Московский авиационный институт

(национальный исследовательский университет)

**Факультет компьютерных наук и прикладной математики**

Кафедра вычислительной математики и программирования

**Курсовая работа по курсу**

**«Объектно-ориентированное программирование»**

**Тема: «Симуляция движения волны по кругу и отображение на сферу»**

№**1**. Обработка исключений,  выбор типа атрибута, конвертация, конкретизация переменных (var)/

№**2**. Паттерны MVC, Delegate Pattern, PictureBox, Компонента MessageBox

№3. Паттерны Abstract Factory и Singleton, Buttons, CheckBox, Label, Track, масштабирование Шрифта RichTextBox

№4. Паттерн Chain-responsibility, Command: встроить паттерн в Window Form

№5. Menu: MenuStrip, ToolStrip, ContextMenuStrip, ToolStripContainer and StatusStrip,  Dialog Boxes: ColorDialog, FolderBrowserDialog, FontDialog, OpenFileDialog, SaveFileDialog,  ReachTextBox,  ClipBoard

№**6**. Tree View, List View, Drag and Drop, Save and Load Image

№7. Lines, Curves and Area Fills. Кривые Безье для авионики, Обработка прерываний от таймера. Вращение  Images and Bitmaps,

№8. Компонента .dll, Типы проектов .Net.

Работу выполнил:

М8О-204Б-21 Рябов С. М.  \_\_\_\_\_\_\_\_

Группа             ФИО:           Подпись     Вариант

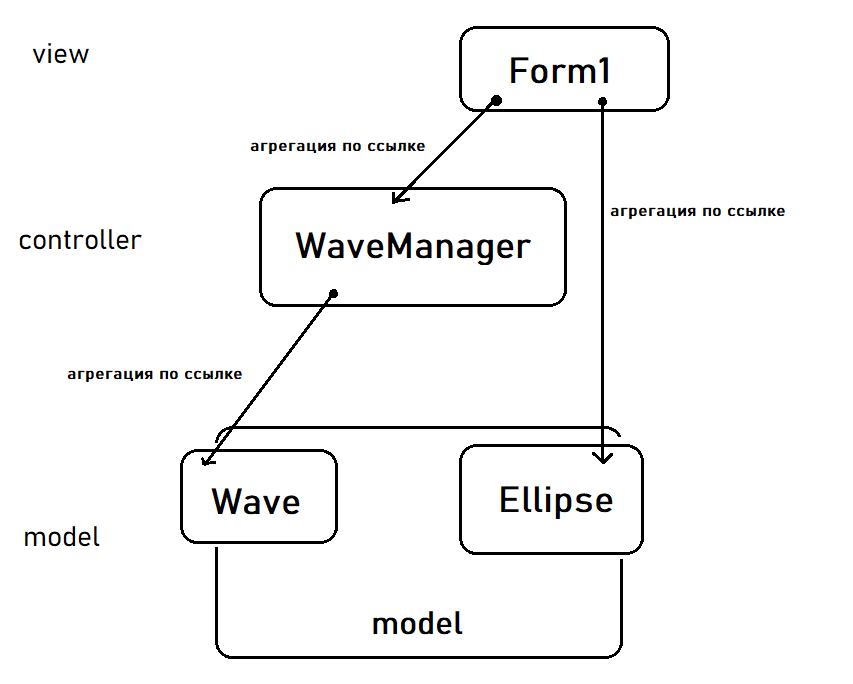
Руководитель: \_\_\_\_\_\_\_\_\_\_/Семенов А.С./

            Подпись:

         Дата: \_\_\_\_\_\_\_\_\_\_ 2023

Задание:

Написать программу на языке c# с применением паттернов ООП и ИСРППС.

Диаграмма классов:  Словесное описание:

Form1 – класс формы, где реализуются все возможные действия в пользовательском интерфейсе, также отсюда передаются запросы.  
WaveManager – класс, который контролирует все необходимые данных, проводит вычисления и возврщает информацию для визуализации.  
Wave – класс, необходимы для хранения состояние волн.  
Ellipse – класс, хранящий в себе методы визулизации сферы.  
Model – пространство имен, хранящее в себе классы.

