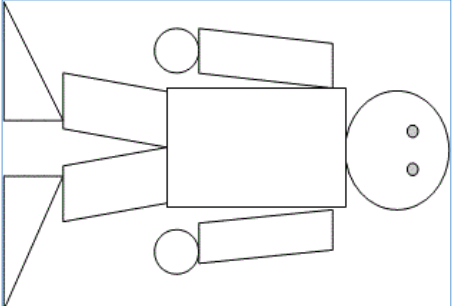


Рис. 1. Рисунок для отображения в программе.

# Функции разрабатываемого приложения

## Варианты использования

Приложение можно использовать в образовательных и ознакомительных целей для иллюстрации работы метода «Монте – Карло».

## Описание интерфейса пользователя

Основные элементы интерфейса изображены на рис. 2.2.1.

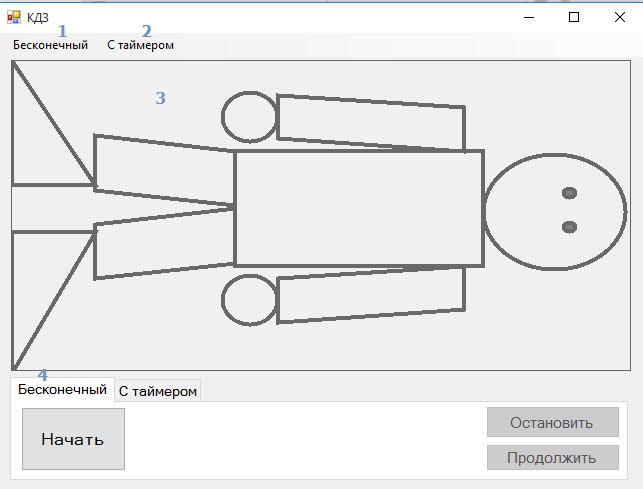


Рис. 2.2.1. Основные элементы интерфейса.

Здесь **1** – пункт меню для бесконечного режима, **2** – пункт меню для режима с таймером, **3** – область отображения изображения и результатов, **4 –** панель инструментов.

**Пункт меню для бесконечного режима (1)** содержит три подпункта:

1. «**Начать**» - запускает метод «Монте – Карло» в бесконечном режиме.
2. «**Остановить**» - приостанавливает работу метода «Монте – Карло»
3. «**Продолжить» -** продолжает работу метода «Монте – Карло»

**Пункт меню для режима с таймером (2)** содержит два подпункта:

1. «**Установить время**» - направляет пользователя на установку времени работы метода «Монте – Карло» в режиме с таймером.
2. «**Начать» -** запускает метод «Монте – Карло» в режиме с таймером

Изображение и результаты работы метода «Монте – Карло» будут отображаться в **области отображения результатов (3).**

**Панель инструментов (4)** состоит из двух вкладок:

1. **«Бесконечный» -** содержит элементы управления для работы метода «Монте – Карло» в **бесконечном** режиме.
2. **«С таймером» -** отображает элементы управления для работы метода «Монте – Карло» в режиме **с таймером**.

Содержимое вкладки (1) представлено на рис. 2.2.2.

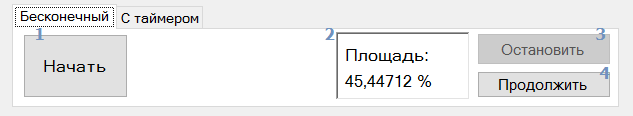


Рис. 2.2.2. Содержимое вкладки **бесконечного** режима.

Здесь **1 –** кнопка «**начать»** бесконечного режима, **2 –** панель отображения площади, **3 –** кнопка «**остановить»**, **4 -**  кнопка «**продолжить».**

Содержимое вкладки (2) представлено на рис. 2.2.3

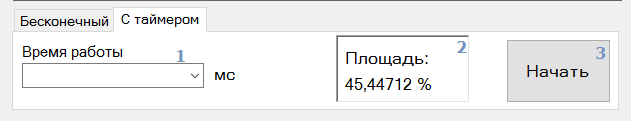


Рис. 2.2.3. Содержимое вкладки режима **с таймером**.

Здесь **1 –** элемент **установки времени выполнения** для режима с таймером, **2 –** панель отображения площади, **3 –** кнопка **«Начать»** режима **с таймером.**

Для запуска метода «Монте – Карло» в **бесконечном** режиме нажмите на кнопку **«Начать»** бесконечного режима или выберите соответствующий пункт меню или нажмите **S,** находясь на вкладке «**Бесконечный»** панели инструментов.

Для запуска метода «Монте – Карло» в режиме **с таймером** нажмите на кнопку **«Начать»** на вкладке панели инструментов «**С таймером»** или выберите соответствующий пункт меню или нажмите **S,** находясь на вкладке «**С таймером»** панели инструментов. Для того, чтобы запустился метод с таймером, необходимо, чтобы в элементе **установки времени** было установлено целое положительное число, не превышающее 2 147483 647. Для того, чтобы установить значение в этот элемент, откройте вкладку «**С таймером»** или выберите в меню пункт «**Установить время»**, после чего введите нужное вам значение, соответствующее вышеописанным правилам или выберите его из списка. Время измеряется в миллисекундах. Метод, запущенный в этом режиме, остановится спустя указанное вами время.

Вы можете сами остановить выполнение, запущенное как в **бесконечном** режиме, так и в режиме **с таймером**. Для остановки метода «Монте – Карло» нажмите кнопку «**Остановить»**,или выберите пункт меню **Бесконечный -> Остановить»**, или нажмите клавишу **P.** Вы можете остановить выполнение, только если оно запущено; если выполнение приостановлено или не начиналось, кнопка и пункт меню «**Остановить»** будут неактивны.

