Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
| Институт информационных технологий и анализа данных |

наименование института

|  |
| --- |
| Допускаю к защите  Руководитель:  И.А.Серышева |
| И.О. Фамилия |

Разработка системы управления частицами на языке C#

наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

|  |
| --- |
| Технологии программирования |

|  |
| --- |
| 1.017.00.00 ПЗ |
| обозначение документа |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | ИСТб 21-1 |  |  |  | Морозов Д.И. |
|  | шифр группы |  | подпись |  | Фамилия И.О |
| Нормоконтроль |  |  |  |  | Серышева И.А. |
|  |  |  | подпись |  | Фамилия И.О |

|  |  |
| --- | --- |
| Курсовая работа защищена с оценкой |  |

Иркутск 2022 г.

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |  |
| --- | --- | --- |
| По курсу | Технологии программирования | |
| Студенту | Морозову Дмитрию Игоревичу | |
|  | (фамилия, инициалы) | |
| Тема работы: | Разработка системы управления частицами на языке C# | | |
| Исходные данные: | | Вариант 56 | |
| Рекомендуемая литература: | | | |
| 1. Троелсен, Эндрю Язык программирования C# 5.0 и платформа .NET 4.5 / Эндрю Троелсен. - М.: Вильямс, 2015. - 486 c  2. Система частиц, часть 1. [Электронный ресурс] // wave.tealeaf.su : чаинка, 2021. URL: http://wave.tealeaf.su/particle-system.html (дата обращения: 01.12.2022). | | | |
|  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Графическая часть на | - | | листах. | |
| Дата выдачи задания |  | 07 / 11 / 2022 г. | | |
| Задание получил студент | | | |  | |  | Морозов Д.И. |
|  | | | | подпись | |  | Фамилия И.О. |

|  |  |
| --- | --- |
| Дата представления работы руководителю | 19 / 12 / 2022 г. |
| Руководитель курсовой работы |  |  | Серышева И.А. |
|  | подпись | Фамилия И.О. |

Содержание

[Введение 4](#_Toc58847158)

[1 Индивидуальный вариант 5](#_Toc58847159)

[2 Внешний вид главного окна 6](#_Toc58847160)

[3 Код эмиттера 7](#_Toc58847161)

[4 Код специальных точек 9](#_Toc58847162)

[5 Код частиц 10](#_Toc58847163)

[6 Код формы 11](#_Toc58847164)

[7 Описание работы интерфейса 13](#_Toc58847165)

[Заключение 15](#_Toc58847166)

[Список использованной литературы 16](#_Toc58847167)

# 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

5. Реализовать точки перекрашивания частиц, попадая в радиус действия которой частицы меняют свой цвет на цвет указанный у точки.

* Реализовать раскрашивание, например, снега падающего с неба
* добавить trackbar с помощью которого можно перемещать точки перекрашивания
* (необязательно) добавить trackbar с помощью которого можно менять радиус точек перекрашивания
* (необязательно) добавить кнопочку которая будет переключать палитру

6.Реализовать точку-счетчик частиц, попадая в которую частица умирает, а на точке пишется сколько частиц она уже собрала.

* при клике мышкой добавлять новый счетчик
* (необязательно) при клике правой кнопкой мыши, удалять счетчик
* (необязательно) при увеличении счетчика менять насыщенность цвета счетчика

# 2 Внешний вид главного окна

Внешний вид главного окна представлен на рисунке 2.1:



Рисунок 2.1

# 3 Код эмиттера

public class Emitter

{

List<Particle> particles = new List<Particle>();

public List<IImpactPoint> impactPoints = new List<IImpactPoint>(); // <<< ТАК ВОТ

//

public int MousePositionX;

public int MousePositionY;

public float GravitationX = 0;

public float GravitationY = 1; // пусть гравитация будет силой один пиксель за такт, нам хватит

//

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 = 2; // минимальный радиус частицы

public int RadiusMax = 10; // максимальный радиус частицы

public int LifeMin = 20; // минимальное время жизни частицы

public int LifeMax = 100; // максимальное время жизни частицы

public int ParticlesPerTick = 1;

public int Score = 0;// Счёт уничтоженных частиц

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

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

public bool SpeedVector = false;

public Matrix GetTransform()