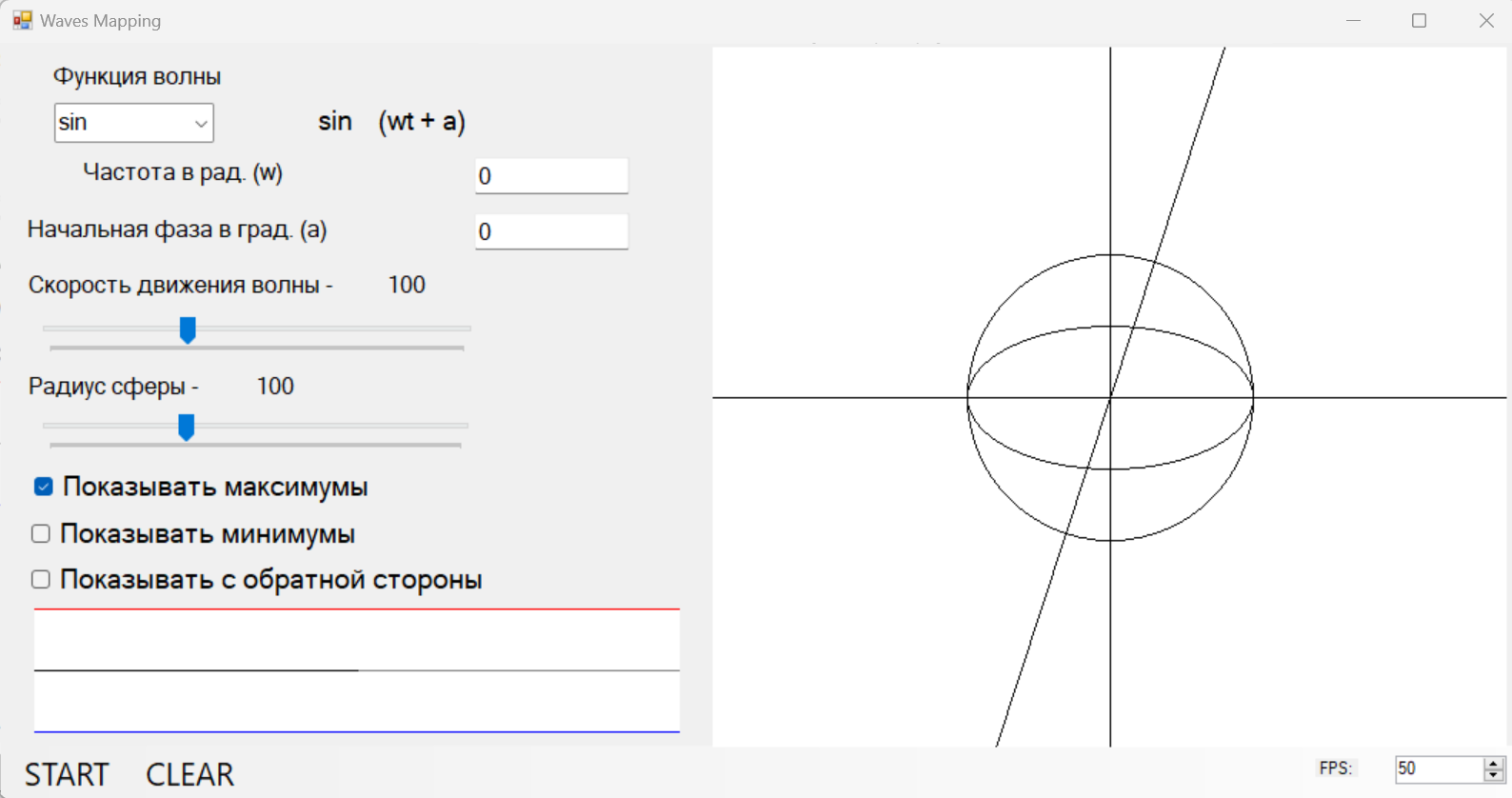
Проект написан с использованием принципа подстановки, замещения и паттернов **MVC** и **Singlton (WaveManager)**.

