

Содержание

[Введение 4](#_Toc58953501)

[1 Индивидуальный вариант 5](#_Toc58953502)

[2 Внешний вид главного окна 6](#_Toc58953503)

[3 Код эмиттера 7](#_Toc58953504)

[4 Код специальных точек 12](#_Toc58953505)

[5 Код частиц 15](#_Toc58953506)

[6 Код формы 20](#_Toc58953507)

[7 Описание работы интерфейса 31](#_Toc58953508)

[Заключение 45](#_Toc58953509)

[Список использованной литературы 46](#_Toc58953510)

Введение

В данной работе необходимо разработать графический интерфейс для управления системой частиц.

Система частиц — используемый в компьютерной графике способ представления объектов, не имеющих чётких геометрических границ (различные облака, туманности, взрывы, струи пара, шлейфы от ракет, дым, снег, дождь и т. п.).

Система частиц состоит из определённого (фиксированного или произвольного) количества частиц. Математически каждая частица представляется как материальная точка с дополнительными атрибутами, такими как внешний вид реализуемого с помощью метода render, скоростью, запасом жизни и т. п.

В ходе работы программы каждая частица изменяет своё состояние по определённому, общему для всех частиц системы, закону. Например, частица может подвергаться воздействию гравитации, менять размер, цвет, скорость и так далее, и, после проведения всех расчётов, частица визуализируется.

Новые частицы испускаются так называемым «эмиттером». Эмиттером может быть точка, тогда новые частицы будут возникать в одном месте. Так можно смоделировать, например, взрыв: эмиттером будет его центр. Эмиттером может быть отрезок прямой или плоскость: например, частицы дождя или снега должны возникать на высоко расположенной горизонтальной плоскости. Эмиттером может быть и произвольный геометрический объект: в этом случае новые частицы будут возникать на всей его поверхности.

Для реализации используется язык C#. C# – это язык программирования, предназначенный для разработки самых разнообразных приложений. Язык C# прост, строго типизирован и объектно-ориентирован. Благодаря множеству нововведений C# обеспечивает возможность быстрой разработки приложений, но при этом сохраняет выразительность и элегантность, присущую си подобным языкам.

1 Индивидуальный вариант

Реализовать debug режим, в котором рисуются вектора скорости каждой частицы, добавить кнопочку для остановки симуляции движения частиц, добавить кнопочку для пошагового выполнения симуляции, добавить trackbar для замедления симуляции, при наведении мышки на частицу, выводить ее координаты и значение Life, (необязательно) добавить кнопочку, которая позволяет выполнять симуляцию в обратную сторону, хотя бы на последние 10 шагов

**Дополнительно:** реализовать точку-область радар, все частицы, попадающие в заданную область должны подсвечиваться как-то цветом, в центре необходимо писать количество частиц, оказавшихся в зоне действия радара, используя колесико мыши менять размеры области

2 Внешний вид главного окна

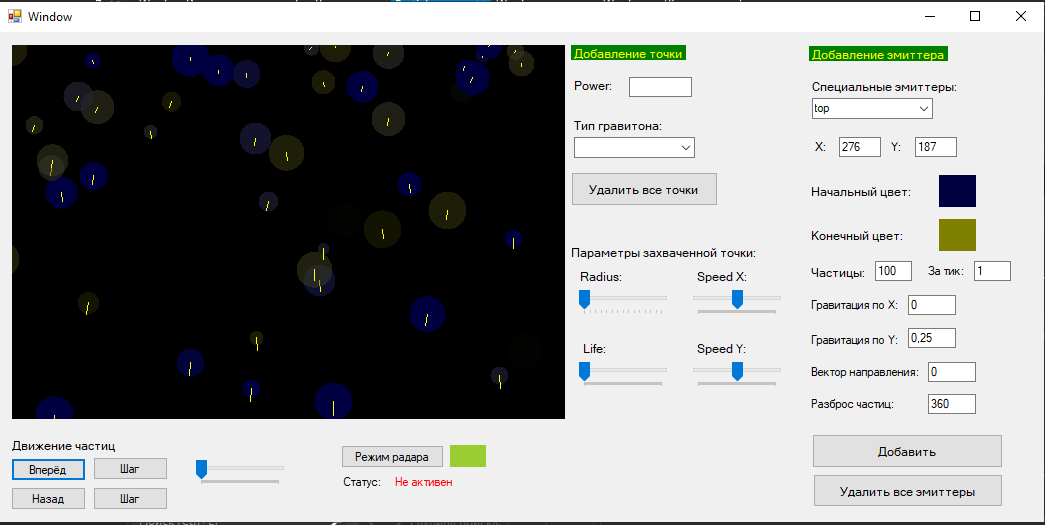


Рисунок 1 – Внешний вид главного окна

3 Код эмиттера

public class Emitter

{

public int X; // координата X центра эмиттера, будем ее использовать вместо MousePositionX

public int Y; // соответствующая координата Y

public int Direction = 0; // вектор направления в градусах куда сыпет эмиттер

public int Spreading = 360; // разброс частиц относительно Direction

public int SpeedMin = 0; // начальная минимальная скорость движения частицы

public int SpeedMax = 10; // начальная максимальная скорость движения частицы

public int RadiusMin = 6; // минимальный радиус частицы

public int RadiusMax = 20; // максимальный радиус частицы

public int LifeMin = 10; // минимальное количество жизни частицы

public int LifeMax = 200; // максимальное количество жизни частицы

public int ParticlesPerTick = 1; // кол-во частиц, создаваемых при каждом тике

public Color ColorFrom = Color.White; // начальный цвет частицы

public Color ColorTo = Color.FromArgb(0, Color.Black); // конечный цвет частиц

public List<Particle> particles = new List<Particle>(); //структура данных List, содержащая все частицы эмиттера

public List<Particle> beginStateParticle = new List<Particle>(); //структура данных List

//содержащая начальное состояние всех частиц (историю при их создании)

public List<IImpactPoint> impactPoints = new List<IImpactPoint>(); //структура данных List,

//содержащая точки притяжения/отторжения

public float GravitationX = 0;

public float GravitationY = 0;

public int MousePositionX;

public int MousePositionY;

public int Height;

public int Width;

public int ParticlesCount = 100; //кол-во частиц, которое будет создано эмиттером

public virtual Particle CreateParticle()

{

var particle = new ParticleInformation();

particle.FromColor = ColorFrom;

particle.ToColor = ColorTo;

particle.Life = Particle.rand.Next(LifeMin, LifeMax);

particle.LifeMax = LifeMax;

particle.LifeMin = LifeMin;

particle.RadiusMax = RadiusMax;

particle.RadiusMin = RadiusMin;

particle.Height = Height;

particle.Width = Width;

return particle;

}

public virtual void ResetParticle(Particle particle)

{

particle.Life = Particle.rand.Next(LifeMin, LifeMax);

particle.X = X;

particle.Y = Y;

particle.Height = Height;

particle.Width = Width;

var direction = Direction

+ (double)Particle.rand.Next(Spreading)

- Spreading / 2;

var speed = Particle.rand.Next(SpeedMin, SpeedMax);

particle.SpeedX = (float)(Math.Cos(direction / 180 \* Math.PI) \* speed);

particle.SpeedY = -(float)(Math.Sin(direction / 180 \* Math.PI) \* speed);

particle.Radius = Particle.rand.Next(RadiusMin, RadiusMax);

}

public void UpdateState()

{

int particlesToCreate = ParticlesPerTick; //сколько нужно создать частиц за 1 тик

for(int i = 0; i < particles.Count; i++)

{

//частицы живут либо до конца работы программы, либо до возвращения их назад (обратное движение)

particles[i].Life -= 1;

particles[i].X += particles[i].SpeedX;

particles[i].Y += particles[i].SpeedY;

foreach(var point in impactPoints)

{

point.ImpactParticle(particles[i]);

}

particles[i].SpeedX += GravitationX;

particles[i].SpeedY += GravitationY;

}

//цикл начальной инициализации всех данных

while((particlesToCreate >= 1) && (particles.Count < ParticlesCount))

{

particlesToCreate -= 1;

var particle = CreateParticle();

ResetParticle(particle);

particles.Add(particle);

beginStateParticle.Add((Particle)particle.Clone());

}

}

public void Render(Graphics g)

{

foreach(var particle in particles)

{

//каждой частице передаём позицию указателя мыши на холстке pictureDisplay

particle.MousePositionX = MousePositionX;

particle.MousePositionY = MousePositionY;

particle.FromColor = ColorFrom;

particle.ToColor = ColorTo;

foreach(var point in impactPoints)

{

if(point is ParticleRadar)

{

point.ImpactParticle(particle);

}

}

if(particle.ActiveRadar)

{

particle.DrawRadar(g);

}

else

particle.Draw(g);

}

foreach(var point in impactPoints)

{

point.Render(g);

}

}

//возврат к обратному состоянию всех частиц

public void UpdateStatePrevious()

{

if(particles.Count == 0)

return;

List<int> particleDeleted = new List<int>(); //элементы, которые необходимо удалить из

//beginLife и particles

for(int i = 0; i < particles.Count; i++)

{

particles[i].Life += 1;

//перемещение будет происходить до тех пор, пока динамически изменяемые величины каждой частицы

//не окажутся равными начальным (относительно истории их создания) значениям

if((particles[i].X == beginStateParticle[i].X) && (particles[i].Y == beginStateParticle[i].Y)

|| (particles[i].Life > beginStateParticle[i].Life))

{

particleDeleted.Add(i);

}

else if (particles[i].Life < beginStateParticle[i].Life)

{

//обратное действие движению

particles[i].X -= particles[i].SpeedX;

particles[i].Y -= particles[i].SpeedY;

foreach(var point in impactPoints)

{

point.AntiImpactParticle(particles[i]); //обратное притяжению действие

}

//обратное действие притяжению

particles[i].SpeedX -= GravitationX;

particles[i].SpeedY -= GravitationY;

}

}

Particle part = new Particle(-1, -1, 0, 0, 0, 0);

foreach(var index in particleDeleted)

{

particles[index] = part;

beginStateParticle[index] = part;

}

//удаление частиц

while(particles.IndexOf(part) >= 0)

particles.Remove(part);

while(beginStateParticle.IndexOf(part) >= 0)

beginStateParticle.Remove(part);

}

/// <summary>

/// Поиск частицы, на которую был совершён клик

/// </summary>

/// <returns>Частица на которую был совершён клик</returns>

public Particle GetClickedParticle()

{

Particle part = null;

foreach(var particle in particles)

{

if(particle.MouseInParticle(MousePositionX, MousePositionY)

&& (particle.Life >= LifeMin))

{

part = particle;

break;

}

}

return part;

}

/// <summary>

/// Подсчёт количества частиц принадлежащих области действия радара

/// </summary>

/// <returns>Количество частиц принадлежащих области действия радара</returns>

public int CounterActiveRadar()

{

int counter = 0; //возврат числа частиц, которые попали в область действия радара

foreach(var particle in particles)

{

if(particle.ActiveRadar)

counter++;

}

return counter;

}

/// <summary>

/// Отменяет принадлежность к области радара всем частицам

/// </summary>

public void AllNoActiveParticle()

{

foreach(var particle in particles)

{

particle.ActiveRadar = false;

}

}

}

/// <summary>

/// Эмиттер, генерирующий частицы с верхней части холста

/// </summary>

public class TopEmitter : Emitter

{

public override void ResetParticle(Particle particle)

{

base.ResetParticle(particle);

particle.X = Particle.rand.Next(Width);

particle.Y = 0;

particle.SpeedY = 1;

particle.SpeedX = Particle.rand.Next(-2, 2);

}

}

/// <summary>

/// Эмиттер, генерирующий частицы с левой части холста

/// </summary>

public class LeftEmitter : Emitter

{

public override void ResetParticle(Particle particle)