{

var matrix = new Matrix();

matrix.Translate(X, Y);

matrix.Rotate(0);

return matrix;

}

public virtual GraphicsPath GetGraphicsPath()

{

return new GraphicsPath();

}

public void UpdateState()

{

Score = ParticlesCount;

int particlesToCreate = ParticlesPerTick; // фиксируем счетчик сколько частиц нам создавать за тик

foreach (var particle in particles)

{

particle.SpeedVector = this.SpeedVector;

if (particle.Life <= 0) // если частицы умерла

{

Score -= 1;

if (particlesToCreate > 0)

{

/\* у нас как сброс частицы равносилен созданию частицы \*/

particlesToCreate -= 1; // поэтому уменьшаем счётчик созданных частиц на 1

ResetParticle(particle);

}

}

else

{

// каждая точка по-своему воздействует на вектор скорости

foreach (var point in impactPoints)

{

point.ImpactParticle(particle);

}

// это не трогаем

particle.SpeedX += GravitationX;

particle.SpeedY += GravitationY;

particle.X += particle.SpeedX;

particle.Y += particle.SpeedY;

}

}

// второй цикл меняем на while,

// этот новый цикл также будет срабатывать только в самом начале работы эмиттера

// собственно пока не накопится критическая масса частиц

while (particlesToCreate >= 1)

{

particlesToCreate -= 1;

var particle = CreateParticle();

ResetParticle(particle);

particles.Add(particle);

}

}

public int ParticlesCount = 500;

public virtual void ResetParticle(Particle particle)

{

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

particle.X = X;

particle.Y = Y;

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 Render(Graphics g)

{

// ну тут так и быть уж сам впишу...

// это то же самое что на форме в методе Render

foreach (var particle in particles)

{

particle.Draw(g);

}

foreach (var point in impactPoints) // тут теперь impactPoints

{

point.Render(g); // это добавили

}

}

public virtual Particle CreateParticle()

{

var particle = new ParticleColorful();

particle.FromColor = ColorFrom;

particle.ToColor = ColorTo;

return particle;

}

}

}

# 4 Код специальных точек

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static course\_work.Particle;

namespace course\_work

{

public abstract class IImpactPoint

{

public float X; // ну точка же, вот и две координаты

public float Y;

public Color NewFromColor;

public Color NewToColor;

// абстрактный метод с помощью которого будем изменять состояние частиц

// например притягивать

public abstract void ImpactParticle(Particle particle);

// базовый класс для отрисовки точечки

public virtual void Render(Graphics g)

{

g.FillEllipse(

new SolidBrush(Color.Red),

X - 5,

Y - 5,

10,

10

);

}

public class GravityPoint : IImpactPoint

{

public int Power = 100; // сила притяжения

// а сюда по сути скопировали с минимальными правками то что было в UpdateState

public override void ImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY); // считаем расстояние от центра точки до центра частицы

if (r + particle.Radius < Power / 2) // если частица оказалось внутри окружности

{

if (particle is ParticleColorful)

{

var p = (particle as ParticleColorful);

p.FromColor = NewFromColor;

p.ToColor = NewToColor;

}

}

}

public override void Render(Graphics g)

{

// буду рисовать окружность с диаметром равным Power

g.DrawEllipse(

new Pen(NewToColor),

X - Power / 2,

Y - Power / 2,

Power,

Power

);

var stringFormat = new StringFormat(); // создаем экземпляр класса

stringFormat.Alignment = StringAlignment.Center; // выравнивание по горизонтали

stringFormat.LineAlignment = StringAlignment.Center; // выравнивание по вертикали

g.DrawString(

$"Я гравитон\nc силой {Power}",

new Font("Verdana", 10),

new SolidBrush(NewToColor),

X,

Y,

stringFormat // передаем инфу о выравнивании

);