**Работа программы**

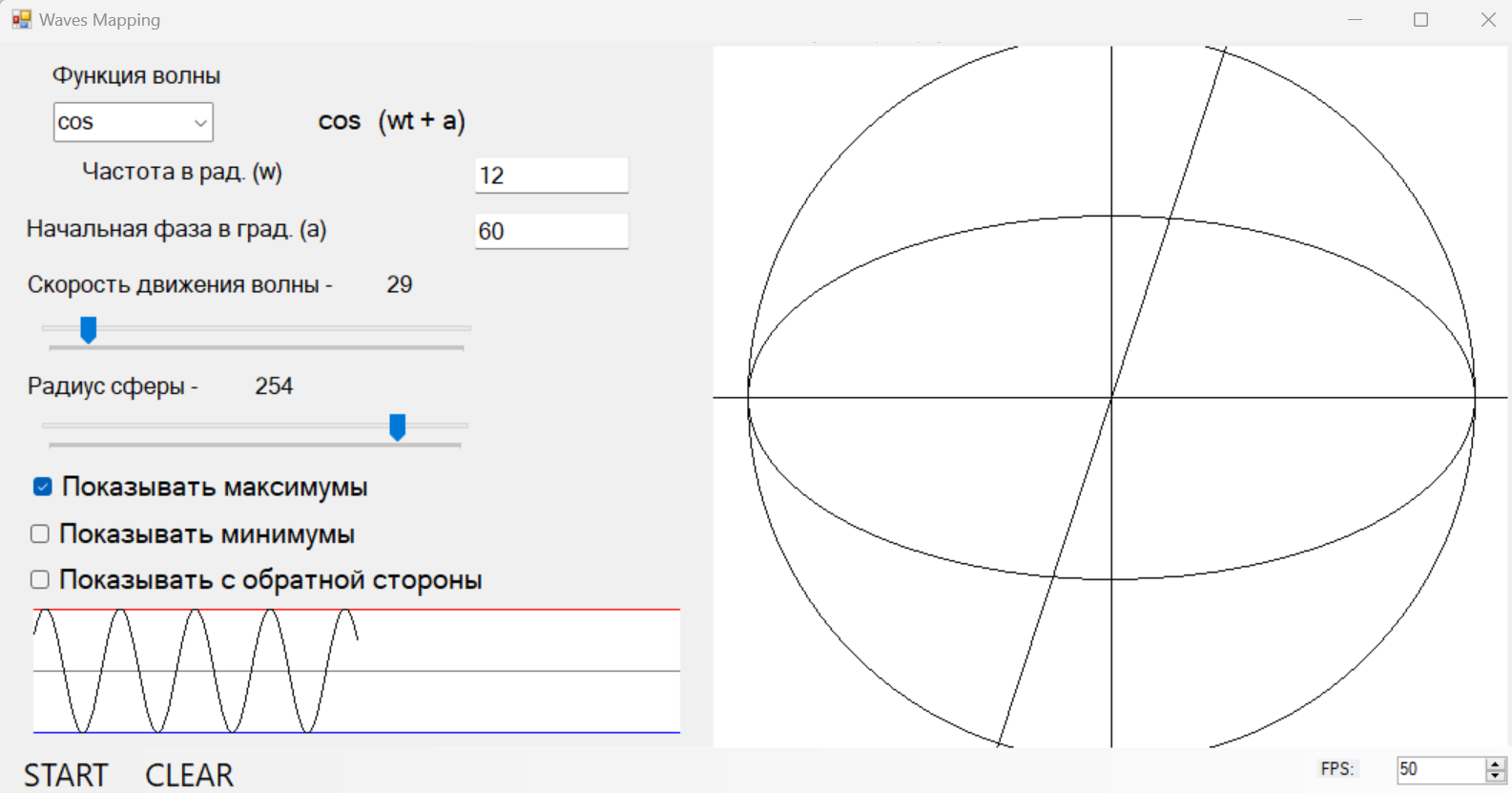
Для симуляции волны, необходимо в начале выбрать саму функцию (sin, cos), настроить частоту и начальную фазу волны, а также скорость волны непосредственно на сфере. Выбрать радиус сферы и поставить или убрать галочки для визуализации тех или иных положений волны. Слева снизу будет кнопка START, при нажатии на которую при успешном вводе данных начнется визуализация движения волны по сфере. При необходимости можно поставить на паузу. Если требуется изменить настройки волны или шара, то можно нажать кнопку CLEAR, которая очищает изображение от волн и позволяет снова настроить параметры. Как бонус, можно справа снизу настроить частоту кадров для программы.