Для продолжения выполнения метода «Монте – Карло» нажмите «**Продолжить»** в меню или на панели задач, или нажмите **C.** Вы можете продолжить только запущенный и остановленный метод. Если метод ещё не был запущен или в данный момент выполняется, кнопка «**Продолжить»** на панели задач и в меню будет неактивна.

Вы можете получить дополнительную информацию о том, сколько точек брошено, сколько из них оказалось в изображении, щелкнув дважды по панели отображения площади на любой вкладке.

Если метод «Монте – Карло» запущен в **бесконечном** режиме, то в **область отображения результатов и на панель отображения площади** периодически будет выводиться текущий результат работы. Период такого обновления зависит от времени, затраченного на отображения текущего результата. В **режиме таймера** результаты выводятся только при остановке выполнения метода или при изменении размера / сворачивании – разворачивании окна.

# 3. Структура приложения

## Диаграмма классов

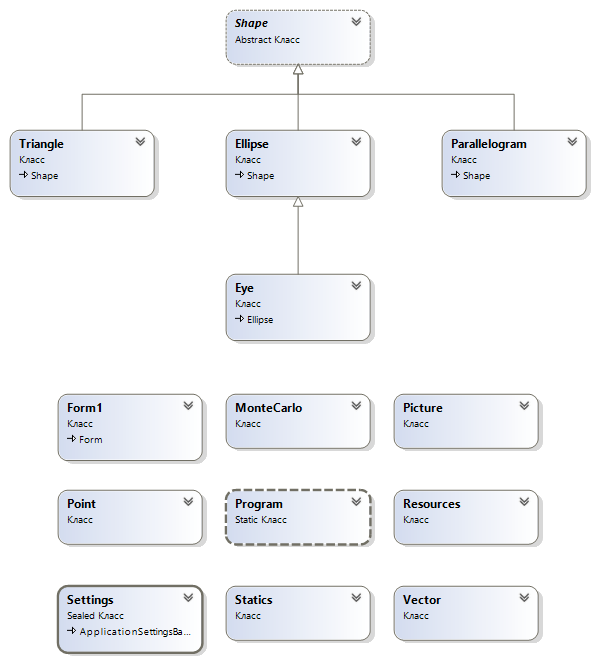


Рис. 3.1. Диаграмма классов

## Описание классов, их полей и методов

**Point –** класс представляет структуру для хранения точек в относительных координатах

* **X, Y - координаты точки**
* **opetator + -** транспонирование точки на вектор **v**
* метод **ToDrawingPoint –** преобразование к типу **System.Drawing.Point** с позиционированием в координатах области рисования

**Vector –** структура для хранения вектора

* **X, Y –** координаты вектора
* **Операторы - - +** операции с векторами

**Shape –** абстрактная фигура.

* метод **Draw –** рисует структуру контуром
* метод **DrawColorful -** рисует структуру контуром и закрашивает цветом **Color**
* метод **IsInside –** проверяет, принадлежит ли точка данной фигуре
* **Color –** Цвет закрашивания

**Statics –** некоторые статические члены для удобства реализации абстрактного класса **Shape** наследниками

* **PEN – System.Drawing.Pen,** которым рисуется контур фигур
* **IsInParallelogramm –** проверяет, принадлежит точка параллелограмму

**Triangle –** треугольник, заданный точкой и двумя векторами

* **M -** точка
* **A -** вектор
* **B -** вектор

**Parallelogram -** треугольник, заданный точкой и двумя векторами

* **M -** точка
* **A -** вектор
* **B -** вектор

**Ellipse –** эллипс, заданный левым верхним, шириной и высотой

* **LC –** левый верхний угол
* **HD -** ширина
* **VD -** высота

**Eye –** переопределенный эллипс для рисования глаз

* **Draw -** переопределили, т.к. незакрашенные глаза должны быть серыми

**Picture –** содержит наш рисунок и методы работы с ним

* метод **init –** строит рисунок из фигур
* метод **Draw** – отрисовывает рисунок
* метод **DrawColorFul –** отрисовывает цветной рисунок(с фигурами, закрашенными каждая в свой цвет)
* метод **Shape –** определяет, какой фигуре принадлежит точка
* поле **shapes –** содержит список фигур рисунка
* свойство **Shapes –** соответствует полю shapes

**MonteCarlo –** содержит логику работы метода «Монте Карло»

* методы **StartAsync, Stop Async –** запускает и, соответственно, останавливает работу метода «Монте Карло»
* метод **createPoints –** генерирует случайные точки
* метод **addPoint –** добавляет точку в **points** или **pointsNotInPicture**
* метод **ResetAsync –** сбрасывает объект метода «Монте Карло»
* метод **All, Inisde, Square –** получают, соответственно, общее количество точек, количество точек внутри фигуры и площадь
* свойство **Picture –** получаетизображение, связанное с данным объектом
* поле **points, pointsNotInPicture –** хранят точки, попавшие в изображение, и, соответственно, не попавшие
* поле **running –** показывает, выполняется ли метод «Монте Карло»
* метод **Draw –** рисует связанное изображение и результат работы
* **rects –** преобразует массив точек в относительных координатах в массив прямоугольников непосредственно для рисования

**Form1 –** форма с графическим интерфейсом пользователя. Содержит обработчики событий и методы для работы с объектом **MonteCarlo** и пользовательским интерфейсом

* **start1, start2 –** запускает выполнение в разных режимах
* **stop –** останавливает выполнение
* **cont –** продолжает выполнение
* **paint** – перерисовывает область рисования
* **show\_sq –** отображает площадь
* **showInfo –** отображает общее количество точек, количество попавших в фигуру точек и площадь в диалоговом окне
* **selectTime –** устанавливает фокус на **элемент установки времени выполнения**

# Распределение кода по файлам проекта

Исходный код проект состоит из папок **Model, Controller,** и частей класса **Form1** в корневом каталоге.