{

base.ResetParticle(particle);

particle.X = 0;

particle.Y = Particle.rand.Next(Height);

particle.SpeedY = Particle.rand.Next(-2, 2);

particle.SpeedX = 1;

}

}

/// <summary>

/// Эмиттер, генерирующий частицы с правой части холста

/// </summary>

public class RightEmitter : Emitter

{

public override void ResetParticle(Particle particle)

{

base.ResetParticle(particle);

particle.X = Width;

particle.Y = Particle.rand.Next(Height);

particle.SpeedY = Particle.rand.Next(-2, 2);

particle.SpeedX = -1;

}

}

/// <summary>

/// Эмиттер, генерирующий частицы с нижней части холста

/// </summary>

public class BottomEmitter : Emitter

{

public override void ResetParticle(Particle particle)

{

base.ResetParticle(particle);

particle.X = Particle.rand.Next(Width);

particle.Y = Height;

particle.SpeedY = -1;

particle.SpeedX = Particle.rand.Next(-2, 2);

}

}

4 Код специальных точек

public abstract class IImpactPoint

{

public float X;

public float Y;

public abstract void ImpactParticle(Particle particle);

public abstract void AntiImpactParticle(Particle particle);

public virtual void Render(Graphics g)

{

g.FillEllipse(

new SolidBrush(Color.White),

X - 5,

Y - 5,

10,

10

);

}

}

/// <summary>

/// Класс точек притяжения

/// </summary>

public class GravityPoint : IImpactPoint

{

public int Power = 100;

public override void ImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY);

if(r + particle.Radius < Power / 2)

{

float r2 = (float)Math.Max(100, gX \* gX + gY \* gY);

particle.SpeedX += gX \* Power / r2;

particle.SpeedY += gY \* Power / r2;

}

}

public override void AntiImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY);

if(r + particle.Radius < Power / 2)

{

float r2 = (float)Math.Max(100, gX \* gX + gY \* gY);

particle.SpeedX -= gX \* Power / r2;

particle.SpeedY -= gY \* Power / r2;

}

}

public override void Render(Graphics g)

{

g.DrawEllipse(

new Pen(Color.Red),

X - Power / 2,

Y - Power / 2,

Power,

Power

);

}

}

/// <summary>

/// Класс точек отторжения

/// </summary>

public class AntiGravityPoint : IImpactPoint

{

public int Power = 100;

public override void AntiImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY);

if(r + particle.Radius < Power / 2)

{

float r2 = (float)Math.Max(100, gX \* gX + gY \* gY);

particle.SpeedX += gX \* Power / r2;

particle.SpeedY += gY \* Power / r2;

}

}

public override void ImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY);

if(r + particle.Radius < Power / 2)

{

float r2 = (float)Math.Max(100, gX \* gX + gY \* gY);

particle.SpeedX -= gX \* Power / r2;

particle.SpeedY -= gY \* Power / r2;

}

}

public override void Render(Graphics g)

{

g.DrawEllipse(

new Pen(Color.Yellow),

X - Power / 2,

Y - Power / 2,

Power,

Power

);

}

}

/// <summary>

/// Класс реализующий логику радара

/// </summary>

public class ParticleRadar : IImpactPoint

{

public int Power = 100;

public Color color = Color.Green; //цвет в который нужно перекрасить частицы

public int count = 0; //количество частиц

public override void ImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY);

if(r + particle.Radius < Power / 2)

{

particle.FromColor = color;

particle.ToColor = color;

particle.ActiveRadar = true;

}

else

particle.ActiveRadar = false;

}

public override void AntiImpactParticle(Particle particle)

{

ImpactParticle(particle);

}

public override void Render(Graphics g)

{

g.DrawEllipse(

new Pen(Color.OrangeRed),

X - Power / 2,

Y - Power / 2,

Power,

Power

);

}

}

5 Код частиц

/// <summary>

/// Класс частицы

/// </summary>

public class Particle : ICloneable //подключение ICloneable для возможности создания клонов объекта (его копий)

{

public bool ActiveRadar = false; //находится ли частица под действием радара

public Color FromColor;

public Color ToColor;

//количественное значение радиуса и его ограничение

public int Radius;

public int RadiusMin = 2;

public int RadiusMax = 10;

//координаты центра частицы

public float X;

public float Y;

//скорость

public float SpeedX;

public float SpeedY;

//количество жизней и её ограничение

public float Life;

public float LifeMin = 0;

public float LifeMax = 100;

//координаты экрана

public int MousePositionX;

public int MousePositionY;

//параметры экрана

public int Height;

public int Width;

public static Random rand = new Random();

public Particle(float x, float y, int radius, float life, float speedx, float speedy)

{

X = x;

Y = y;

Radius = radius;

Life = life;

SpeedX = speedx;

SpeedY = speedy;

}

public Particle()

{

var direction = (double)rand.Next(360);

var speed = 1 + rand.Next(10);

SpeedX = (float)(Math.Cos(direction / 180 \* Math.PI) \* speed);

SpeedY = -(float)(Math.Sin(direction / 180 \* Math.PI) \* speed);

Radius = RadiusMin + rand.Next(RadiusMax);

Life = LifeMin + rand.Next((int)LifeMax);

}

public virtual void DrawRadar(Graphics g) //функция рисования используемая радаром

{

Draw(g);

}

public virtual void Draw(Graphics g)

{

float k = Math.Min(1f, Life / 100);

int alpha = (int)(k \* 255);

var color = Color.FromArgb(alpha, Color.Black);

var b = new SolidBrush(color);

g.FillEllipse(b, X - Radius, Y - Radius, Radius \* 2, Radius \* 2);

b.Dispose();

}

/// <summary>

/// Метод определяющий находится ли указатель мыши в области частицы

/// </summary>

/// <param name="mouseX">Координата X мыши на холсте PictureBox</param>

/// <param name="mouseY">Координата Y мыши на холсте PictureBox</param>

/// <returns>true - если указатель мыши находится в области частицы, и false - в другом случае</returns>

public bool MouseInParticle(float mouseX, float mouseY)

{

return (MousePositionX <= X + Radius) && (MousePositionX >= X - Radius)

&& (MousePositionY <= Y + Radius) && (MousePositionY >= Y - Radius);

}

/// <summary>

/// Метод создающий копию текущего объекта частицы

/// </summary>

/// <returns>Новый объект, содержащий данные текущего объекта</returns>

public object Clone()

{

return new Particle(X, Y, Radius, Life, SpeedX, SpeedY);

}

}

public class ParticleColorful : Particle

{

public static Color MixColor(Color color1, Color color2, float k) //ункция формирующая цвет

{

int alpha = (int)(color2.A \* k + color1.A \* (1 - k)),

red = (int)(color2.R \* k + color1.R \* (1 - k)),

green = (int)(color2.G \* k + color1.G \* (1 - k)),

blue = (int)(color2.B \* k + color1.B \* (1 - k));

//с помощью тернарных операторов сделано ограничение значений, для

//автоматической обработки ситуаций "крайних" границ

alpha = (alpha < 0) ? 0 : (alpha > 255) ? 255 : alpha;

red = (red < 0) ? 0 : (red > 255) ? 255 : red;

green = (green < 0) ? 0 : (green > 255) ? 255 : green;

blue = (blue < 0) ? 0 : (blue > 255) ? 255 : blue;

return Color.FromArgb(alpha, red, green, blue);

}

public override void Draw(Graphics g) //рисование цветной частицы

{

float k = Math.Min(1f, Life / 100);

var color = MixColor(ToColor, FromColor, k);

var b = new SolidBrush(color);

g.FillEllipse(b, X - Radius, Y - Radius, Radius \* 2, Radius \* 2);

b.Dispose();

}

}

/// <summary>

/// Класс частиц, информирующих пользователя о своём состоянии

/// </summary>

public class ParticleInformation : ParticleColorful

{

public override void DrawRadar(Graphics g)

{

float k = Math.Min(1f, Life / 100);

var color = MixColor(ToColor, FromColor, k);

var b = new SolidBrush(color);

g.FillEllipse(b, X - Radius, Y - Radius, Radius \* 2, Radius \* 2);

}

public override void Draw(Graphics g) //рисование цветной частицы с информацией

{

float k = Math.Min(1f, Life / 100);

var color = MixColor(ToColor, FromColor, k);

var b = new SolidBrush(color);

g.FillEllipse(b, X - Radius, Y - Radius, Radius \* 2, Radius \* 2);

float coordX = X + SpeedX, coordY = Y + SpeedY;

if(Life >= LifeMin) //рисование частиц с информацией доступно только если количество жизни у частицы

//больше её минимального ограничителя

{

Pen pen = new Pen(Color.Yellow); //создание кисти

g.DrawLine(pen, X, Y, coordX, coordY); //создание линии (вектор направления частицы)

if((MousePositionX <= X + Radius) && (MousePositionX >= X - Radius)

&& (MousePositionY <= Y + Radius) && (MousePositionY >= Y - Radius))

{

g.DrawEllipse(pen, X - Radius, Y - Radius, Radius \* 2, Radius \* 2);

var stringFormat = new StringFormat();

stringFormat.Alignment = StringAlignment.Near;

stringFormat.LineAlignment = StringAlignment.Near;

var text = $"X: {X}\nY: {Y}\nRadius: {Radius}\nLife: {Life}";

var font = new Font("Verdana", 10);

var size = g.MeasureString(text, font);

//вычисление позиции четырёхугольника с информацией на холсте

float x = MousePositionX + size.Width / 4,

y = MousePositionY + size.Height / 4;

while((x + size.Width) > Width)

x--;

while((y + size.Height) > Height)

y--;

g.FillRectangle(

new SolidBrush(Color.OrangeRed),

x,

y,

size.Width,

size.Height

);

g.DrawString(

text,

font,

new SolidBrush(Color.Black),

x,

y,

stringFormat

);

}

pen.Dispose();

}

b.Dispose();

}

}

public override void Draw(Graphics g)

{

float k = Math.Min(1f, Life / 100);

// так как k уменшается от 1 до 0, то порядок цветов обратный

var color = MixColor(ToColor, FromColor, k);

var b = new SolidBrush(color);

g.FillEllipse(b, X - Radius, Y - Radius, Radius \* 2, Radius \* 2);

b.Dispose();

}

}

6 Код формы

public partial class Window : Form

{

List<Emitter> emitters = new List<Emitter>();

Emitter emitter;

bool start = false; //началось ли движение частиц вперёд или нет

bool previous = false; //началось ли движение частиц назад или нет

bool radar = false; //режим радара

Particle currentParticle = null; //изменяемая частица

List<string> typeGraviton = null;

List<string> typeStandEmitter = null;

ParticleRadar radarParticle = null; //объект радара

int MouseClickX = 0;

int MouseClickY = 0;

int MousePositionX = -100;

int MousePositionY = -100;

public Window()

{

InitializeComponent();

//первоначальная инициализация

this.emitter = new TopEmitter

{

Width = picDisplay.Width,

Height = picDisplay.Height,

Direction = 0,

Spreading = 10,

SpeedMin = 10,

SpeedMax = 10,

GravitationY = 0.25f,

ColorFrom = Color.White,

ColorTo = Color.FromArgb(0, Color.Black),

ParticlesPerTick = 1,

X = picDisplay.Width / 2,

Y = picDisplay.Height / 2,

};

emitters.Add(this.emitter);

\_tbrRadiusParticle.Maximum = emitter.RadiusMax;

\_tbrRadiusParticle.Minimum = emitter.RadiusMin;

\_tbrLifeParticle.Maximum = emitter.LifeMax;

\_tbrLifeParticle.Minimum = emitter.LifeMin;

\_tbrSpeedXParticle.Maximum = 100;

\_tbrSpeedXParticle.Minimum = -100;

\_tbrSpeedYParticle.Maximum = 100;