**Скриншоты работы программы**

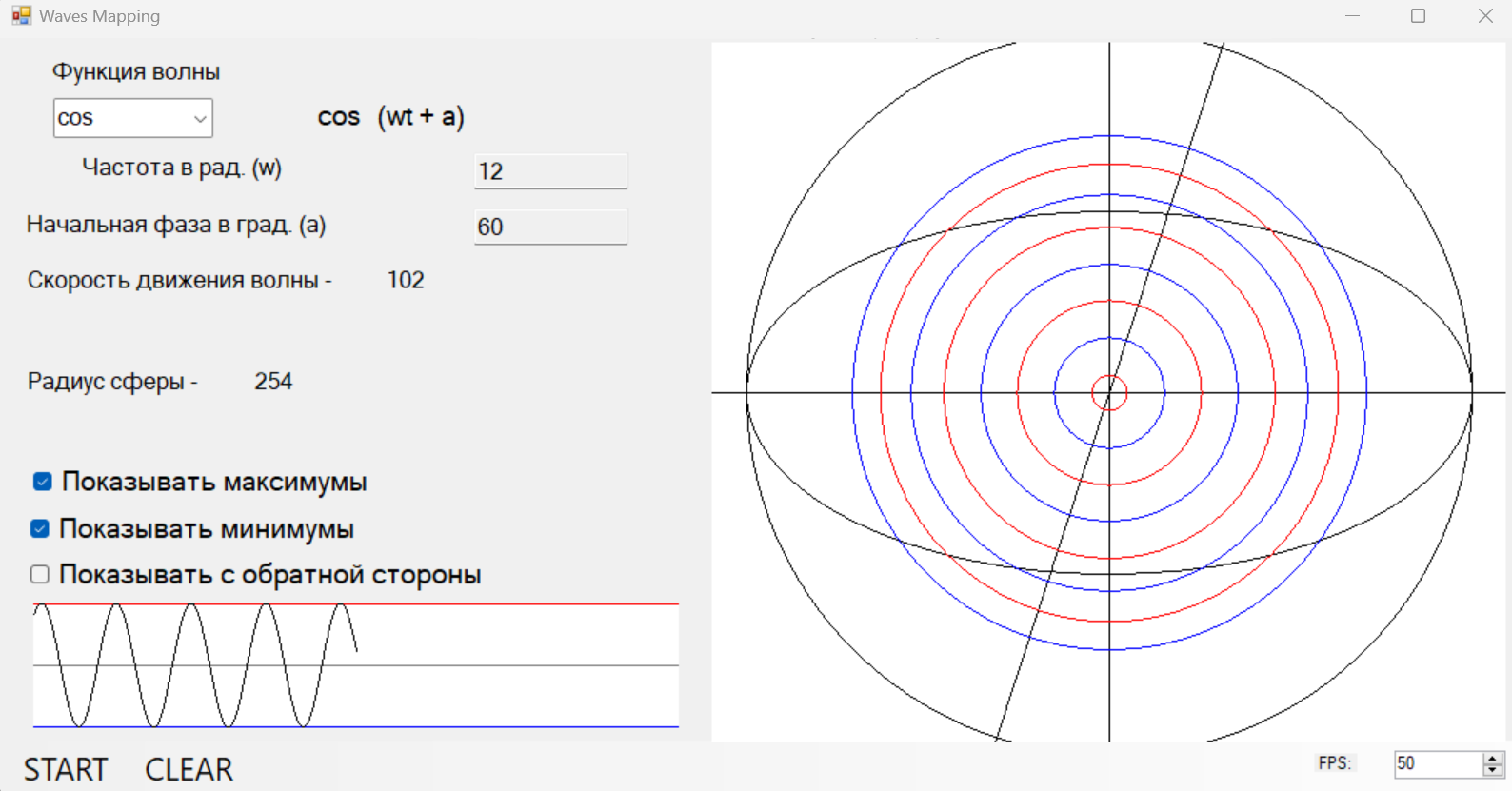
Окно формы



Изменение параметров



Работа программы



Алгоритм преобразования экстремумов функции в визуализацию движения по сфере:

1. Смотрим значение производной функции в данный момент и сравниваем со значением производной на предыдущем шаге (tick’е).
2. Если знак производной меняется с минуса на плюс или с плюса на минус, то добавляем в список волн на сфере новую волну минимума или максимума соответственно.