Папка **Model** содержит классы изображения:

* **Model/Shapes/Primitives –** геометрические примитивы
* **Model/Shapes/Shape.cs –** класс **Shape**
* **Model/Shapes/Statics.cs –** класс **Statics**
* **Model/Eye.cs –** класс **Eye**
* **Model/Picture.cs -** класс **Picture**

Папка **Model.Controller** содержит класс **MonteCarlo** в двух файлах:

* **Controller/MonteCarlo.cs –** логика метода «Монте – Карло»
* **Controller/MonteCarlo.Draw.cs –** отрисовка результатов работы

Класс **Form1** разбит на следующие части:

* **Form1.cs –** содержит конструктор и члены, объявленные пользователем
* **Form1.Designer.cs –** содержит сгенерированный код графического интерфейса
* **Form1\_InterfaceCallbacks.cs –** содержит обработчики событий формы
* **Form1\_PrivateMethods.cs –** содержит методы для работы с объектом класса **MonteCarlo** и вывода результатов

# 5.Контрольный пример и описание результатов

Запустим метод «Монте- Карло» в режиме **с таймером** на 5 секунд

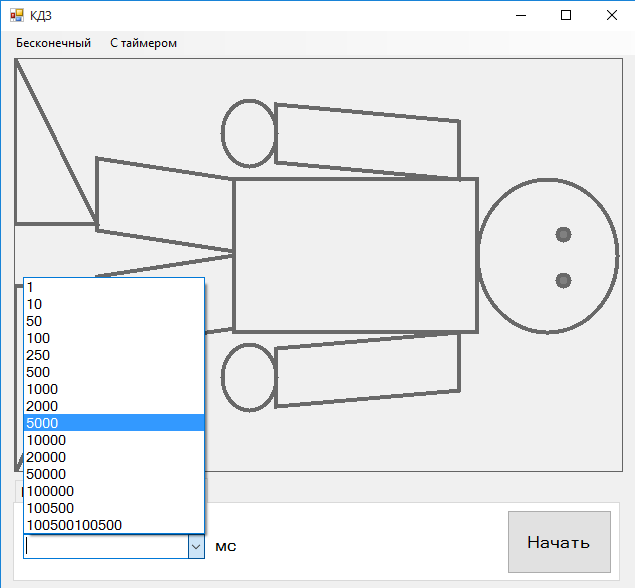


Рис5.1

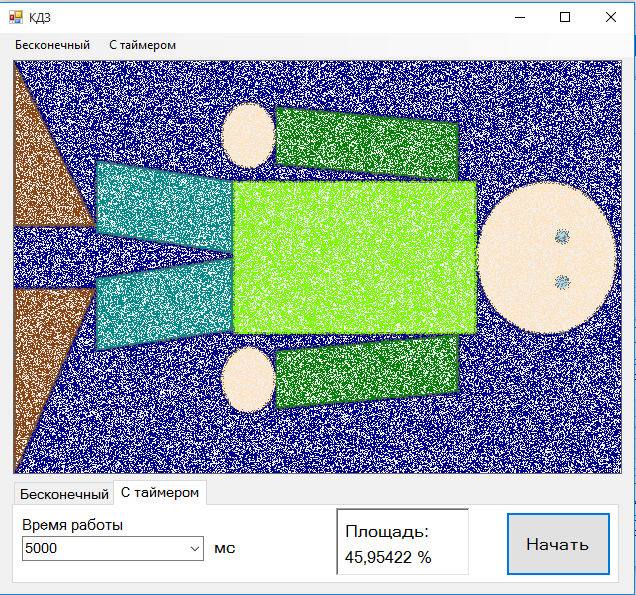


Рис. 5.2. Результат работы

Запустим программу ещё много раз в разных режимах на разное время. Результат не будет отличаться от изображённого здесь более, чем на 1 %. Значит, я смог добиться стабильного результата.

А теперь давайте посмотрим на картинку. Она очень красивая. Это значит, что я, определённо добился своей цели.

# Текст программы

**Model/Shapes/Primitives/Point.cs**

namespace ControlHomeWork1.Model.Shapes.Primitives

{

// Точка в относительных координатах

public class Point

{

public Point() { }

public Point(float x, float y)

{

this.X = x;

this.Y = y;

}

public float X { get; set; }

public float Y { get; set; }

// Хитрое вычисление хэша

// Необходимо для определения одинаковых точек

public override int GetHashCode()

{

return ((int)(X \* (1 << 16))) << 16 + (int)(Y \* (1 << 16));

}

//Перевод в System.Drawing.Point

public System.Drawing.Point ToDrawingPoint(int Width, int Height)

{

return new System.Drawing.Point((int)(X \* Width), (int)(Y \* Height));

}

//Транспонирование на вектор v

public static Point operator +(Point p, Vector v)

{

return new Point(p.X + v.X, p.Y + v.Y);

}

}

}

**Model/Shapes/Primitives/Vector.cs**

namespace ControlHomeWork1.Model.Shapes.Primitives

{

public class Vector

{

public Vector() { }

public Vector(float X, float Y)

{

this.X = X;

this.Y = Y;