\_tbrSpeedYParticle.Minimum = -100;

picDisplay.Image = new Bitmap(picDisplay.Width, picDisplay.Height);

typeGraviton = new List<string> { "гравитон", "антигравитон" };

typeStandEmitter = new List<string> { "top", "left", "bottom", "right" };

foreach(var type in typeGraviton)

\_cmbTypeGraviton.Items.Add(type);

foreach(var type in typeStandEmitter)

\_cmbStandType.Items.Add(type);

\_cdFromColor.FullOpen = true;

\_picFromColor.BackColor = \_cdFromColor.Color = Color.White;

\_cdToColor.FullOpen = true;

\_picToColor.BackColor = \_cdToColor.Color = Color.Black;

\_txtStatusRadar.ForeColor = Color.Red;

\_txtStatusRadar.Text = "Не активен";

radarParticle = new ParticleRadar{

color = Color.YellowGreen

};

pictureBox1.BackColor = Color.YellowGreen;

}

private void EmptyParticleInterface()

{

\_txtRadiusValue.Text = "";

\_txtLifeValue.Text = "";

\_txtSpeedXParticle.Text = "";

\_txtSpeedYParticle.Text = "";

}

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

{

int counter = 0; //количество частиц из всех эммитеров, которые попали в область действия радара

using(var g = Graphics.FromImage(picDisplay.Image))

{

g.Clear(Color.Black);

foreach(var emit in emitters)

{

if(start)

emit.UpdateState();

else if(previous)

emit.UpdateStatePrevious(); //обратное движение частиц

emit.Render(g);

counter += emit.CounterActiveRadar(); //подсчёт частиц, которые попали в область действия радара

}

if(counter > 0)

{ //рисование чисел в области радара (количество частиц попавших в его область действия)

g.DrawString($"{counter}",

new Font("Verdana", 12),

new SolidBrush(Color.White),

MousePositionX,

MousePositionY + 15);

}

}

picDisplay.Invalidate();

}

private void picDisplay\_MouseMove(object sender, MouseEventArgs e) //обработка движений указателя мыши по холсту

{

foreach(var emit in emitters)

{

emit.MousePositionX = MousePositionX = e.X;

emit.MousePositionY = MousePositionY = e.Y;

}

if(radar)

{

radarParticle.X = e.X;

radarParticle.Y = e.Y;

}

}

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

{

start = !start;

previous = false;

currentParticle = null;

EmptyParticleInterface();

}

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

{

timer1.Interval = \_tbSpeedNext.Value;

\_txtSpeedNext.Text = \_tbSpeedNext.Value.ToString();

}

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

{

start = true;

timer1\_Tick(sender, e);

start = false;

}

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

{

if(currentParticle == null)

return;

currentParticle.Radius = \_tbrRadiusParticle.Value;

\_txtRadiusValue.Text = currentParticle.Radius.ToString();

}

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

{

if(currentParticle == null)

return;

currentParticle.Life = \_tbrLifeParticle.Value;

\_txtLifeValue.Text = currentParticle.Life.ToString();

}

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

{

if(currentParticle == null)

return;

currentParticle.SpeedX = \_tbrSpeedXParticle.Value;

\_txtSpeedXParticle.Text = currentParticle.SpeedX.ToString();

}

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

{

if(currentParticle == null)

return;

currentParticle.SpeedY = \_tbrSpeedYParticle.Value;

\_txtSpeedYParticle.Text = currentParticle.SpeedY.ToString();

}

private void picDisplay\_MouseLeave(object sender, EventArgs e)

{

foreach(var emit in emitters)

{

emit.MousePositionX = -100;

emit.MousePositionY = -100;

}

}

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

{

start = false;

previous = !previous;

}

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

{

previous = true;

timer1\_Tick(sender, e);

previous = false;

}

private bool ValidateComboBox(ComboBox cmb, List<string> lists)

{

return (lists.IndexOf(cmb.Text) >= 0);

}

private void picDisplay\_MouseClick(object sender, MouseEventArgs e) //обработка нажатий на клавишу мыши

{

MouseClickX = e.X;

MouseClickY = e.Y;

\_txtEmitterX.Text = MouseClickX.ToString();

\_txtEmitterY.Text = MouseClickY.ToString();

if(radar) //если включен режим радара прерываем обработку нажатия на клавишу мыши

return;

if(e.Button == MouseButtons.Right) //система создания точки (при щелчке на левую кнопку мыши)

{

if((!ValidateComboBox(\_cmbTypeGraviton, typeGraviton))

|| (\_txtInputPower.Text.Length == 0)){

MessageBox.Show("Ошибка: не корректное заполнение полей для добавления точки!");

return;

}

//добавление точек двух типов: гравитона и антигравитона

if(\_cmbTypeGraviton.Text.Equals(typeGraviton[0]))

{

foreach(var emit in emitters)

{

emit.impactPoints.Add(new GravityPoint

{

X = emit.MousePositionX,

Y = emit.MousePositionY,

Power = int.Parse(\_txtInputPower.Text)

});

}

}else if(\_cmbTypeGraviton.Text.Equals(typeGraviton[1]))

{

foreach(var emit in emitters)

{

emit.impactPoints.Add(new AntiGravityPoint

{

X = emit.MousePositionX,

Y = emit.MousePositionY,

Power = int.Parse(\_txtInputPower.Text)

});

}

}

}else if(e.Button == MouseButtons.Left) //обработка нажатия на правую кнопку мыши

{

if((start == true) || (previous == true))

{

MessageBox.Show("Ошибка: нельзя захватить частицу при движении частиц");

return;

}

foreach(var emit in emitters)

{

currentParticle = emit.GetClickedParticle();

if(currentParticle != null)

break;

}

if(currentParticle == null)

return;

//открытие режима корректирования частицы

\_tbrRadiusParticle.Value = currentParticle.Radius;

\_tbrLifeParticle.Value = (int)currentParticle.Life;

\_tbrSpeedXParticle.Value = (int)(currentParticle.SpeedX);

\_tbrSpeedYParticle.Value = (int)(currentParticle.SpeedY);

\_txtRadiusValue.Text = currentParticle.Radius.ToString();

\_txtLifeValue.Text = ((int)currentParticle.Life).ToString();

\_txtSpeedXParticle.Text = ((int)(currentParticle.SpeedX)).ToString();

\_txtSpeedYParticle.Text = ((int)(currentParticle.SpeedY)).ToString();

}

}

private void \_txtInputPower\_KeyPress(object sender, KeyPressEventArgs e) //обработка нажатия на клавишу

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

}

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

{

if(radar)

{

MessageBox.Show("Ошибка: вы находитесь в режиме радара, выйдите из него чтобы продолжить настройку холста");

return;

}

foreach(var emit in emitters)

emit.impactPoints.Clear();

}

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

{

//обработка выбора цвета

if(\_cdFromColor.ShowDialog() == DialogResult.Cancel)

return;

\_picFromColor.BackColor = \_cdFromColor.Color;

}

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

{

if(\_cdToColor.ShowDialog() == DialogResult.Cancel)

return;

\_picToColor.BackColor = \_cdToColor.Color;

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

\_txtParticlePerTick.Text = "0";

}

private void timer2\_Tick\_1(object sender, EventArgs e) //обработка рендеринга

{

int counter = 0;

using(var g = Graphics.FromImage(picDisplay.Image))

{

g.Clear(Color.Black);

foreach(var emit in emitters)

{

emit.Render(g);

counter += emit.CounterActiveRadar();

}

if(counter > 0)

{

g.DrawString($"{counter}",

new Font("Verdana", 12),

new SolidBrush(Color.White),

MousePositionX,

MousePositionY + 15);

}

}

picDisplay.Invalidate();

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

}

private void \_btnAddEmitter\_Click(object sender, EventArgs e) //добавление эммитера

{

if(radar)

{

MessageBox.Show("Ошибка: вы находитесь в режиме радара, выйдите из него чтобы продолжить настройку холста");

return;

}

if((\_txtGravitationX.Text.Length == 0) || (\_txtGravitationY.Text.Length == 0)

|| (\_txtSpreading.Text.Length == 0) || (\_txtDirection.Text.Length == 0)

|| (\_txtCountParticle.Text.Length == 0) || (\_txtParticlePerTick.Text.Length == 0))

{

MessageBox.Show("Ошибка: не корректное заполнение полей для добавления эмиттера!");

return;

}

float grX = float.Parse(\_txtGravitationX.Text), grY = float.Parse(\_txtGravitationY.Text);

int sp = int.Parse(\_txtSpreading.Text), dir = int.Parse(\_txtDirection.Text);

int count = int.Parse(\_txtCountParticle.Text), tick = int.Parse(\_txtParticlePerTick.Text);

int coordX = int.Parse(\_txtEmitterX.Text), coordY = int.Parse(\_txtEmitterY.Text);

if(ValidateComboBox(\_cmbStandType, typeStandEmitter))

{

if(\_cmbStandType.Text.Equals(typeStandEmitter[0]))

{

emitters.Add(new TopEmitter

{

Width = picDisplay.Width,

Height = picDisplay.Height,

Direction = dir,

Spreading = sp,

SpeedMin = 10,

SpeedMax = 10,

GravitationY = grY,

GravitationX = grX,

ColorFrom = \_picFromColor.BackColor,

ColorTo = Color.FromArgb(0, \_picToColor.BackColor),

ParticlesPerTick = tick,

ParticlesCount = count,

X = picDisplay.Width / 2,

Y = picDisplay.Height / 2,

});

}else if(\_cmbStandType.Text.Equals(typeStandEmitter[1]))

{

emitters.Add(new LeftEmitter

{

Width = picDisplay.Width,

Height = picDisplay.Height,

Direction = dir,

Spreading = sp,

SpeedMin = 10,

SpeedMax = 10,

GravitationY = grY,

GravitationX = grX,

ColorFrom = \_picFromColor.BackColor,

ColorTo = Color.FromArgb(0, \_picToColor.BackColor),

ParticlesPerTick = tick,

ParticlesCount = count,

X = picDisplay.Width / 2,

Y = picDisplay.Height / 2,

});

}

else if(\_cmbStandType.Text.Equals(typeStandEmitter[2]))

{

emitters.Add(new BottomEmitter

{

Width = picDisplay.Width,

Height = picDisplay.Height,

Direction = dir,

Spreading = sp,

SpeedMin = 10,

SpeedMax = 10,

GravitationY = grY,

GravitationX = grX,

ColorFrom = \_picFromColor.BackColor,

ColorTo = Color.FromArgb(0, \_picToColor.BackColor),

ParticlesPerTick = tick,

ParticlesCount = count,

X = picDisplay.Width / 2,

Y = picDisplay.Height / 2,

});

}

else if(\_cmbStandType.Text.Equals(typeStandEmitter[3]))

{

emitters.Add(new RightEmitter

{

Width = picDisplay.Width,

Height = picDisplay.Height,

Direction = dir,

Spreading = sp,

SpeedMin = 10,

SpeedMax = 10,

GravitationY = grY,

GravitationX = grX,

ColorFrom = \_picFromColor.BackColor,

ColorTo = Color.FromArgb(0, \_picToColor.BackColor),

ParticlesPerTick = tick,

ParticlesCount = count,

X = picDisplay.Width / 2,

Y = picDisplay.Height / 2,

});

}

}

else

{

emitters.Add(new Emitter

{

Width = picDisplay.Width,

Height = picDisplay.Height,

Direction = dir,

Spreading = sp,

SpeedMin = 10,

SpeedMax = 10,

GravitationY = grY,

GravitationX = grX,

ColorFrom = \_picFromColor.BackColor,

ColorTo = Color.FromArgb(0, \_picToColor.BackColor),

ParticlesPerTick = tick,

ParticlesCount = count,

X = coordX,

Y = coordY,

});

}

if(emitters.Count >= 2) //наследование точек притяжения/отторжения с предыдущего эмиттера

foreach(var point in emitters[emitters.Count - 2].impactPoints)

emitters[emitters.Count - 1].impactPoints.Add(point);