Пример кода на C#

if (Math.Sign(derivativeWas) != Math.Sign(derivativeNow))

{

if (Math.Sign(derivativeWas) < 0 || Math.Sign(derivativeNow) > 0)

waves.Add(new Wave(false));

else if (Math.Sign(derivativeWas) > 0 || Math.Sign(derivativeNow) < 0)

waves.Add(new Wave(true));

}

derivativeWas – значение производной на предыдущем шаге

derivativeNow – значение производной сейчас

waves – список волн на сфере

new Wave(true) – новый максимум, new Wave(false) – новый минимум

Math.Sign() – метод, определяющий знак числового значения (-1, 0, 1)

**Исходный код**

***Program.cs***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using WindowsFormsApp1.Model;

using System.Windows.Forms;

namespace WindowsFormsApp1

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

***Form1.cs***

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using WindowsFormsApp1.Model;

using System.Windows.Forms;

using System.Diagnostics;

using System.Reflection;

namespace WindowsFormsApp1

{

public partial class Form1 : Form

{

Graphics g;

Bitmap bitmap;

public Ellipse ellipse;

WaveManager waveManager = WaveManager.GetInstance();

uint radius = 100;

uint waveSpeed = 100;

public Form1()

{

InitializeComponent();

this.ellipse = new Model.Ellipse(this.pictureBoxImage);

bitmap = new Bitmap(this.pictureBoxImage.Width, this.pictureBoxImage.Height);

this.g = Graphics.FromImage(bitmap);

pictureBoxImage = ellipse.Draw\_Ellipse(pictureBoxImage, radius);

timer1.Interval = 1000 / 50; // 50 кадров в секунду

}

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

{

}

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

{

double freq;

double phase;

if (Double.TryParse(textBoxFreq.Text.Replace('.', ','), out freq) &&

Double.TryParse(textBoxPhase.Text.Replace('.', ','), out phase))

{

if (comboBoxFunc.Text == "cos")

phase += 90;

waveManager.Update(timer1.Interval, waveSpeed, radius, (float)freq,

((float)phase \* 2 \* (float)Math.PI / 360f) % 720);

if (!timer1.Enabled)

{

timer1.Start();

CloseUI();

sTARTToolStripMenuItem.Text = "PAUSE";

}

else

{

timer1.Stop();

sTARTToolStripMenuItem.Text = "START";

}

}

else

{

if (!Double.TryParse(textBoxFreq.Text, out freq))

textBoxFreq.Text = "ERROR";

if (!Double.TryParse(textBoxPhase.Text, out phase))

textBoxPhase.Text = "ERROR";

}

}

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

{

waveManager.CheckExtremums();

waveManager.CalculateWaves();

RefreshImages();

if (waveManager.waves.Count != 0 || waveManager.GetFuncTime() != 0)

{

CloseUI();

}

else

{

OpenUI();

waveManager.Reset();

}

}

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

{

foreach (var wave in waveManager.waves)

{

var currentHeight = (float)Math.Sin(wave.Position) \* radius \* 2;

Pen pen;

if (!checkBoxMin.Checked && !wave.IsMax())

continue;

if (!checkBoxMax.Checked && wave.IsMax())

continue;

if (wave.Position < Math.PI / 2)

{

if (wave.IsMax())

{

pen = Pens.Red;

}

else

{

pen = Pens.Blue;

}

}

else

{

if (!checkBoxSeeBack.Checked)

continue;

if (wave.IsMax())

pen = Pens.Green;

else

pen = Pens.Gray;

}

e.Graphics.DrawEllipse(pen, pictureBoxImage.Width / 2 - currentHeight / 2, pictureBoxImage.Height / 2 - currentHeight / 2, currentHeight, currentHeight);

}

}

private void trackBar1\_Scroll(object sender, EventArgs e)