}

public Vector(Point p1, Point p2)

{

this.X = p2.X - p1.X;

this.Y = p2.Y - p1.Y;

}

public float X { get; set; }

public float Y { get; set; }

//Сумма векторов v1 и v2

public static Vector operator +(Vector v1, Vector v2)

{

return new Vector(v1.X + v2.X, v1.Y + v2.Y);

}

//Разность векторов v1 и v2

public static Vector operator -(Vector v1, Vector v2)

{

return new Vector(v1.X - v2.X, v1.Y - v2.Y);

}

//Умножение на -1

public static Vector operator -(Vector v)

{

return new Vector(-v.X, -v.Y);

}

}

}

**Model/Shapes/Primitives/Ellipse.cs**

using System;

using System.Drawing;

namespace ControlHomeWork1.Model.Shapes.Primitives

{

public class Ellipse : Shape

{

public virtual Point LC { get; set; } //Left Corner

public virtual float HD { get; set; } //Horizontal diameter

public virtual float VD { get; set; } //Vertical diameter

public Ellipse() { }

public Ellipse(Point LC, float HD, float VD, Color Color)

{

this.LC = LC;

this.HD = HD;

this.VD = VD;

this.Color = Color;

}

//Принадлежность точки еллипсу

public override bool IsInside(Primitives.Point p)

{

float a = Math.Max(HD, VD) / 2f; // большая полуось

float b = Math.Min(HD, VD) / 2f; // малая полуось

//эксцентриситет

float e = (float)Math.Sqrt(Math.Pow(a, 2) - Math.Pow(b, 2)) / a;

// Фокусы в Системе координат с центром, совпадающем с центром еллипса

float f1 = -a \* e;

float f2 = a \* e;

// Приводим к другой системе координат

float \_x = p.X - LC.X - HD / 2;

float \_y = p.Y - LC.Y - VD / 2;

//Важно, была большая полуось вертикальной, или горизонтальной

if (VD > HD)

{

return (Math.Sqrt(Math.Pow(\_x, 2) + Math.Pow(f1 - \_y, 2)) + Math.Sqrt(Math.Pow(\_x, 2) + Math.Pow(f2 - \_y, 2))) <= 2 \* a;

}

else

{

return (Math.Sqrt(Math.Pow(f1 - \_x, 2) + Math.Pow(\_y, 2)) + Math.Sqrt(Math.Pow(f2 - \_x, 2) + Math.Pow(\_y, 2))) <= 2 \* a;

}

}

//Нарисовать контур

public override void Draw(Graphics gr, int width, int height)

{

Rectangle r = new Rectangle((int)(LC.X \* width), (int)(LC.Y \* height), (int)(HD \* width), (int)(VD \* height));

gr.DrawEllipse(Statics.PEN, r);

}

//Нарисовать контур и закрасить цветом Color

public override void DrawColorful(Graphics gr, int width, int height)

{

Rectangle r = new Rectangle((int)(LC.X \* width), (int)(LC.Y \* height), (int)(HD \* width), (int)(VD \* height));

gr.FillEllipse(new SolidBrush(Color), r);

gr.DrawEllipse(Statics.PEN, r);

}

}

}

**Model/Shapes/Primitives/Triangle.cs**

using System.Drawing;

namespace ControlHomeWork1.Model.Shapes.Primitives

{

public class Triangle : Shape

{

//Треугольник образован точкой M и векторами A и B

//точка M

public Point M { get; set; }

//Вектор A

public Vector A { get; set; }

//Вектор B

public Vector B { get; set; }

public Triangle() { }

public Triangle(Point M, Vector A, Vector B)

{

this.M = M;

this.A = A;

this.B = B;

}

//Проверка на принадлежность треугольнику

public override bool IsInside(Point p)

{

//Через два параллелограмма

return Statics.IsInParallelogramm(p, M, A, B) && Statics.IsInParallelogramm(p, M, A, B - A);

}

//Нарисовать контур

public override void Draw(Graphics gr, int width, int height)

{

System.Drawing.Point[] points = { M.ToDrawingPoint(width, height),

(M+A).ToDrawingPoint(width, height),

(M+B).ToDrawingPoint(width, height)};

gr.DrawPolygon(Statics.PEN, points);

}

//Нарисовать контур и закрасить цветом Color

public override void DrawColorful(System.Drawing.Graphics gr, int width, int height)

{

System.Drawing.Point[] points = {M.ToDrawingPoint(width, height),

(M + A).ToDrawingPoint(width, height),

(M + B).ToDrawingPoint(width, height)};

gr.FillPolygon(new System.Drawing.SolidBrush(Color), points);

gr.DrawPolygon(Statics.PEN, points);

}

}

}

**Model/Shapes/Primitives/Parallelogram.cs**

namespace ControlHomeWork1.Model.Shapes.Primitives

{

public class Parallelogram : Shape

{

// Параллелограмм образован точкой M и вокторами A и B

// вектор A

public Vector A { get; set; }

// вектор B

public Vector B { get; set; }

// точка M

public Primitives.Point M { get; set; }

public Parallelogram() { }

public Parallelogram(Point M, Vector A, Vector B)

{

this.M = M;

this.A = A;

this.B = B;

}

// Проверка на принадлежность параллелограмму

public override bool IsInside(Primitives.Point p)

{

return Statics.IsInParallelogramm(p, M, A, B);

}

//Нарисовать контур

public override void Draw(System.Drawing.Graphics gr, int width, int height)