}

private void \_cmbStandType\_TextChanged(object sender, EventArgs e) //установка специальных типов эммитера

{

if(!ValidateComboBox(\_cmbStandType, typeStandEmitter))

return;

\_txtEmitterX.Text = (picDisplay.Width / 2).ToString();

\_txtEmitterY.Text = (picDisplay.Height / 2).ToString();

if(\_cmbStandType.Text.Equals(typeStandEmitter[0]))

{

\_txtGravitationX.Text = "0";

\_txtGravitationY.Text = "0,25";

}else if(\_cmbStandType.Text.Equals(typeStandEmitter[1]))

{

\_txtGravitationX.Text = "0,25";

\_txtGravitationY.Text = "0";

}

else if(\_cmbStandType.Text.Equals(typeStandEmitter[2]))

{

\_txtGravitationX.Text = "0";

\_txtGravitationY.Text = "-0,25";

}

if(\_cmbStandType.Text.Equals(typeStandEmitter[3]))

{

\_txtGravitationX.Text = "-0,25";

\_txtGravitationY.Text = "0";

}

\_txtCountParticle.Text = (100).ToString();

\_txtDirection.Text = "0";

\_txtSpreading.Text = "360";

\_txtParticlePerTick.Text = "1";

}

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

{

if(radar)

{

MessageBox.Show("Ошибка: вы находитесь в режиме радара, выйдите из него чтобы продолжить настройку холста");

return;

}

emitters.Clear();

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

if(((number == ',') || (number == '-')) && (\_txtGravitationX.Text.IndexOf(number) < 0)

&& (\_txtGravitationX.Text.Length > 0))

{

e.Handled = false;

}

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

if(((number == ',') || (number == '-')) && (\_txtGravitationY.Text.IndexOf(number) < 0)

&& (\_txtGravitationY.Text.Length > 0))

{

e.Handled = false;

}

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

}

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

{

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

}

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

{

if(\_txtCountParticle.Text.Length == 0)

{

e.Handled = true;

return;

}

char number = e.KeyChar;

if((!Char.IsDigit(number)) && (number != 8))

{

e.Handled = true;

}

if(number == 8)

{

return;

}

if(int.Parse(\_txtParticlePerTick.Text + number) > int.Parse(\_txtCountParticle.Text))

{

e.Handled = true;

}

}

private void button1\_Click\_1(object sender, EventArgs e)

{

//активация / отключение радара

radar = !radar;

if(radar)

{

\_txtStatusRadar.ForeColor = Color.Green;

\_txtStatusRadar.Text = "Активен";

foreach(var emit in emitters)

emit.impactPoints.Add(radarParticle); //при активации радара, он добавляется

//всем эмиттерам, для того, чтобы захватить все частицы

}

else

{

\_txtStatusRadar.ForeColor = Color.Red;

\_txtStatusRadar.Text = "Не активен";

foreach(var emit in emitters)

{

int index = (-1);

for(int i = 0; (i < emit.impactPoints.Count) && (index < 0); i++)

{

if(emit.impactPoints[i] is ParticleRadar)

index = i;

}

if(index >= 0)

emit.impactPoints.RemoveAt(index);

emit.AllNoActiveParticle();

}

}

}

private void pictureBox1\_Click(object sender, EventArgs e)

{

if(\_cdRadarColor.ShowDialog() == DialogResult.Cancel)

return;

pictureBox1.BackColor = \_cdRadarColor.Color;

radarParticle.color = \_cdRadarColor.Color; //установка цвета, которым радар будет выделять

//частицы попавшие в область радара

}

//обработка движения ползунка мыши

private void picDisplay\_MouseWheel(object sender, MouseEventArgs e)

{

if(radar)

{

int number = e.Delta \* SystemInformation.MouseWheelScrollLines / 30;

if((radarParticle.Power + number) >= 0)

radarParticle.Power += number;

}

}

}

7 Описание работы интерфейса

Для начала симуляции движения точек эмиттеров необходимо нажать на кнопку “Вперёд” или “Шаг”, которые запускают процесс симуляции точек с изменением их состояния и положения в пространстве с течением времени (симуляция направленная вперёд).

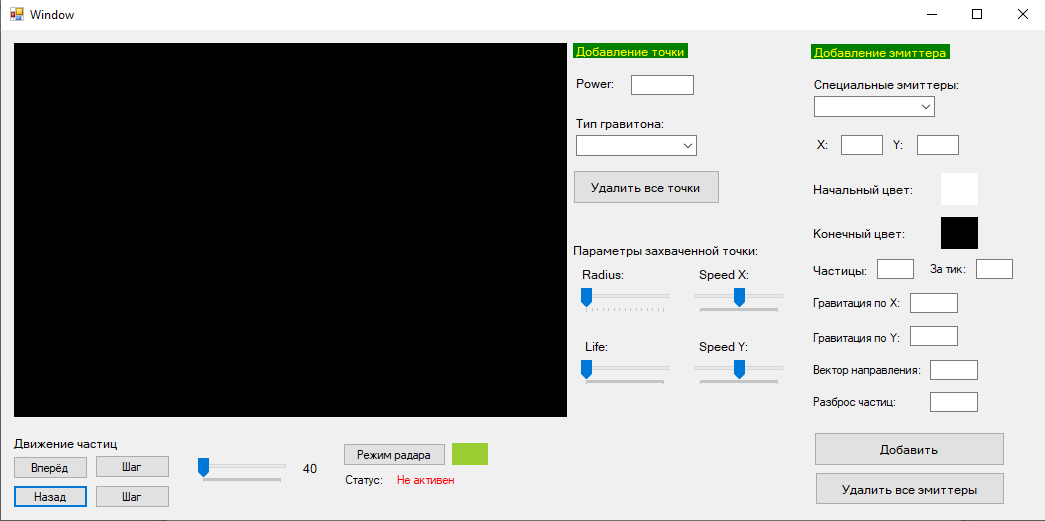


Рисунок 2 – До нажатия на кнопку “Вперёд”



Рисунок 3 – После нажатия на кнопку “Вперёд”

Симуляция может быть пошаговая (нужно нажать на клавишу “Шаг”), при которой частицы меняют своё состояние на 1 шаг вперёд.

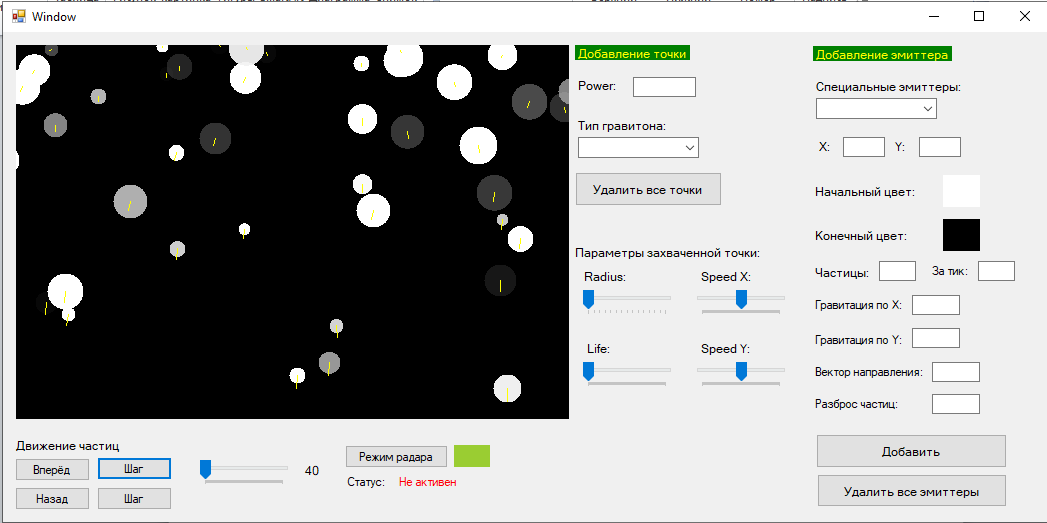


Рисунок 4 – После нажатия на кнопку “Шаг”, результат от предыдущего состояния (см. рис. 3)

Также, скорость симуляции можно регулировать ползунком treckbar, расположенный напротив кнопки “Шаг”. Задать значения можно от 40 до 1000. Скорость изменяется для симуляций направленных вперёд и назад.

Для симуляции направленной назад (переход к прошлому состоянию частиц), необходимо нажать на кнопку “Назад” или “Шаг”, расположенную напротив неё. Нажав на данную кнопку, запустится процесс обратной симуляции, который вернёт частицы в обратное состояние (данный процесс возвращает к жизни все не действительные частицы).

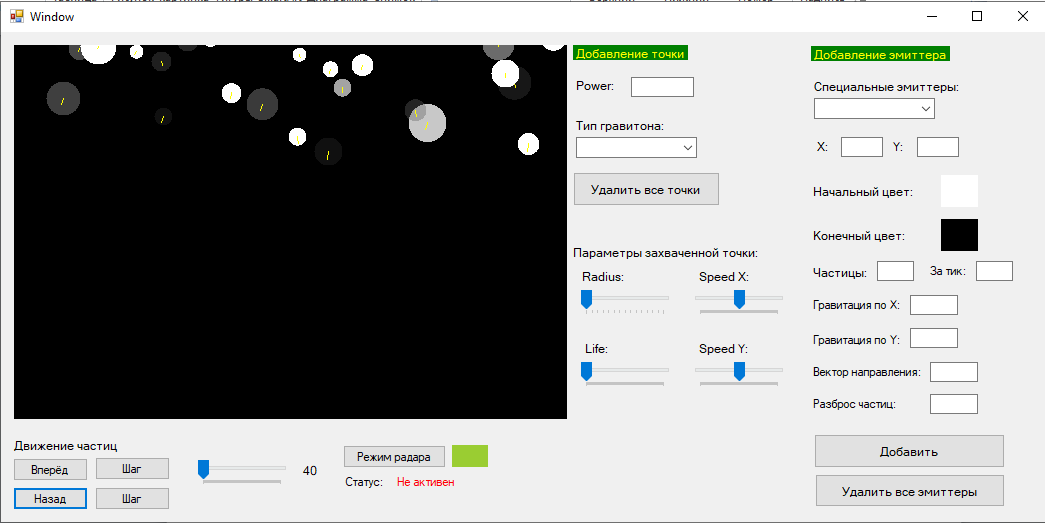


Рисунок 5 – После нажатия на кнопку “Назад”, результат от предыдущего состояния (см. рис. 4)

Интерфейс пользователя позволяет создавать различное количество эмиттеров, как специальных (верхний, нижний, левый и правый эмиттеры), так и абсолютно различные (пользовательские). Чтобы выбрать специальные эмиттеры, необходимо в CheckBox расположенном ниже текста “Специальные эмиттеры:” выбрать один из соответствующих специальных эмиттеров. Можно задавать свои цветовые палитры для визуализации жизненного цикла частиц, общее количество частиц генерируемое эмиттером, количество частиц генерируемое эмиттером за тик, а также можно задавать собственную гравитацию, однако, специальные эмиттеры игнорируют параметры X, Y, Вектор направления и Разброс частиц.

Для добавления эмиттера к общему числу эмиттеров (их может быть большое множество) достаточно после задания настроек эмиттера нажать на кнопку “Добавить”, а затем, чтобы визуально наблюдать результат работы эмиттера, нажать на одну из кнопок запуска симуляции (рекомендуется нажать на кнопку “Вперёд”, если до добавления эмиттера симуляция не была активирована).