{

radius = (uint)trackBar1.Value;

try { labelRadius.Text = Convert.ToString(radius); }

catch (FormatException) { Environment.Exit(0); }

this.pictureBoxImage = ellipse.Draw\_Ellipse(pictureBoxImage, radius);

pictureBoxImage.Refresh();

}

private void numericUpDown3\_ValueChanged(object sender, EventArgs e)

{

timer1.Interval = 1000 / (int)numericUpDown3.Value;

waveManager.Update(timer1.Interval, waveSpeed, radius);

}

private void comboBoxFunc\_SelectedIndexChanged(object sender, EventArgs e)

{

label8.Text = comboBoxFunc.Text;

textBoxWave\_TextChanged(sender, e);

}

private void trackBar2\_Scroll(object sender, EventArgs e)

{

waveSpeed = (uint)trackBar2.Value;

try { labelSpeed.Text = Convert.ToString(waveSpeed); }

catch (FormatException) { Environment.Exit(0); }

}

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

{

waveManager.waves.Clear();

waveManager.Reset();

if (timer1.Enabled)

{

timer1.Stop();

sTARTToolStripMenuItem.Text = "START";

}

OpenUI();

RefreshImages();

}

void CloseUI()

{

trackBar1.Hide();

trackBar2.Hide();

textBoxFreq.ReadOnly = true;

textBoxPhase.ReadOnly = true;

//numericUpDown3.Enabled = false;

/\*checkBoxMax.Enabled = false;

checkBoxMin.Enabled = false;

checkBoxSeeBack.Enabled = false;\*/

}

void OpenUI()

{

trackBar1.Show();

trackBar2.Show();

textBoxFreq.ReadOnly = false;

textBoxPhase.ReadOnly = false;

//numericUpDown3.Enabled = true;

/\*checkBoxMax.Enabled = true;

checkBoxMin.Enabled = true;

checkBoxSeeBack.Enabled = true;\*/

}

void RefreshImages()

{

pictureBoxImage.Refresh();

pictureBoxFunc.Refresh();

}

private void checkBoxMax\_CheckedChanged(object sender, EventArgs e)

{

RefreshImages();

}

private void checkBoxMin\_CheckedChanged(object sender, EventArgs e)

{

RefreshImages();

}

private void checkBoxSeeBack\_CheckedChanged(object sender, EventArgs e)

{

RefreshImages();

}

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

{

Graphics g1 = e.Graphics;

var width = pictureBoxFunc.Width;

var height = pictureBoxFunc.Height;

var points = waveManager.GetFuncPoints(width, height);

g1.DrawLine(Pens.Red, new Point(0, 0), new Point(width, 0));

g1.DrawLine(Pens.Blue, new Point(0, height - 1), new Point(width, height - 1));

g1.DrawLine(Pens.Gray, new Point(0, height / 2), new Point(width, height / 2));

g1.DrawLines(Pens.Black, points);

}

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

{

double freq;

double phase;

if (Double.TryParse(textBoxFreq.Text.Replace('.', ','), out freq) &&

Double.TryParse(textBoxPhase.Text.Replace('.', ','), out phase))

{

if (freq > 100)

{

textBoxFreq.Text = "100";

freq = 100;

}

else if (freq < -100)

{

textBoxFreq.Text = "-100";

freq = -100;

}

if (comboBoxFunc.Text == "cos")

phase += 90;

waveManager.Update((float)freq, ((float)phase \* 2 \* (float)Math.PI / 360f) % 720);

pictureBoxFunc.Refresh();

}

}

}

}

***Model.cs***

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp1.Model

{

public class WaveManager

{

private static WaveManager singlton = null;

private WaveManager() { }

public static WaveManager GetInstance()

{

if (singlton == null)

singlton = new WaveManager();

return singlton;

}

public List<Wave> waves = new List<Wave>();

int timerInterval;

float speed;

uint radius;

float functionTime = 0;

float waveFreq;

float wavePhase;

public Point[] GetFuncPoints(int pictureWidth, int pictureHeight)

{

float time = 1f / 100;

var width = pictureWidth / 2;

Point[] points = new Point[width + 1];

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

{

double y = Math.Sin(waveFreq \* (functionTime - (width - i) \* time) + wavePhase);

points[i] = new Point(i , (int)(pictureHeight / 2 - y \* pictureHeight / 2));

