|  |
| --- |
| ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «МЭИ»  Кафедра ВМСС  **ЛАБОРАТОРНАЯ РАБОТА**  **Тема: ООП в C#**  **Выполнил: Черкасов А. А.**  **А-08-19**  **Проверил: Раскатова М.В.**  Москва 2020 |

**Содержание**

[**Задание (8.1)** 2](#_Toc59181484)

[**Задание (8.2)** 2](#_Toc59181485)

[**1. Постановка задачи** 2](#_Toc59181486)

[**2. Разработка программы** 2](#_Toc59181487)

[2.1 Разработка функций и методов классов 2](#_Toc59181488)

[2.2 Разработка интерфейса 4](#_Toc59181489)

[**3. Реализация и тестирование программы** 5](#_Toc59181490)

[3.1 Описание разработанной программы 5](#_Toc59181491)

[3.2 Тестирование программы 8](#_Toc59181492)

[**Вывод** 10](#_Toc59181493)

[**Приложение. Код программы** 10](#_Toc59181494)

# **Задание (8.1)**

Часть I.

1. Построить иерархию классов в соответствии с вариантом задания.

2. Разработать методы и свойства для каждого из определяемых классов.

3. Реализовать программу на C# в соответствии с вариантом задания.

Вывести результаты

Часть II.

4. Изменить иерархию классов и реализовать ее на С#.

5. Показать на примере одного из методов, присутствующих в каждом классе, свойство

полиморфизма.

6. Расширить иерархию классов с использованием виртуального класса в качестве основы иерархии.

Показать пример использования полиморфизма методов.

Вывести результаты

Часть III.

7. Реализовать для иерархии (п.6) механизм интерфейсов, при этом один из классов должен

реализовывать как минимум два интерфейса. Использовать для проверки всех методов данного

класса многоадресный делегат.

# **Задание (8.2)**

С использованием методов объектно-ориентированного программирования разработать программу моделирования выбранной предметной области

# **1. Постановка задачи**

Разработать объектно-ориентрованную программу в соответствии с заданием в виде настольного приложения на C# средствами WindowsForms

Для темы Симуляция «атомного уровня» разработать соответствующую иерархию с механизмами интерфейсов и наследований.

Дать пользователю возможность влиять на симуляцию: добавление новых частиц, изменение «температурного коэффициента». Атомам дать возможность собираться в простейшие молекулы, а молекулам – реагировать друг с другом

Каждой частице дать «свободное поведение» - «шатание» из стороны в сторону. Создать аналог «закона Тяготения» для частиц.

# **2. Разработка программы**

2.1 Разработка функций и методов классов

Часть1:

Разработать систему, в соответствии с рисунком 2.1

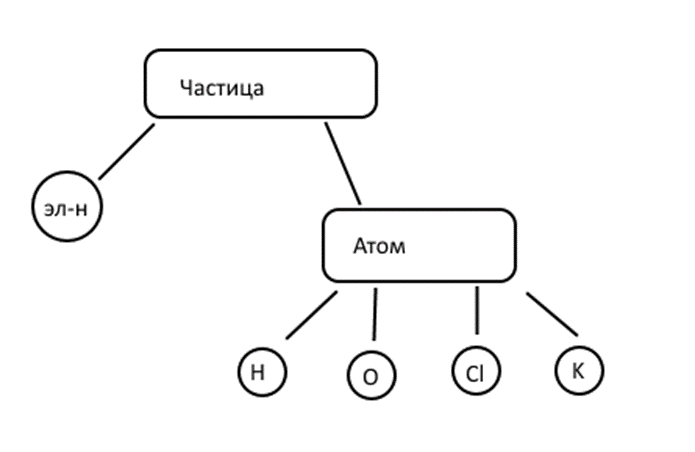
1

рис 2.1 Начальная иерархия классов

Каждому классу присвоить:

методы: Tick, Draw, Clear, SetCoords, GetCoords,

свойства: BasePoint, Width, Height

Атомам дополнительно присвоить метод: Wander, т.к. может двигаться сам по себе, а также свойство Color, для их цветового обозначения:

1. Водород – синий
2. Кислород – красный
3. Хлор – желтый
4. Калий – серый

Часть 2:

Изменить иерархию в соответствии с рис. 2.2

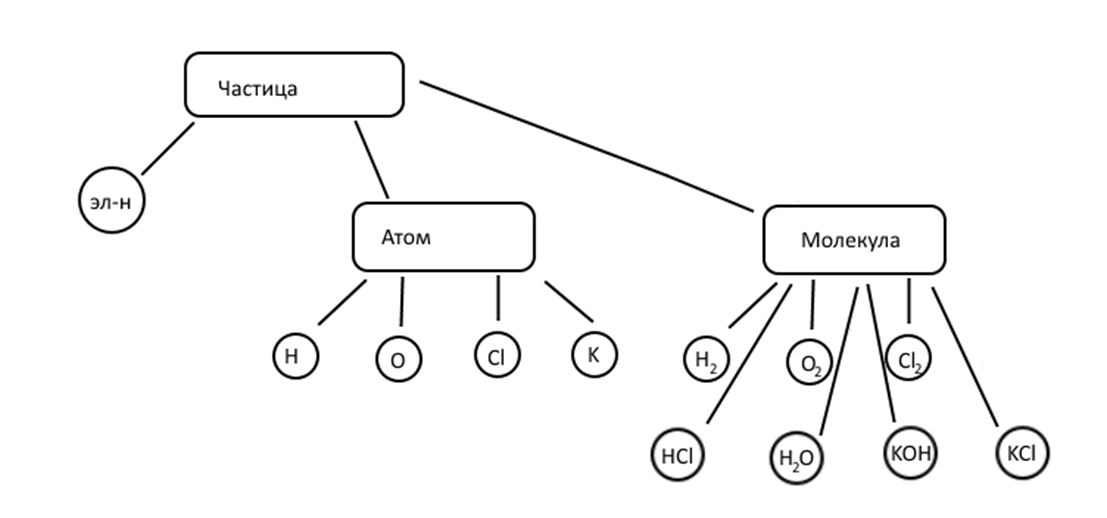
2

рис 2.2 Измененная иерархия классов

Часть 3:

Классы Частица, Атом, Молекула перевести в раздел интерфейсов (абстрактных классов).

Часть 4:

Финальная диаграмма классов выглядит так:



рис 2.3 диаграмма классов

Разработать подобие «закона Кулона» или «закона Тяготения» для частиц. Дать возможность молекулам реагировать:

1. H2+O2 = H2O + O
2. H2 + Cl2 =2HCl
3. HCl + K = KCl + H
4. H2O+KCl = H+ Cl + KOH

Для повышения производительности при большом количестве объектов на поле разработать класс дерево квадрантов для ограничения «области видимости» объектов.

2.2 Разработка интерфейса

Дать возможность пользователю наблюдать за происходящим в симуляции. Отрисовывать объекты на «поле» в соответствии с их проекцией на плоскость. Так как отрисовка электронов может сильно нагрузить систему, сделать CheckBox на «переключение» показа электронов. Реализовать возможность пользователя создавать новые частицы на месте курсора путем клика на «поле». Выделить отдельный CheckBox на отображение границ дерева квадрантов. Дать пользователю ползунок для регулирования «температурного коэффициента». Так как поле может забиться объектами и понизить framerate или просто мешать пользователю добавить кнопку «очистка», удаляющую все объекты на поле.