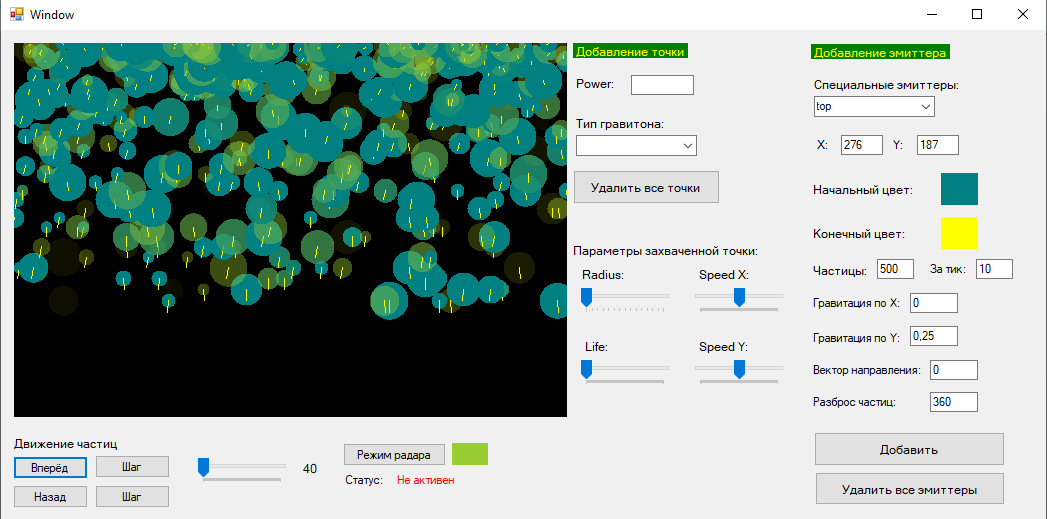


Рисунок 6 – Результат добавления специального эмиттера “top”

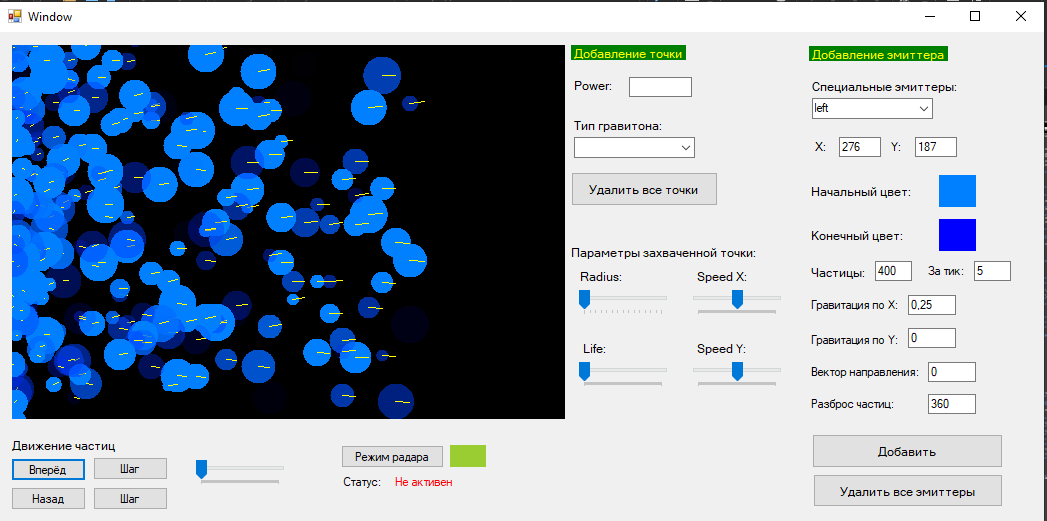


Рисунок 7 – Результат добавления специального эмиттера “left”

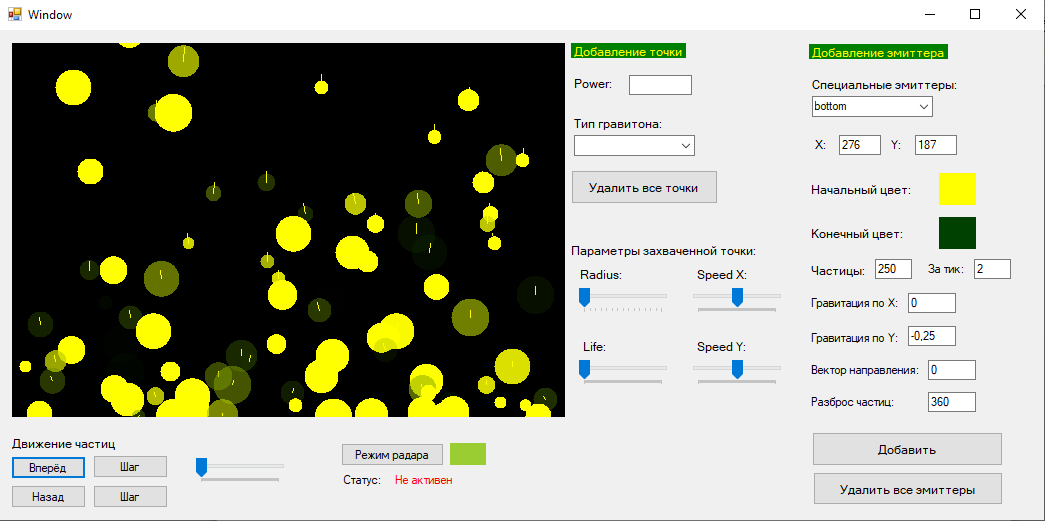


Рисунок 8 – Результат добавления специального эмиттера “bottom”

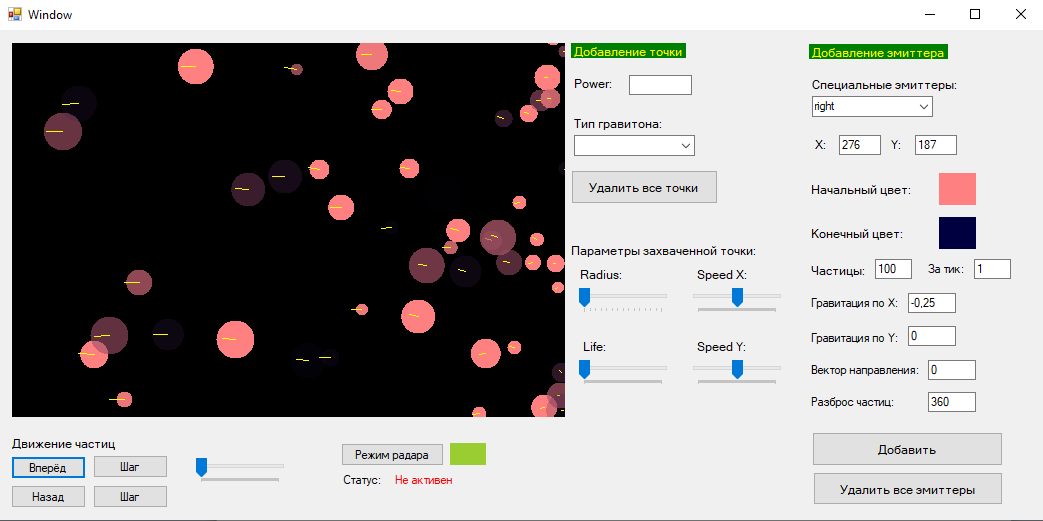


Рисунок 9 – Результат добавления специального эмиттера “right”

При добавлении своего эмиттера (пользовательского), необходимо учитывать реагирование программы на щелчёк мыши. При щелчке мыши автоматически меняются значения координат эмиттера X и Y (они будут равны координатам позиции указателя мыши на холсте, заданным PictureBox).

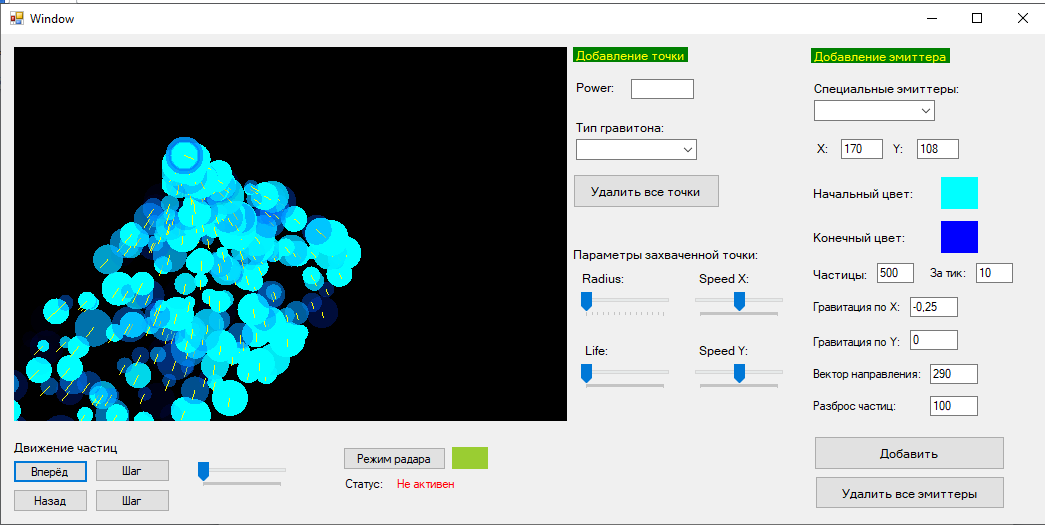


Рисунок 10 – Результат добавления пользовательского эмиттера

Для удаления всех эмиттеров нужно нажать на кнопку “Удалить все эмиттеры”, после чего холст очистится независимо от того, какая симуляция запущена.

Можно добавить гравитон и антигравитон в пространство, для разнообразия движения точек. Чтобы добавить специальные точки нужно вырать тип точки в CheckBox, расположенном ниже текста “Тип гравитона:”, добавить значение силы притяжение / отторжения в соответсвующий TextBox, раположенный напротив текста “Power” и при ведении мыши по холсту (карте, на которой расположены частицы) нужно произвести щелчёк мыши нажав правую кнопку мыши, на холсте появится гравитон или антигравитон соответствующей силы, которые будут влиять на поведение движения частиц после запуска симуляции движения частиц.

Стоит учесть, что добавление специальных точек допустимо только при использовании хотя бы одного эмиттера, т.е. если не добавлено ни одного эмиттера, не будет добавлено ни одной специальной точки, поскольку частиц в таком случае не существует, а это означает, что нет смысла изменять состояние частиц.