}

return points;

}

public float GetFuncTime()

{

return functionTime;

}

public void Reset()

{

functionTime = 0;

}

public void Update(int \_timerInterval, float \_speed, uint \_radius)

{

timerInterval = \_timerInterval;

speed = \_speed;

radius = \_radius;

}

public void Update(float \_waveFreq, float \_wavePhase)

{

waveFreq = \_waveFreq;

wavePhase = \_wavePhase;

}

public void Update(int \_timerInterval, float \_speed, uint \_radius, float \_waveFreq, float \_wavePhase)

{

timerInterval = \_timerInterval;

speed = \_speed;

radius = \_radius;

waveFreq = \_waveFreq;

wavePhase = \_wavePhase;

}

public void CheckExtremums()

{

float time = (float)timerInterval / 1000f;

float derivativeWas = waveFreq \* (float)Math.Cos(waveFreq \* (functionTime) + wavePhase);

float derivativeNow = waveFreq \* (float)Math.Cos(waveFreq \* (functionTime + time) + wavePhase);

float eps = (float)10e-6;

if (Math.Abs(derivativeWas) < eps)

derivativeWas = 0;

if (Math.Abs(derivativeNow) < eps)

derivativeNow = 0;

if (Math.Sign(derivativeWas) != Math.Sign(derivativeNow))

{

if (Math.Sign(derivativeWas) < 0 || Math.Sign(derivativeNow) > 0)

waves.Add(new Wave(false));

else if (Math.Sign(derivativeWas) > 0 || Math.Sign(derivativeNow) < 0)

waves.Add(new Wave(true));

}

functionTime += time;

//Debug.WriteLine(functionTime);

}

public void CalculateWaves()

{

for (int i = 0; i < waves.Count; )

{

Wave wave = waves[i];

wave.Position += speed / 1000f \* (float)timerInterval / radius;

if (waves[i].Position >= Math.PI)

waves.RemoveAt(i);

else

i++;

}

}

}

public class Wave

{

public float Position = 0;

readonly private bool isMax;

public Wave(bool \_isMax)

{

this.isMax = \_isMax;

}

public bool IsMax()

{

return isMax;

}

}

public class Ellipse

{

Graphics g;

public Bitmap bitmap;

public Ellipse(PictureBox pictureBoxImage)

{

bitmap = new Bitmap(pictureBoxImage.Width, pictureBoxImage.Height);

this.g = Graphics.FromImage(bitmap);

g.Clear(Color.White);

}

public PictureBox Draw\_Ellipse(PictureBox pictureBoxImage, uint Radius)

{

renew\_bitmap(pictureBoxImage);

g.DrawLine(Pens.Black, 0, (bitmap.Size.Height) / 2, bitmap.Size.Width, (bitmap.Size.Height) / 2);

g.DrawLine(Pens.Black, bitmap.Size.Width / 2, 0, (bitmap.Size.Width) / 2, bitmap.Size.Height);

g.DrawLine(Pens.Black, bitmap.Size.Width / 2, 0, (bitmap.Size.Width) / 2, bitmap.Size.Height);

g.DrawLine(Pens.Black, bitmap.Size.Width / 2 + 80, 0, (bitmap.Size.Width) / 2 - 80, bitmap.Size.Height);

g.DrawEllipse(Pens.Black, (bitmap.Size.Width) / 2 - Radius, (bitmap.Size.Height) / 2 - Radius, 2 \* Radius, 2 \* Radius);

g.DrawEllipse(Pens.Black, (bitmap.Size.Width) / 2 - Radius, (bitmap.Size.Height) / 2 - Radius / 2, 2 \* Radius, Radius);

pictureBoxImage.Image = bitmap;

return pictureBoxImage;

}

PictureBox renew\_bitmap(PictureBox pictureBoxImage)

{

pictureBoxImage.Image = null;

pictureBoxImage.BackColor = Color.White;

bitmap.Dispose();

g.Clear(Color.White);

bitmap = new Bitmap(pictureBoxImage.Width, pictureBoxImage.Height);

this.g = Graphics.FromImage(bitmap);

pictureBoxImage.Image = bitmap;

return pictureBoxImage;

}

}

}