{

System.Drawing.Point[] points = {M.ToDrawingPoint(width, height),

(M + A).ToDrawingPoint(width, height),

(M + A + B).ToDrawingPoint(width, height),

(M + B).ToDrawingPoint(width, height)};

gr.DrawPolygon(Statics.PEN, points);

}

//Нарисовать контур и закрасить цветом Color

public override void DrawColorful(System.Drawing.Graphics gr, int width, int height)

{

System.Drawing.Point[] points = {M.ToDrawingPoint(width, height),

(M + A).ToDrawingPoint(width, height),

(M + A + B).ToDrawingPoint(width, height),

(M + B).ToDrawingPoint(width, height)};

gr.FillPolygon(new System.Drawing.SolidBrush(Color), points);

gr.DrawPolygon(Statics.PEN, points);

}

}

}

**Model/Shapes/Shape.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Drawing;

namespace ControlHomeWork1.Model

{

//Представляет абстрактный класс фигуры

public abstract class Shape

{

public Shape() { }

//Проверяет на принадлежность точки p фигуре

public abstract bool IsInside(Shapes.Primitives.Point p);

//Рисует контур фигуры

public abstract void Draw(Graphics gr, int width, int height);

//Рисует контур фигуры и красит её цветом Color

public abstract void DrawColorful(Graphics gr, int width, int height);

//Цвет для закрашивания фигуры

public virtual Color Color { get; set; }

}

}

**Model/Shapes/Statics.cs**

using System.Drawing;

using ControlHomeWork1.Model.Shapes.Primitives;

namespace ControlHomeWork1.Model.Shapes

{

// Кое - какой повторяющийся код

class Statics

{

// Pen для рисованя контура фигур

public static readonly System.Drawing.Pen PEN = new System.Drawing.Pen(Color.DimGray, 4f);

// Проверяет, принадлежит ли точка p параллелограмму, образованному точкой M и векторами A и B

public static bool IsInParallelogramm(Primitives.Point p, Primitives.Point M, Vector A, Vector B)

{

//Координаты вектора c = Mp

float xc = p.X - M.X;

float yc = p.Y - M.Y;

/\*

\* Решим систему

\*

\* alpha \* xa + beta \* xb = xc

\* alpha \* ya + beta \* yb = yc

\*

\* Где alpha, beta - координаты вектора c = Mp в базисе A, B

\*

\*/

float delta = A.X \* B.Y - A.Y \* B.X;

float delta\_alpha = xc \* B.Y - yc \* B.X;

float delta\_beta = A.X \* yc - A.Y \* xc;

if (delta == 0)

{

return false;

}

float alpha = delta\_alpha / delta;

float beta = delta\_beta / delta;

//Точка принадлежит параллелограмму, если координаты c в базисе A, B меньше или равны 1

return (0 <= alpha && alpha <= 1f) && (0 <= beta && beta <= 1f);

}

}

}

**Model/Eye.cs**

using System.Drawing;

using ControlHomeWork1.Model.Shapes.Primitives;

namespace ControlHomeWork1.Model.Shapes.Parts

{

class Eye : Ellipse

{

public override float HD { get; set; } = 0.02f; //Horizontal diameter

public override float VD { get; set; } = 0.03f; //Vertical diameter

// Незакрашенные глаза должны быть серыми...

public override void Draw(Graphics gr, int width, int height)

{

Brush br = new SolidBrush(Color.Gray);

Rectangle r = new Rectangle((int)(LC.X \* width), (int)(LC.Y \* height), (int)(HD \* width), (int)(VD \* height));

gr.FillEllipse(br, r);

gr.DrawEllipse(Statics.PEN, r);

}

public override Color Color { get; set; } = Color.LightBlue;

}

}

**Model/Picture.cs**

using System.Collections.Generic;

using System.Drawing;

using ControlHomeWork1.Model.Shapes.Parts;

using ControlHomeWork1.Model.Shapes.Primitives;

namespace ControlHomeWork1.Model

{

// Представляет исследуемый рисунок

public class Picture

{

//Список всех фигур

List<Shape> shapes = new List<Shape>();

public Picture()

{

init();

}

public List<Shape> Shapes

{

get { return shapes; }

}

// Рисует контуром данный рисунок

public void Draw(Graphics gr, int width, int height)

{

foreach (Shape shape in shapes)

{

shape.Draw(gr, width, height);

}

}

// Рисует данный рисунок контуром и закрашивает каждую фигуру в соответствующий ей цвет

public void DrawColorfull(Graphics gr, int width, int height)

{

foreach (Shape shape in shapes)

{

shape.DrawColorful(gr, width, height);

}

}

// Определяет, какой фигуре принадлежит точка

public Shape Shape(Shapes.Primitives.Point p)

{

return shapes.FindLast((System.Predicate<Model.Shape>)delegate (Shape sh)

{

return sh.IsInside(p);

});

}

// Строит рисунок из фигур

private void init()

{

//Left Foot

Triangle lf = new Triangle();

lf.M = new Shapes.Primitives.Point(0f, 0f);

lf.A = new Vector(0f, 0.4f);

lf.B = new Vector(0.135f, 0.4f);

lf.Color = Color.SaddleBrown;

//RightFoot

Triangle rf = new Triangle();

rf.M = new Shapes.Primitives.Point(0f, 1.0f);

rf.A = new Vector(0f, -0.45f);

rf.B = new Vector(0.135f, -0.45f);

rf.Color = Color.SaddleBrown;

//Left Leg

Parallelogram ll = new Parallelogram();

ll.M = new Shapes.Primitives.Point(0.135f, 0.7f);

ll.A = new Vector(0.225f, -0.05f);

ll.B = new Vector(0f, -0.175f);

ll.Color = Color.DarkCyan;