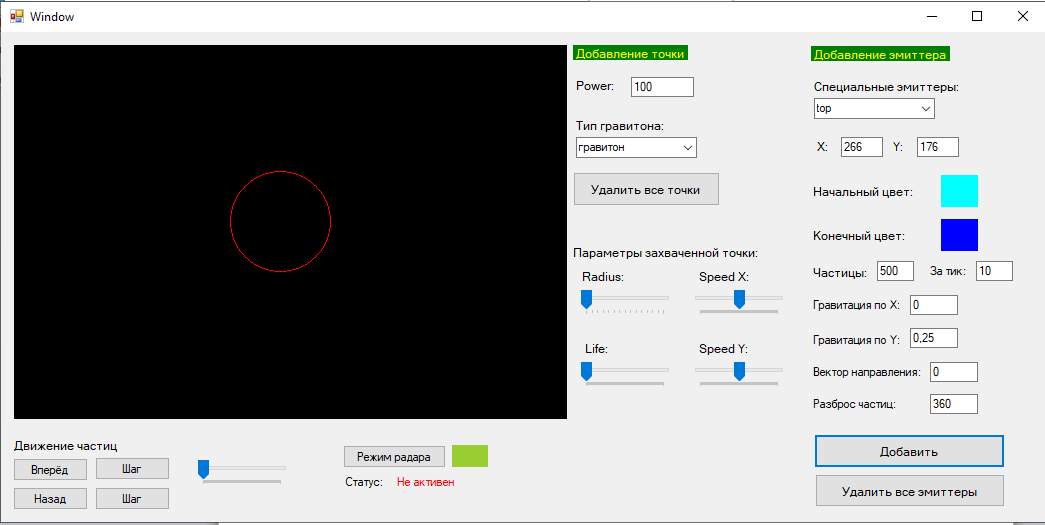


Рисунок 11 – Добавление гравитона с силой 100

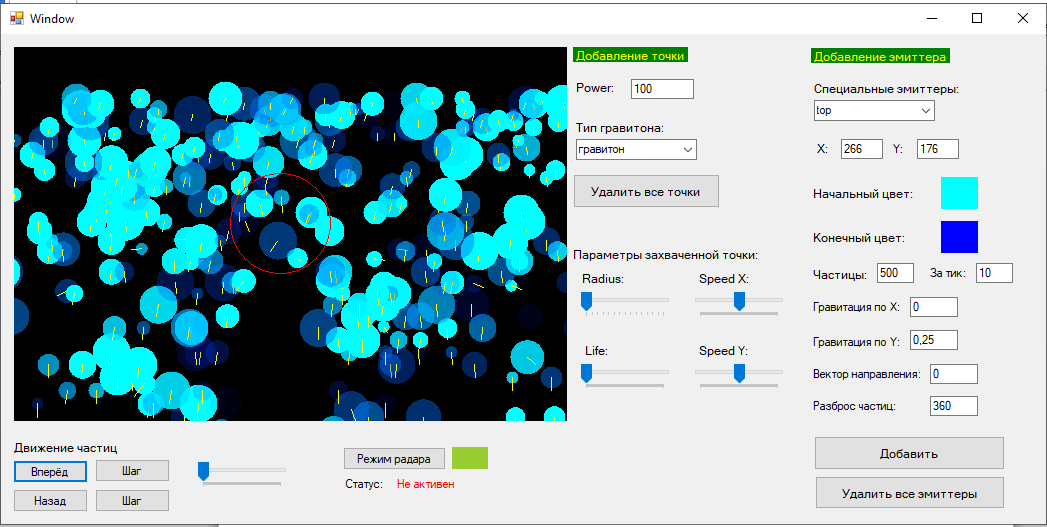


Рисунок 12 – Визуальный результат добавления гравитона (см. рис 11) при старте симуляции

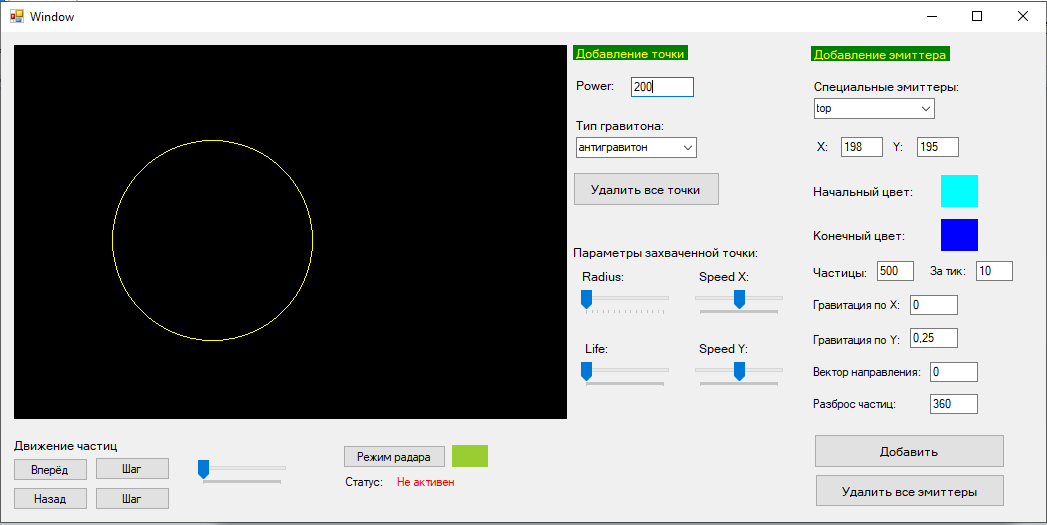


Рисунок 13 – Добавление антигравитона с силой 200

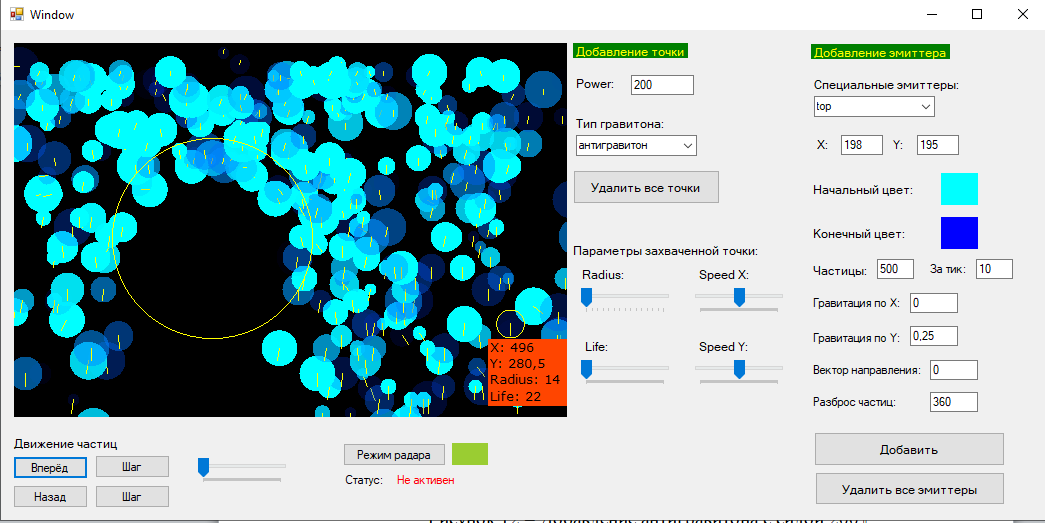


Рисунок 14 – Визуальный результат добавления антигравитона (см. рис 13) при старте симуляции

Для удаления всех специальных точек нужно нажать на кнопку “Удалить все точки”.

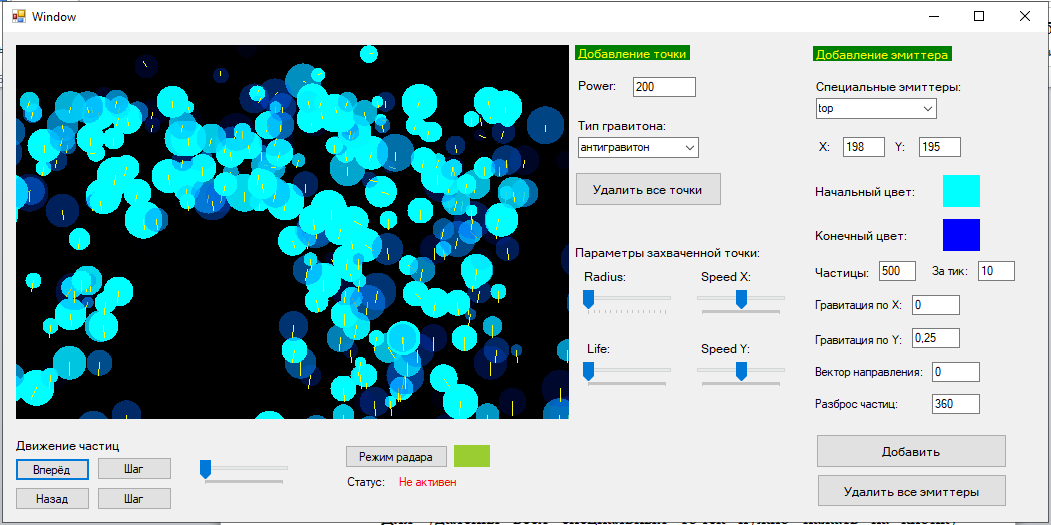


Рисунок 15 – Результат удаления всех точек с предыдущего состояния симуляции (см. рис. 14)

Можно добавить любое количество гравитонов и антигравитонов.

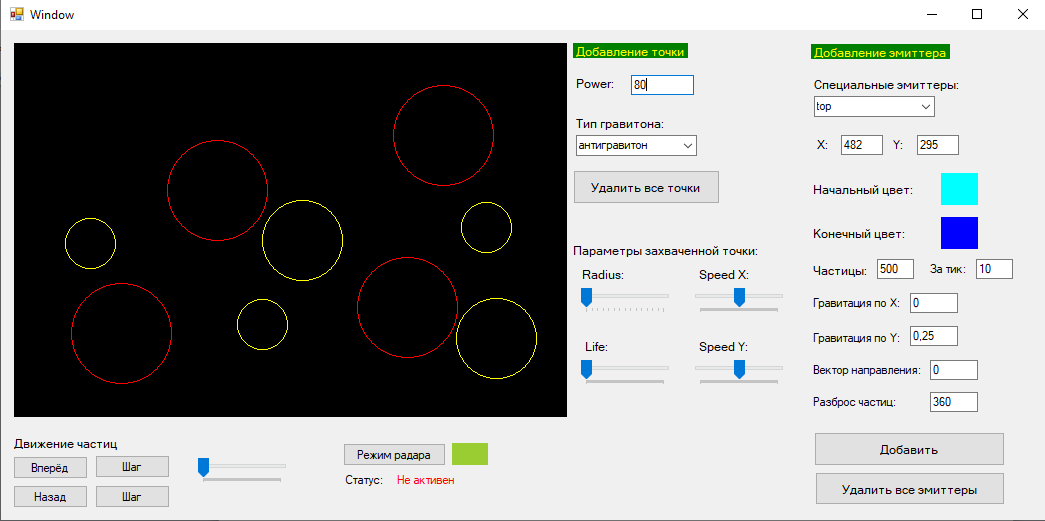


Рисунок 16 – Добавление некоторого количества гравитонов и антигравитонов

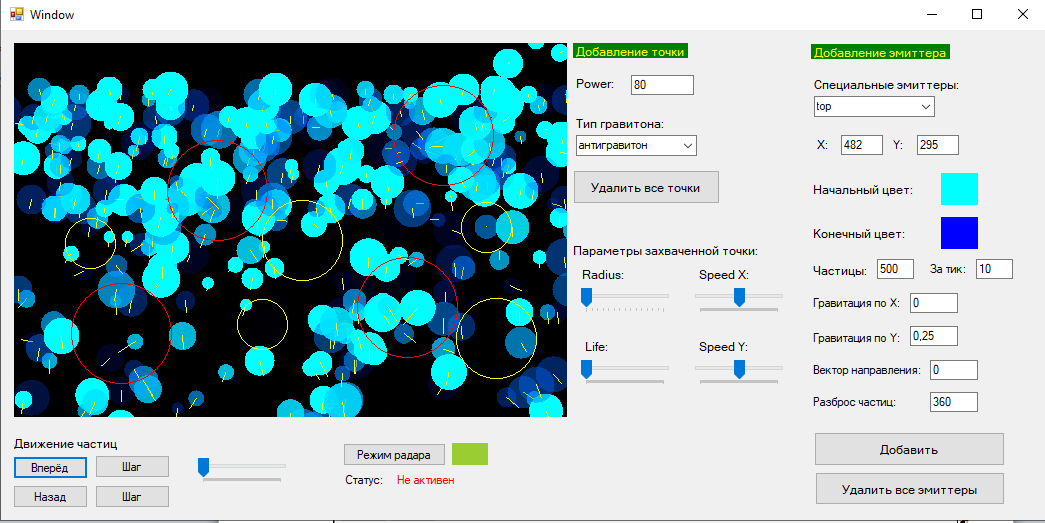


Рисунок 17 – Результат симуляции после добавления некоторого количества гравитонов и антигравитонов (см. рис. 16)

Для работы радара необходимо активировать режим радара. Это можно сделать, нажав на кнопку “Режим радара”, чуть ниже показан статус радара “Не активирован” / “Активирован”. Радар окрашивает все частицы в определённый цвет, который можно задать, щелкнув в цветное окно напротив кнопки “Режим радара”. Следует учесть, что при активации радара добавление, удаление эмиттеров, добавление и удаление специальных точек, а также захват частицы – игнорируется.