}

}

public class AntiGravityPoint : IImpactPoint

{

public int Power = 65; // сила отторжения

// а сюда по сути скопировали с минимальными правками то что было в UpdateState

public override void ImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

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

particle.SpeedX -= gX \* Power / r2; // тут минусики вместо плюсов

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

double r = Math.Sqrt(gX \* gX + gY \* gY); // считаем расстояние от центра точки до центра частицы

if (r + particle.Radius < Power / 2) // если частица оказалось внутри окружности

{

if (particle is ParticleColorful)

{

var p = (particle as ParticleColorful);

p.FromColor = NewFromColor;

p.ToColor = NewToColor;

}

}

}

public override void Render(Graphics g)

{

// буду рисовать окружность с диаметром равным Power

g.DrawEllipse(

new Pen(NewToColor),

X - Power / 2,

Y - Power / 2,

Power,

Power

);

var stringFormat = new StringFormat(); // создаем экземпляр класса

stringFormat.Alignment = StringAlignment.Center; // выравнивание по горизонтали

stringFormat.LineAlignment = StringAlignment.Center; // выравнивание по вертикали

g.DrawString(

$"Я антигравитон\nc силой {Power}",

new Font("Verdana", 10),

new SolidBrush(NewToColor),

X,

Y,

stringFormat // передаем инфу о выравнивании

);

}

}

public class DeathGravityPoint : IImpactPoint//шар уничтожающий эмиттеры

{

//public Action<IImpactPoint> onDeath;

public int Power = 150;

public int Counter = 0;//счёт для насыщенности цвета

public int Score = 0;//счет уничтоженных шаров

public override void ImpactParticle(Particle particle)

{

float gX = X - particle.X;

float gY = Y - particle.Y;

double r = Math.Sqrt(gX \* gX + gY \* gY); // считаем расстояние от центра точки до центра частицы

if (r + particle.Radius < Power / 2) // если частица оказалось внутри окружности

{

particle.Life = 0;

if (particle.Life <= 0)

{

Counter++;

Score++;

Red++;

if(Red >= 255)

{

Red = 255;

Green++;

if (Green >= 255)

{

Green = 255;

Blue++;

if(Blue >= 255)

{

Blue =255;

if(Blue >= 255 && Green >= 255 && Red >= 255)

{

Red = 0;

Green = 0;

Blue = 0;

}

}

}

}

}

}

}

public int Red, Green, Blue;//для изменения цвета

public override void Render(Graphics g)

{

if (Counter >= 235)

{

Counter = 255;

}

Color red = Color.FromArgb(Red, Green, Blue);//для изменения цвета

Color my = Color.FromArgb(Counter, red);

g.FillEllipse(new SolidBrush(my), X - Power / 2, Y - Power / 2, Power, Power);

g.DrawString(

$"", // надпись, можно перенос строки вставлять (если вы Катя, то может не работать и надо использовать \r\n)

new Font("Verdana", 10), // шрифт и его размер

new SolidBrush(Color.White), // цвет шрифта

X, // расположение в пространстве

Y

);

var stringFormat = new StringFormat(); // создаем экземпляр класса

stringFormat.Alignment = StringAlignment.Center; // выравнивание по горизонтали

stringFormat.LineAlignment = StringAlignment.Center; // выравнивание по вертикали

g.DrawString(

$"Уничтожаю шары\n{Score}",

new Font("Verdana", 10),

new SolidBrush(Color.Black),

X,

Y,

stringFormat // передаем инфу о выравнивании

);

}

}

}

}

# 5 Код частиц

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.ProgressBar;

namespace course\_work

{

public class Particle

{

public int Radius; // радиус частицы

public float X; // X координата положения частицы в пространстве

public float Y; // Y координата положения частицы в пространстве

public float SpeedX; // скорость перемещения по оси X

public float SpeedY; // скорость перемещения по оси Y

public float Life; // запас здоровья частицы

// добавили генератор случайных чисел

public static Random rand = new Random();

public bool SpeedVector = false;

// конструктор по умолчанию будет создавать кастомную частицу

public Particle()

{

// генерируем произвольное направление и скорость

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

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

// рассчитываем вектор скорости

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

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

// а это не трогаем

Radius = 2 + rand.Next(10);

Life = 20 + rand.Next(100);

}

public virtual void Draw(Graphics g)