//Right Leg

Parallelogram rl = new Parallelogram();

rl.M = new Shapes.Primitives.Point(0.135f, 0.24f);

rl.A = new Vector(0.225f, 0.05f);

rl.B = new Vector(0f, 0.175f);

rl.Color = Color.DarkCyan;

//Left Hand

Ellipse lh = new Ellipse();

lh.LC = new Shapes.Primitives.Point(0.34f, 0.69f); //Левый верхний угол

lh.HD = 0.09f; //Horizontal diameter

lh.VD = 0.16f; //Vertical diameter

lh.Color = Color.Bisque;

//Left Arm

Parallelogram la = new Parallelogram();

la.M = new Shapes.Primitives.Point(0.43f, 0.84f);

la.A = new Vector(0.3f, -0.04f);

la.B = new Vector(0f, -0.14f);

la.Color = Color.Green;

//Right Arm

Parallelogram ra = new Parallelogram();

ra.M = new Shapes.Primitives.Point(0.43f, 0.11f);

ra.A = new Vector(0.3f, 0.04f);

ra.B = new Vector(0f, 0.14f);

ra.Color = Color.Green;

//Right Hand

Ellipse rh = new Ellipse();

rh.LC = new Shapes.Primitives.Point(0.34f, 0.1f);

rh.HD = 0.09f; //Horizontal diameter

rh.VD = 0.16f; //Vertical diameter

rh.Color = Color.Bisque;

//Body

// Параллелограмм образован точкой M и вокторами A и B

Parallelogram b = new Parallelogram();

b.M = new Shapes.Primitives.Point(0.36f, 0.29f);

b.A = new Vector(0.4f, 0f);

b.B = new Vector(0f, 0.37f);

b.Color = Color.LawnGreen;

//Head

Ellipse h = new Ellipse();

h.LC = new Shapes.Primitives.Point(0.76f, 0.29f);

h.HD = 0.23f; //Horizontal diameter

h.VD = 0.37f; //Vertical diameter

h.Color = Color.Bisque;

/\*

\* Все остальные параметры переопределены в классе Eye

\*/

//Left Eye

Eye le = new Eye();

le.LC = new Shapes.Primitives.Point(0.89f, 0.41f);

//Right Eye

Eye re = new Eye();

re.LC = new Shapes.Primitives.Point(0.89f, 0.52f);

shapes.Add(lf);

shapes.Add(rf);

shapes.Add(ll);

shapes.Add(rl);

shapes.Add(lh);

shapes.Add(la);

shapes.Add(b);

shapes.Add(ra);

shapes.Add(rh);

shapes.Add(h);

shapes.Add(le);

shapes.Add(re);

}

}

}

**Controller/MonteCarlo.cs**

using System;

using System.Collections;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

using ControlHomeWork1.Model;

using ControlHomeWork1.Model.Shapes.Primitives;

using ControlHomeWork1.Model.Shapes.Parts;

namespace ControlHomeWork1.Controller

{

/\* Хранит методы для вычисления площади данного изображения методом Монте - Карло,

\* а также точки, испульзукмые для вычисления этим методом

\* и методы для отрисовки этих точек.

\*/

partial class MonteCarlo

{

// Хранит точки, попавшие в изображение

// HashSet используется потому, что в него не добавляется объект, если в нём уже имеется

// другой объект с таким же хэш - кодом.

private Dictionary<Shape, HashSet<Point>> points = new Dictionary<Shape, HashSet<Point>>();

// Хранит точки, не попавшие в рисунок

// HashSet используется потому, что в него не добавляется объект, если в нём уже имеется

// другой объект с таким же хэш - кодом.

private HashSet<Point> pointNotInPicture = new HashSet<Point>();

// Показывает, выполняется ли метод

private Boolean running = false;

// Хранит данное изображение

public Picture Picture { get; set; }

// pict - объект данного изображения

public MonteCarlo(Picture pict)

{

this.Picture = pict;

ResetAsync();

}

// Запускает выполнение метода, не блокируя вызывающий поток.

// При вызове используйте await.

public async Task StartAsync()

{

lock (this)

{

running = true;

}

await createPoints();

}

// Останавливает выполнение метода, не блокируя вызывающий поток.

// При вызове используйте await.

public async Task StopAsync()

{

lock (this)

{

running = false;

}

}

// Сбрасывает хранящиеся значения сгенерированных точек, не блокируя вызывающий поток.

// При вызове используйте await.

public async Task ResetAsync()

{

lock (this)

{

points.Clear();

pointNotInPicture.Clear();

foreach (Shape sh in Picture.Shapes)

{

points.Add(sh, new HashSet<Point>());

}

}

}

// Возвращает количество точек внутри данного изображения.

public int Inside()

{

int inside = 0;

foreach (HashSet<Point> set in points.Values)

{

inside += set.Count;

}

return inside;

}

// Возвращает общее количество сгенерированных точек.

public int All()

{

return Inside() + pointNotInPicture.Count;

}

// Возвращает площадь данного изображения в процентах.

public float Square()

{

return 100 \* (float)Inside() / All();

}

// Генерирует точки для выполнения метода.

private async Task createPoints()

{

await Task.Run(delegate

{

Random rand = new Random();

while (running)

{

lock (this)

{

float x = (float)rand.NextDouble();

float y = (float)rand.NextDouble();

Point p = new Point(x, y);

addPoint(p);

}

}

});

}

// Добавляет точки в соответсвующий массив.

private void addPoint(Point p)

{

Shape sh = Picture.Shape(p);

if (sh != null)

{

// Если точка принадлежит какой - нибудь фигуре данного изабражения,

// добавим её в массив, соответствующий этой фигуре.

points[sh].Add(p);

}

else

{

// Если точка не принадлежит какой - нибудь фигуре данного изабражения,

// добавим её в массив точек, не принадлежащих изображению.

pointNotInPicture.Add(p);

}

}

}

}

**Controller/MonteCarlo.Draw.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using ControlHomeWork1.Model;

using System.Drawing;

namespace ControlHomeWork1.Controller

{

// Содержит метод отрисовки результатов работы метода Монте - Карло.

partial class MonteCarlo

{

// Отрисовывает данное изображение и сгенерированные точки.

public void Draw(Graphics gr, int width, int height)

{

// Сначала рисуем данное изображение.