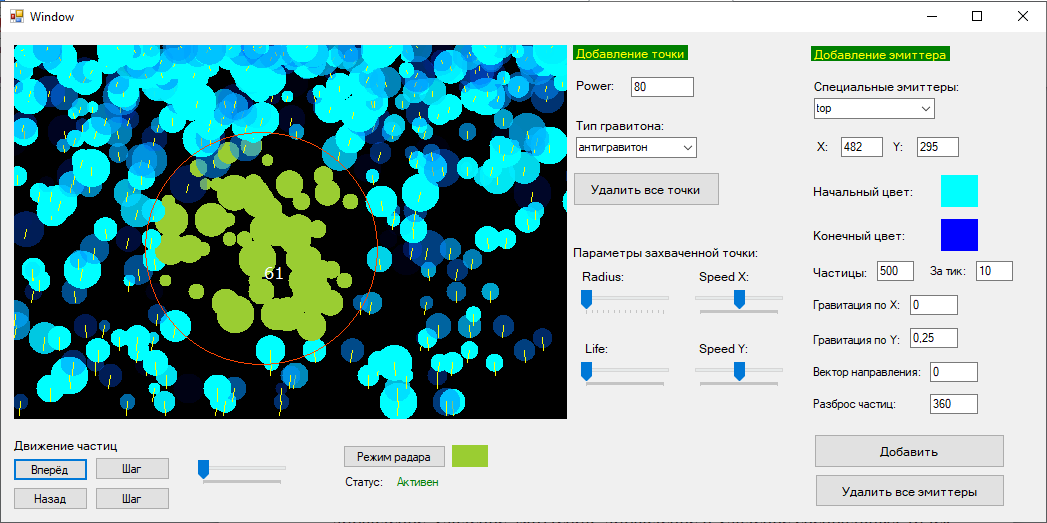


Рисунок 18 – Пример использования радара

С помощью колёсика мыши можно увеличивать область радара или уменьшать её.

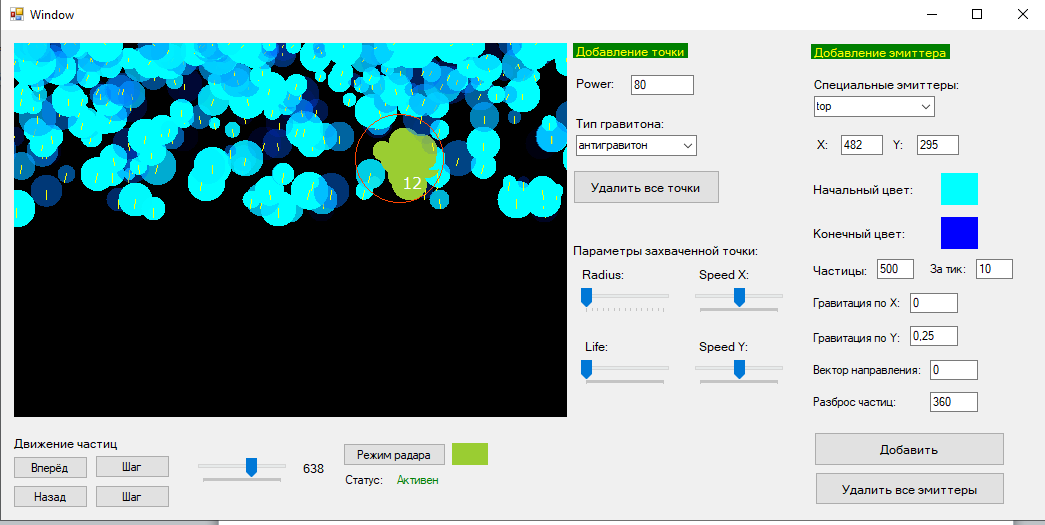


Рисунок 19 – Пример использования радара

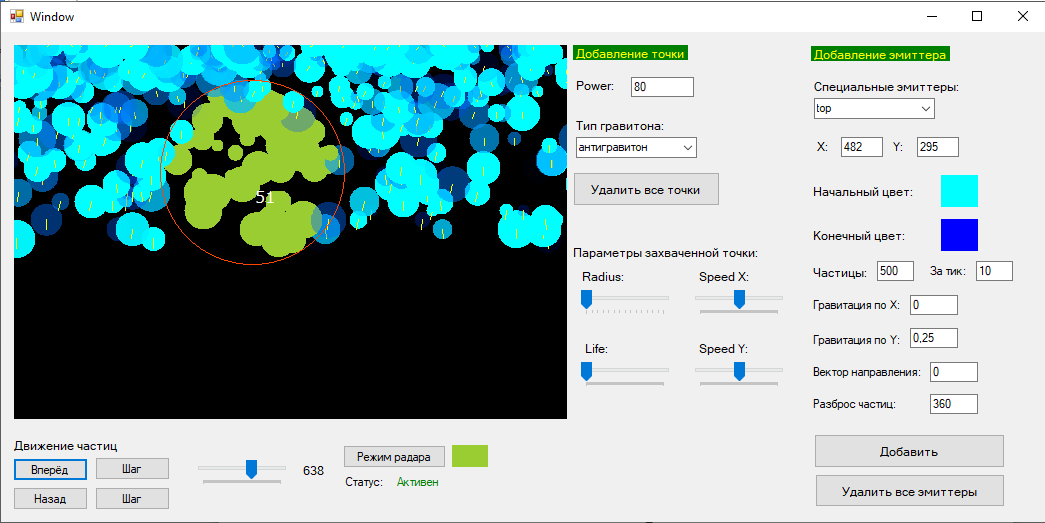


Рисунок 20 – Увеличение используемого ранее радара (см. рис. 19)

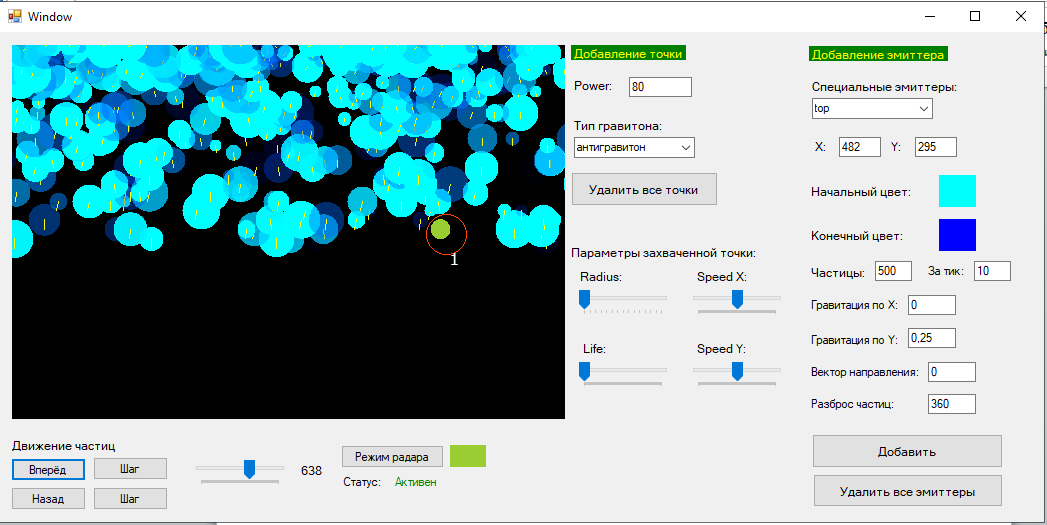


Рисунок 21 – Уменьшение используемого ранее радара (см. рис. 19)

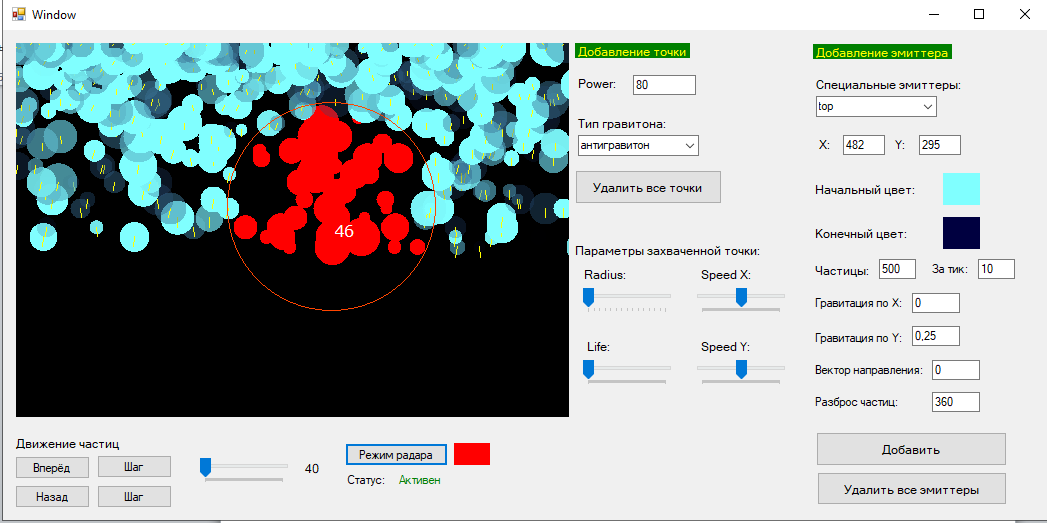


Рисунок 22 – Изменение цвета закрашиваемых частиц в области радара

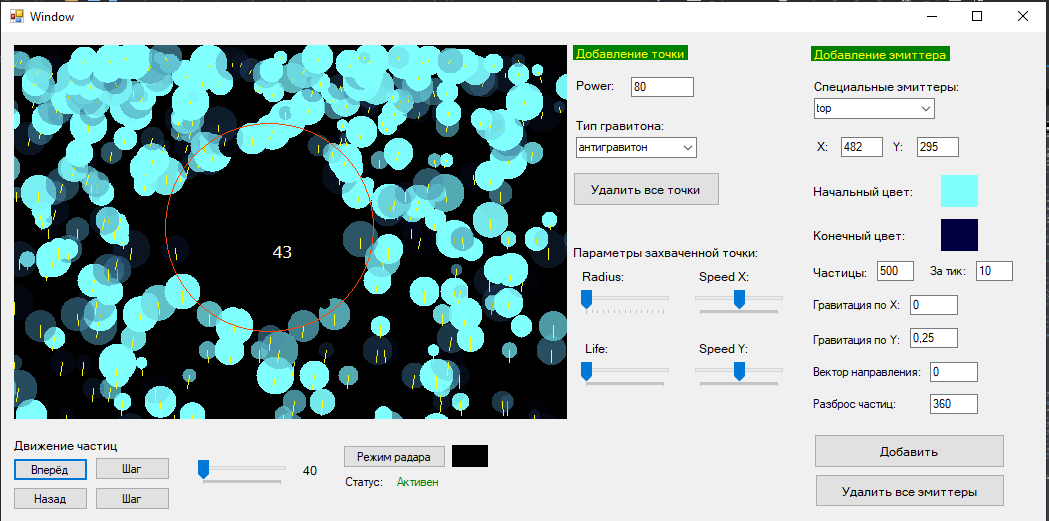


Рисунок 23 – Изменение цвета закрашиваемых частиц в области радара

В пользовательском интерфейсе также имеется возможность захватить точку и изменить её параметры, такие как: радиус, количество жизни, скорость по X и скорость по Y. Стоит учесть, что при старте симуляции (в любом направлении) частицу захватить нельзя. Для изменения состояния частицы необходимо остановить симуляцию, для этого нужно нажать на кнопку, с помощью которой симуляция была начата: “Вперёд” или “Назад”.