{

// рассчитываем коэффициент прозрачности по шкале от 0 до 1.0

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

// рассчитываем значение альфа канала в шкале от 0 до 255

// по аналогии с RGB, он используется для задания прозрачности

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

// создаем цвет из уже существующего, но привязываем к нему еще и значение альфа канала

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

var b = new SolidBrush(color);

// остальное все так же

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

b.Dispose();

}

// новый класс для цветных частиц

public class ParticleColorful : Particle

{

// два новых поля под цвет начальный и конечный

public Color FromColor;

public Color ToColor;

// для смеси цветов

public static Color MixColor(Color color1, Color color2, float k)

{

return Color.FromArgb(

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

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

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

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

);

}

// ну и отрисовку перепишем

public override void Draw(Graphics g)

{

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

// так как k уменьшается 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);

if (SpeedVector)

{

g.DrawLine(new Pen(Color.White), X, Y, X + SpeedX \* 2, Y + SpeedY \* 2);

}

b.Dispose();

}

}

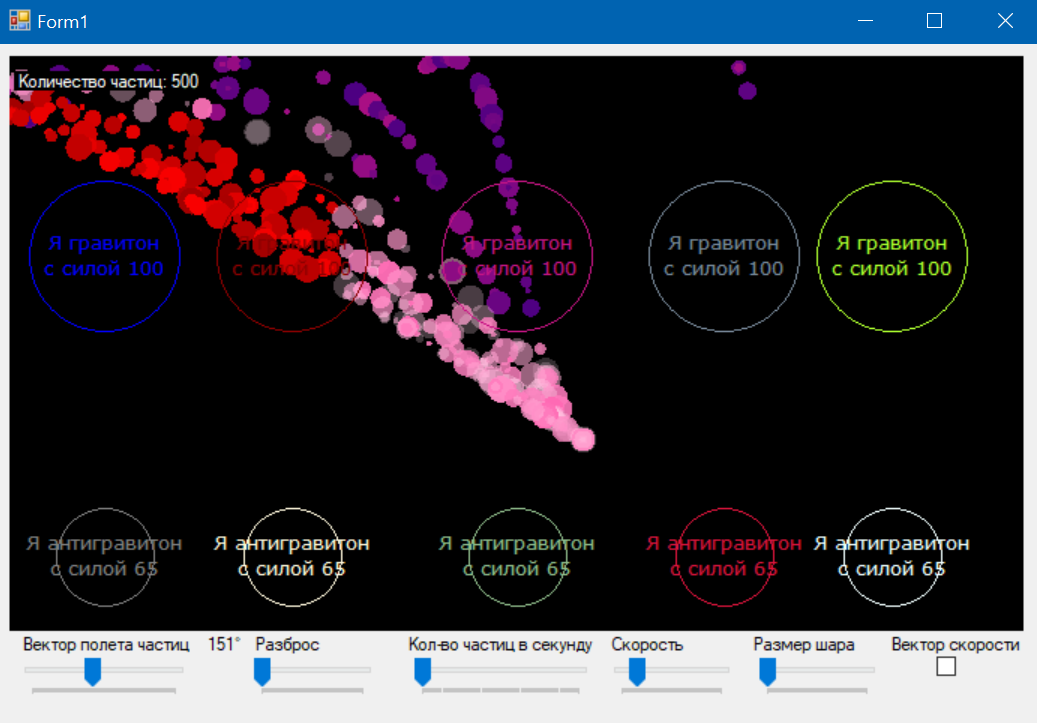
}

}

# 

# 6 Описание работы интерфейса

Меняя trackback **Вектор полёта частиц**(см рисунок 7.1) мы меняем направление в которое выстреливают новые частицы:

  
Рисунок 7.1

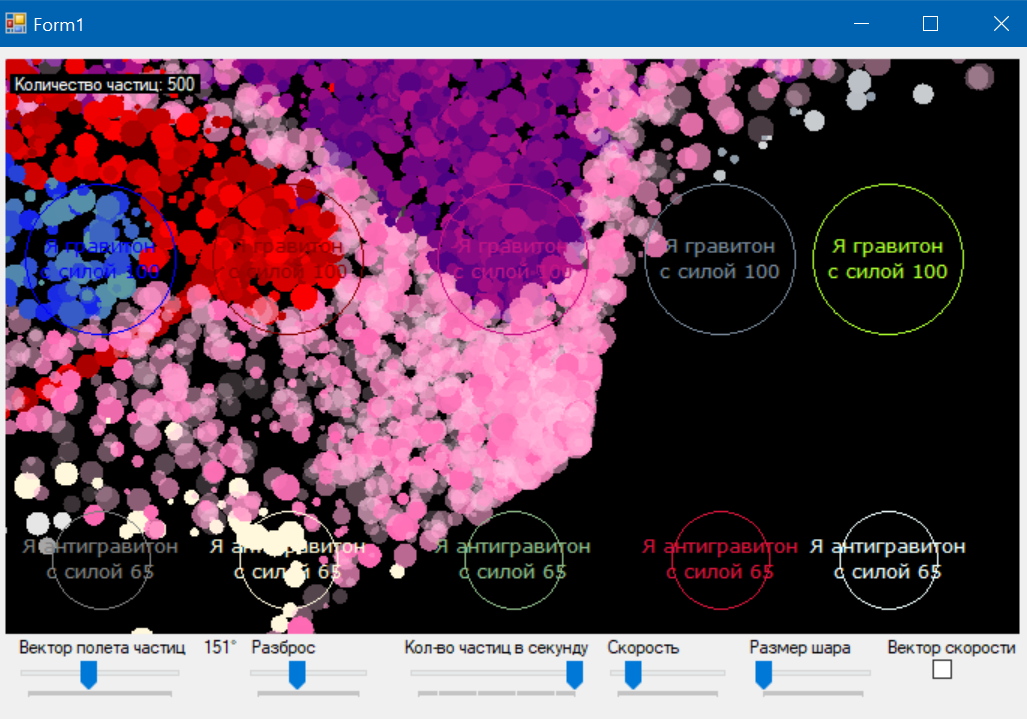
Меняя trackback **Разброс**(см рисунок 7.2) мы меняем распределение частиц около заданого направления при генерации:

Рисунок 7.2

Меняя trackbar **Количество частиц в секунду**(см рисунок 7.3),мы управляем количеством активных частиц в области генерации, при резком изменении частиц в больлшую сторону и с последующем:

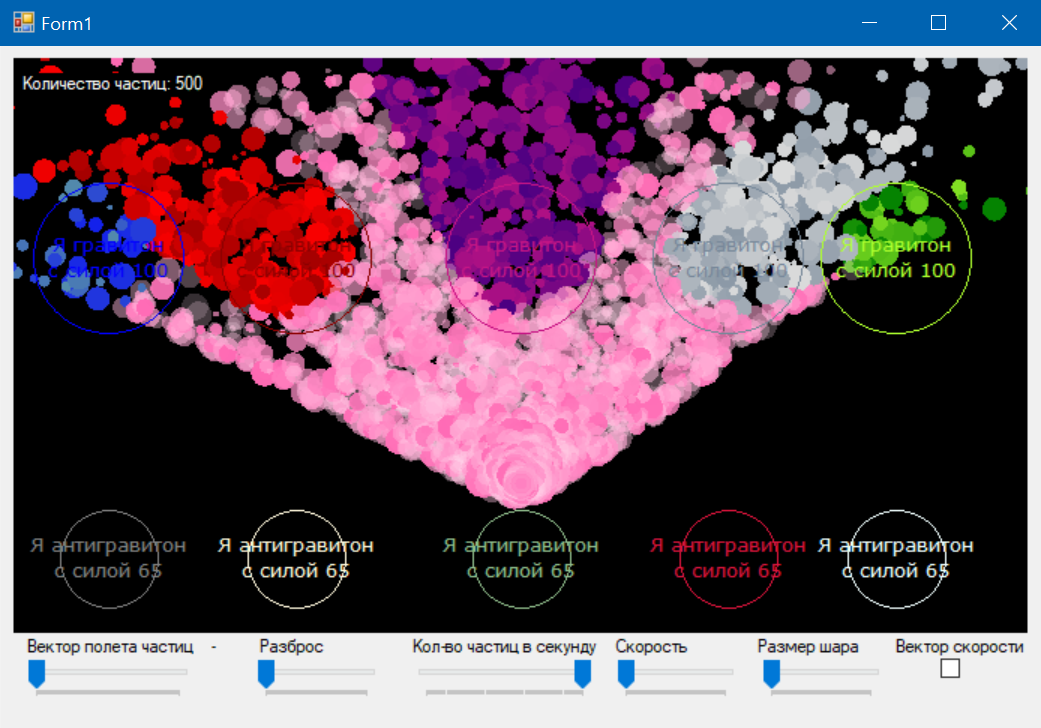


Рисунок 7.3

Частицы попадающие в область действия специальной точки уничтожаются и увеличивают счётчик изменяя тем самым цвет шара(см рисунок 7.4)

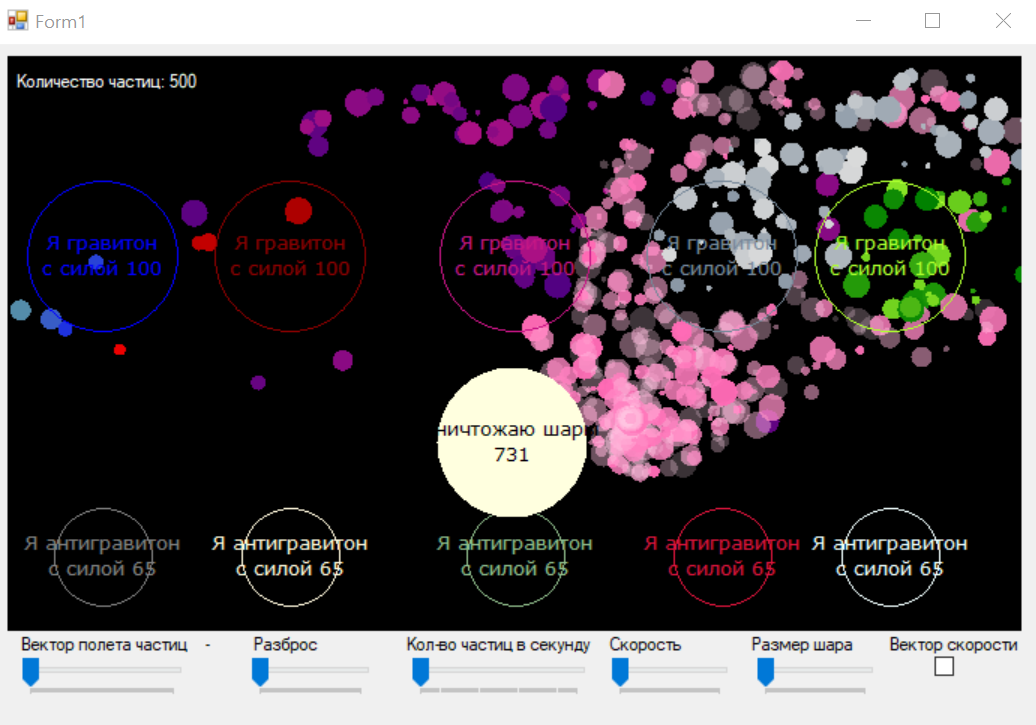


Рисунок 7.4

При нажатии кнопки **Вектор скорости**  (см рисунок 7.5) появляется вектор указывающий направление движения и этот вектор изменяет свою длину в зависимости от скорости



Рисунок 7.5

# Заключение

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

* Ползунок «Вектор полёта частиц», для управления направлением вылета частиц
* Ползунок «Разброс», для управления разбросом вылета частиц около заданного направления
* Ползунок «Количество частиц в секунду» - для управления количеством частицы на форме,
* Специальная точка «Убийца шаров» которая уничтожает частицы оказывающиеся в заданной области
* Ползунок изменяющий размер «Убийцы шаров»
* Кнопку вектора скорости

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] // aqua.tealeaf.su : чаинка, 2021. URL: http://aqua.tealeaf.su/particle-system.html (дата обращения: 01.12.2022).