Picture.Draw(gr, width, height);

// Рисуем точки, попавшие в данное изображение.

foreach (KeyValuePair<Shape, HashSet<ControlHomeWork1.Model.Shapes.Primitives.Point>> entry in points)

{

if (entry.Value.Count > 0)

{

gr.FillRectangles(new SolidBrush(entry.Key.Color), rects(entry.Value, width, height));

}

}

// Рисуем точки, не попавшие в данное изображение.

if (pointNotInPicture.Count > 0)

{

gr.FillRectangles(new SolidBrush(Color.DarkBlue), rects(pointNotInPicture, width, height));

}

}

// Создает массив прямоугольников для рисования в System.Drawing.Graphics

// из массива точек, представленных в относительных координатах.

private Rectangle[] rects(IEnumerable<Model.Shapes.Primitives.Point> points, int width, int height)

{

// Set используется потому, что в него не будут добавлятся точки с одинаковыми координатами,

// а, значит, впоследствии не будут обрабатываться несколько раз.

HashSet<Point> dr\_points = new HashSet<Point>();

foreach(Model.Shapes.Primitives.Point p in points)

{

dr\_points.Add(p.ToDrawingPoint(width, height));

}

Func<Point, Rectangle> selector =

delegate (Point p)

{

return new Rectangle(p.X, p.Y, 1, 1);

};

return dr\_points.Select(selector).ToArray();

}

}

}

**Form1.cs**

using System.Windows.Forms;

using ControlHomeWork1.Controller;

namespace ControlHomeWork1

{

public partial class Form1 : Form

{

// Экземпляр метода Монте - Карло

private MonteCarlo monteCarlo = new MonteCarlo(new Model.Picture());

// Таймер для периодической перерисовки результатов работы метода Монте - Карло

private Timer paintTimer = new Timer();

// Измеряет время перерисовки;

// Используется для более эффективного использования времени

// и для задания интервала таймера paintTimer

private System.Diagnostics.Stopwatch draw\_watch = new System.Diagnostics.Stopwatch();

// Показывает режим отображения изображения:

// рисовать результаты работы метода Монте - Карло

// или раскрашенного изображения

private bool drawColorfull = false;

public Form1()

{

InitializeComponent();

// Запуск таймера

paintTimer.Interval = 100;

paintTimer.Tick += PaintTimer\_Tick;

}

}

}

**Form1\_InterfaceCallbacks.cs**

using System.Windows.Forms;

namespace ControlHomeWork1

{

public partial class Form1

{

// Событие отрисовки pictureBox1

private async void pictureBox1\_Paint(object sender, PaintEventArgs e)

{

await paint(e.Graphics, pictureBox1.Width, pictureBox1.Height);

}

// Событие изменения размеров pictureBox1

private void pictureBox1\_Resize(object sender, System.EventArgs e)

{

pictureBox1.Invalidate();

}

// Событие нажатия кнопки Начать для бесконечного режима

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

{

start1();

}

// Событие нажатия кнопки Остановить

private void button2\_Click(object sender, System.EventArgs e)

{

stop();

}

// Событие нажатия кнопки Продолжить

private void button3\_Click(object sender, System.EventArgs e)

{

cont();

}

// Событие нажатия кнопки Начать для режима с таймером

private void button4\_Click(object sender, System.EventArgs e)

{

start2();

}

// Событие двойного щелчка мыши по pictureBox1

private void pictureBox1\_DoubleClick(object sender, System.EventArgs e)

{

drawColorfull = !drawColorfull;

pictureBox1.Invalidate();

}

// Событие выбора пункта меню "Начать" для бесконечного режима

private void start1ToolStripMenuItem\_Click(object sender, System.EventArgs e)

{

start1();

}

// Событие выбора пункта меню "Остановить" для бесконечного режима

private void stopToolStripMenuItem\_Click(object sender, System.EventArgs e)

{

stop();

}

// Событие выбора пункта меню "Продолжить" для бесконечного режима

private void contToolStripMenuItem\_Click(object sender, System.EventArgs e)

{

cont();

}

// Событие выбора пункта меню "Установить время" для режима с таймером

private void setTimeToolStripMenuItem\_Click(object sender, System.EventArgs e)

{

selectTime();

}

// Событие выбора пункта меню "Начать" для режима с таймером

private void start2ToolStripMenuItem\_Click(object sender, System.EventArgs e)

{

start2();

}

// на случай, если пользователь выберет не то

private void comboBox1\_SelectedIndexChanged(object sender, System.EventArgs e)

{

if (comboBox1.Text == "100500100500")

{

MessageBox.Show("Ха - Ха", "Это шутка!");

}

}

// Событие двойного щелчка по панели отображения результатов во вкладке бесконечного режима

private void panel1\_MouseDoubleClick(object sender, MouseEventArgs e)