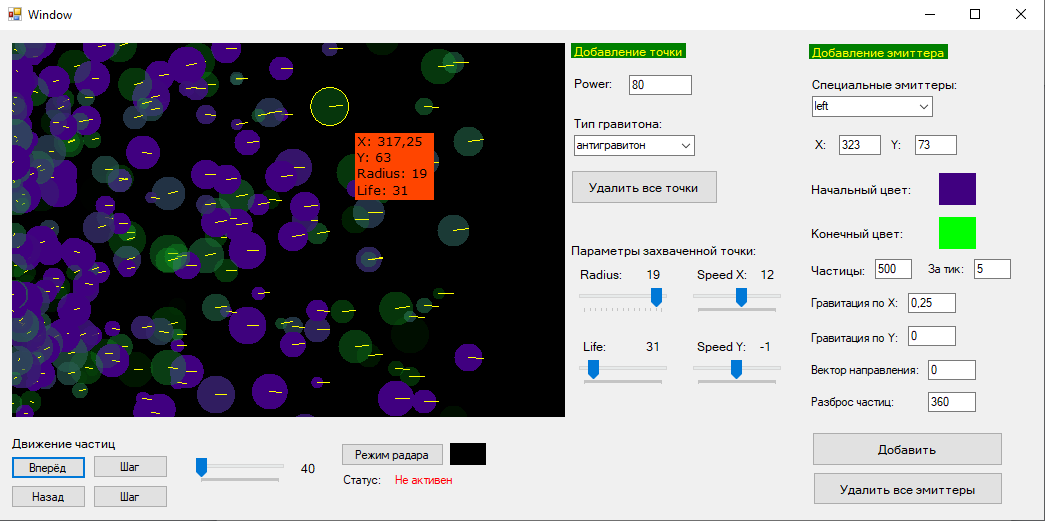


Рисунок 24 – Пример захвата частицы

Чтобы захватить частицу, необходимо направить на неё указатель мыши, а затем нажать правую кнопку мыши. При захвате текущее состояние частицы транслируется на treackbars, с помощью которых можно изменить параметры частицы. Также, при наведении указателя мыши на частицу высвечивается четырёхугольник, в котором содержится информация о частице, в том числе: координаты (X;Y) центра частицы, радиус частицы и количество жизней у частицы. Для каждой частицы рисуется вектор скорости. Визуальное отображение частиц согласовано с индивидуальным вариантом.

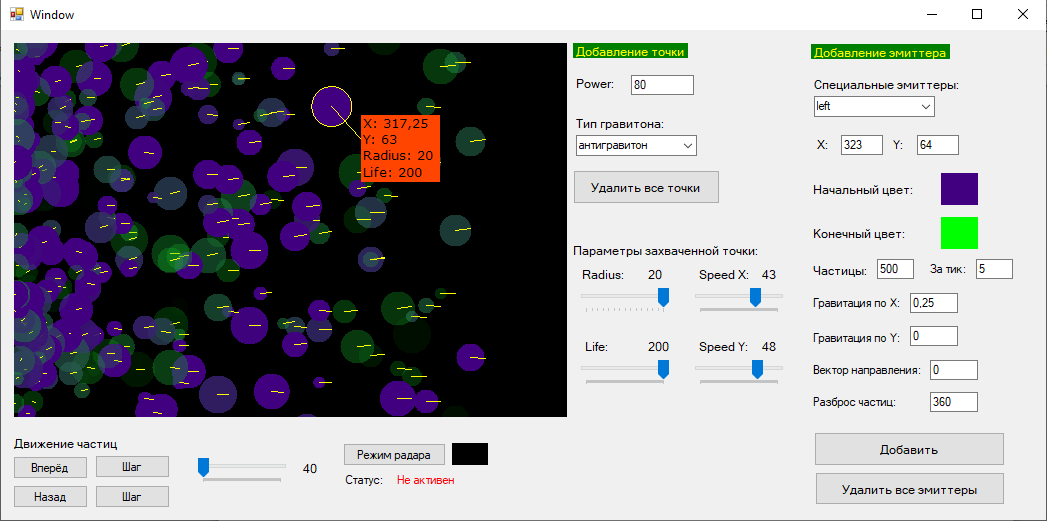


Рисунок 25 – Пример изменения параметров частицы (см. рис. 24)

Изменение состояния частицы, а также её расположение в пространстве изменятся после старта симуляции или при шаге симуляции.

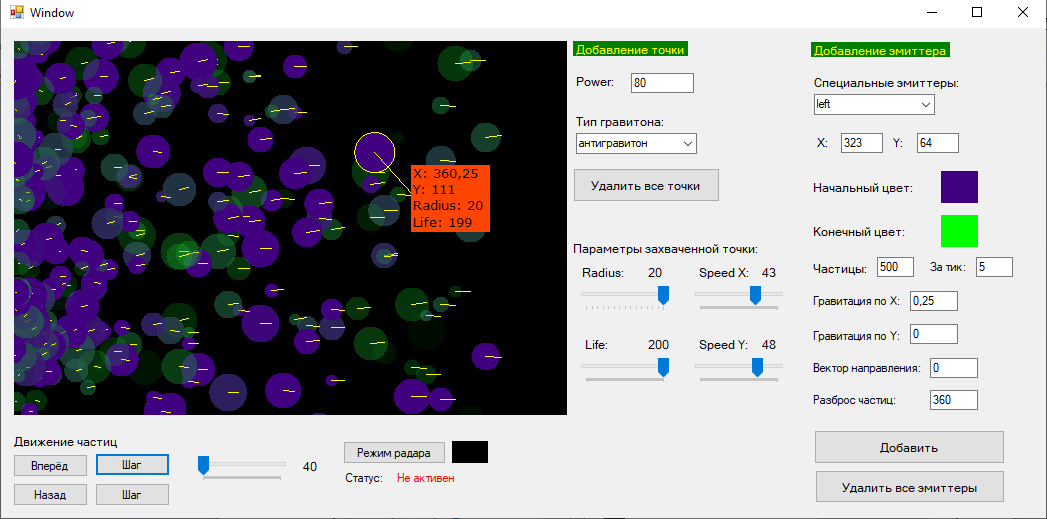


Рисунок 26 – Изменение состояния и положения частицы после шага симуляции (см. рис. 25)

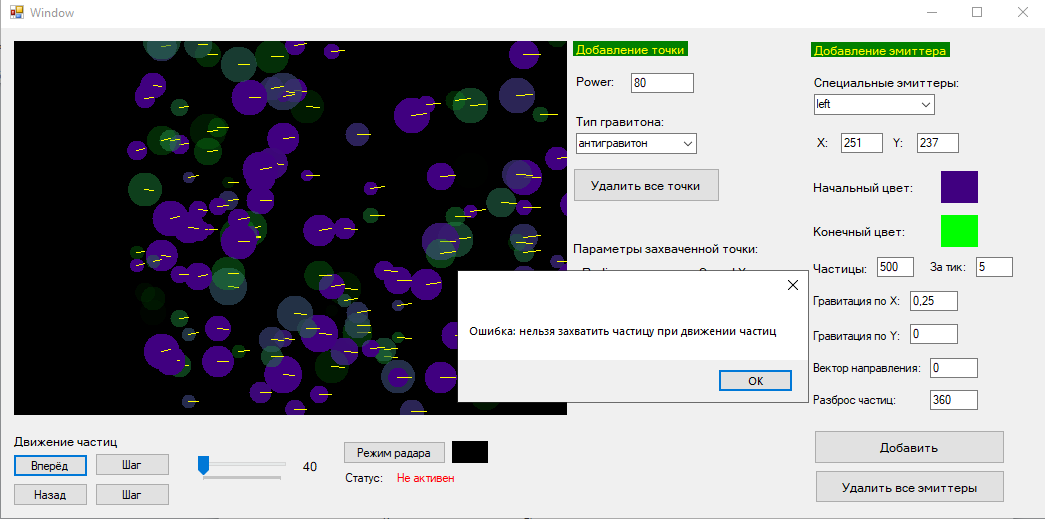


Рисунок 27 – Попытка захвата частицы во время старта симуляции

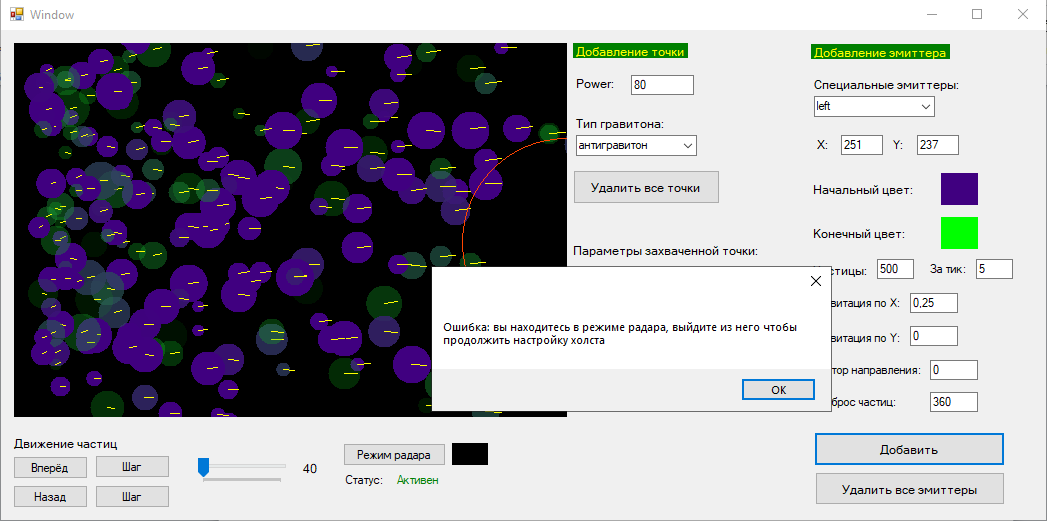


Рисунок 28 – Попытка доступа к настройкам системы частиц при активированном радаре

# Заключение

В рамках курсовой работы было разработано приложение, позволяющее управлять системой частиц. В приложении реализованы следующие возможности:

Реализован debug режим:

* в котором рисуются вектора скорости каждой частицы.
* кнопка “Вперёд” для начала/остановки симуляции движения частиц
* кнопка “Шаг” для пошагового выполнения симуляции
* treckbar для замедления симуляции
* При наведении мышки на частицу выводится следующая информация о частице: координаты (X;Y) центра частицы, радиус частицы (Radius) и количество жизней у частицы (Life)
* добавлена кнопка “Назад” для выполнения обратной симуляции, а также кнопка “Шаг” для пошагового выполнения обратной симуляции

Реализована система радар:

* все частицы, попадающие в заданную область подсвечиваются определённым цветом
* в центра пишется количество частиц, попавших в область действия радара
* используя колёсико мыши можно изменять размеры области действия радара

Приложение было всесторонне протестировано и отлажено. Все поставленные в рамках курсовой работы задачи были выполнены.

В рамках выполнения данной работы был получен обширный спектр сведений об объектно-ориентированном языке программирования C#, разработки графических приложений и эмуляции сложных динамических систем.

# Список использованной литературы

1. Троелсен, Эндрю Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 c.
2. Вагнер, Билл С# Эффективное программирование / Билл Вагнер. - М.: ЛОРИ, 2013. - 320 c.
3. Ишкова, Э. А. Самоучитель С#. Начала программирования / Э.А. Ишкова. - М.: Наука и техника, 2013. - 496 c.
4. Биллинг В.А. Основы программирования на C#. - Т.: Интернет-университет информационных технологий, Бином, 2012. - 488 с.
5. Евдокимов П.В. C# на примерах. - М.: Наука и техника,2016. - 304 с.
6. Нейгел К., Ивьен Б., Глинн Д., Уотсон К., Скиннер М. C# 4.0 и платформа .NET 4, 2011.
7. Система частиц, часть 1. [Электронный ресурс] // olive.tealeaf.su : чаинка, 2020. URL: <http://olive.tealeaf.su/particle-system.html> (дата обращения: 01.12.2020).