{

showInfo();

}

// Событие двойного щелчка по панели отображения результатов во вкладке режима с таймером

private void panel2\_MouseDoubleClick(object sender, MouseEventArgs e)

{

showInfo();

}

// Событие нажатия клавиши на форме

protected override void OnKeyPress(KeyPressEventArgs e)

{

switch (e.KeyChar)

{

case ('S'):

switch (tabControl1.SelectedIndex)

{

case (0):

start1();

break;

case (1):

start2();

break;

default:

break;

}

break;

case ('s'):

switch (tabControl1.SelectedIndex)

{

case (0):

start1();

break;

case (1):

start2();

break;

default:

break;

}

break;

case ('P'):

if (button2.Enabled)

{

stop();

}

break;

case ('p'):

if (button2.Enabled)

{

stop();

}

break;

case ('C'):

if (button3.Enabled)

{

cont();

}

break;

case ('c'):

if (button3.Enabled)

{

cont();

}

break;

default:

break;

}

}

}

}

**Form1\_PrivateMethods.cs**

using System;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ControlHomeWork1

{

public partial class Form1

{

// Отрисовка данного изображения и результатов работы метода Монте - Карло на pictureBox1.

private async Task paint(System.Drawing.Graphics gr, int width, int height)

{

try

{

if (!drawColorfull)

{

// Измеряем время рисования результатов работы метода Монте - Карло

draw\_watch.Restart();

lock (monteCarlo)

{

monteCarlo.Draw(gr, width, height);

show\_sq();

}

draw\_watch.Stop();

}

else

{

gr.Clear(System.Drawing.Color.DarkBlue);

monteCarlo.Picture.DrawColorfull(gr, width, height);

show\_sq();

}

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Отображает площадь на панели отображения площади

private void show\_sq()

{

try

{

float s = monteCarlo.Square();

panel1.Visible = panel2.Visible = !float.IsNaN(s) && !float.IsInfinity(s);

s1\_label.Text = string.Format("{0} %", s);

s2\_label.Text = string.Format("{0} %", s);

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Отображает более подробную информацию о результатах работы метода Монте - Карло в диалоговом окне.

private void showInfo()

{

try

{

MessageBox.Show(string.Format("Всего вброшено: {0}\n" +

"Попало в фигуру: {1}\n" +

"Площадь: {2} %", monteCarlo.All(), monteCarlo.Inside(), monteCarlo.Square()),

"Информация");

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Направляет фокус на элемент выбора времени.

private void selectTime()

{

try

{

tabControl1.SelectTab(1);

comboBox1.Select();

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Запускает бесконечный режим работы метода Монте - Карло и отображает результаты при остановке.

private async void start1()

{

try

{

button2.Enabled = true;

stopToolStripMenuItem.Enabled = true;

button3.Enabled = false;

contToolStripMenuItem.Enabled = false;

await monteCarlo.ResetAsync();

draw\_watch.Reset();

pictureBox1.Refresh();

paintTimer.Start();

await monteCarlo.StartAsync();

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Останавливает работу метода Монте - Карло

private async void stop()

{

try

{

paintTimer.Stop();

await monteCarlo.StopAsync();

pictureBox1.Refresh();

button2.Enabled = false;

stopToolStripMenuItem.Enabled = false;

button3.Enabled = true;

contToolStripMenuItem.Enabled = true;

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Продолжает работу метода Монте - Карло в бесконечном режиме

private async void cont()

{

try

{

button2.Enabled = true;

stopToolStripMenuItem.Enabled = true;

button3.Enabled = false;

contToolStripMenuItem.Enabled = false;

paintTimer.Start();

await monteCarlo.StartAsync();

pictureBox1.Refresh();

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Запускает работу метода Монте - Карло в режиме с таймером

// и отображает результаты по завершении / при остановке

private async void start2()

{

try

{

paintTimer.Stop();

button2.Enabled = true;

stopToolStripMenuItem.Enabled = true;

button3.Enabled = false;

contToolStripMenuItem.Enabled = false;

await monteCarlo.StopAsync();

await monteCarlo.ResetAsync();

int val = 0;

if (!int.TryParse(comboBox1.Text, out val))

{

MessageBox.Show("Введённое время должно быть целым положительным числом", "Ошибка");

selectTime();

return;

}

if (val <= 0)

{

MessageBox.Show("Введённое время должно быть целым положительным числом", "Ошибка");

selectTime();

return;

}

pictureBox1.Refresh();

Timer timer = new Timer();

timer.Enabled = false;

timer.Interval = val;

timer.Tick += Timer\_Tick;

timer.Start();

await monteCarlo.StartAsync();

button2.Enabled = false;

stopToolStripMenuItem.Enabled = false;

button3.Enabled = true;

contToolStripMenuItem.Enabled = true;

}

catch (Exception e)

{

MessageBox.Show(e.Message, "Ошибка");

}

}

// Событие срабатывания таймера перерисовки

private void PaintTimer\_Tick(object sender, System.EventArgs e)

{

try

{

pictureBox1.Invalidate();

paintTimer.Interval = (int)(draw\_watch.ElapsedMilliseconds \* 2) + 10;

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка");

}

}

// Событие срабатывания таймера для режима с таймером

private async void Timer\_Tick(object sender, System.EventArgs e)

{

try

{

((Timer)sender).Stop();

await monteCarlo.StopAsync();

pictureBox1.Refresh();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Ошибка");

}

}

}

}

# Список литературы

* https://msdn.microsoft.com/ru-ru/default.aspx?f=255&MSPPError=-2147217396