Примеры интерфейсов представлены на рис 2.4 и 2.5

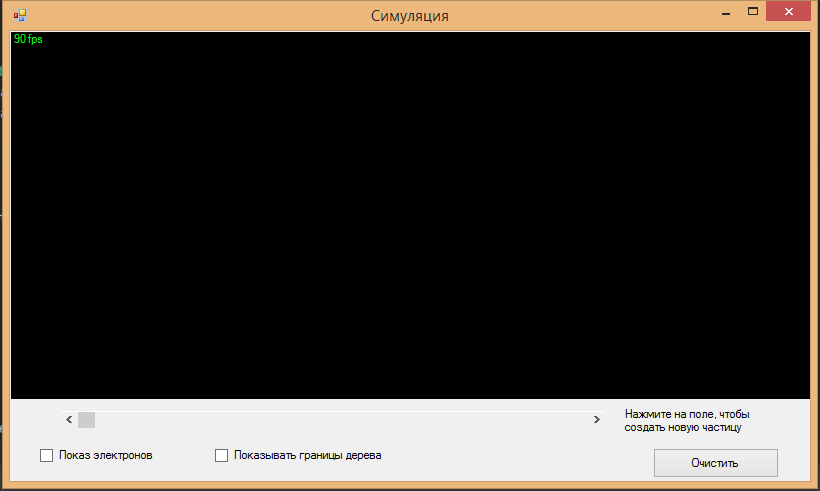


рис 2.4 основное окно

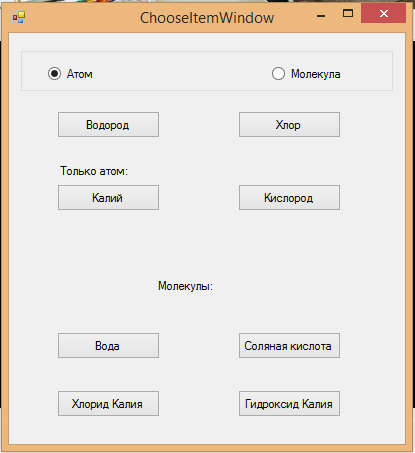


рис 2.5 выбор атома на добавление

# **3. Реализация и тестирование программы**

3.1 Описание разработанной программы

Система классов расположена в соответствии со своей смысловой нагрузкой в разных файлах

Утилитарные:

ThreeDim.cs

Для «упрощения» расчетов вращения электронов относительно атома вводим псевдотрехмерное измерение (основанное на изометрии). Для этого разработано:

Структуру TPoint:

Поля: double x, y, z – координаты точки.

Операторы +, -, \*,/

таблица 3.1 Класс TTT:

|  |  |
| --- | --- |
| **Методы** | **Описание** |
| public static Point Projection(TPoint tDim) | Проецирует заданную 3х мерную на плоскость |
| public static TPoint RotateZ(TPoint startVector,double angle) | Применение матрицы вращения для поворота «вектора» по оси Z |
| public static TPoint RotateX(TPoint startVector,double angle) | Применение матрицы вращения для поворота «вектора» по оси X |
| public static TPoint RotateY(TPoint startVector,double angle) | Применение матрицы вращения для поворота «вектора» по оси Y |
| public static double Radius(TPoint lhs, TPoint rhs) | Нахождение расстояния между двумя точками |

QuadTree.cs

Дерево квадрантов позволяет уменьшить время, затрачиваемое на «проведение реакций» путем прохождения не по всем существующим объектам (O(n2)), а только по ближайшим (O(nlogn)).

таблица 3.2 Класс QuadTree

|  |  |
| --- | --- |
| **Поля** | **Описание** |
| private QuadTree ul;  private QuadTree ur;  private QuadTree ll;  private QuadTree lr; | «Листья» дерева квадрантов. |
| private Rectangle \_boundaries; | Границы текущей ветви |
| private List<IParticle> \_elements; | «Содержание» данной ветви дерева. Пустое, данная ветвь имеет «листья» |
| private const int Capacity =4; | Вместимость ветви. После добавления следующего элемента, ветвь получит «листья» с раскиданными по ним содержимым данной ветви |
| **Конструктор** public QuadTree(Rectangle boundaries) | Задает границы данной ветви дерева, инициализация списка элементов |
| **Методы** |  |
| public void Append(IParticle element) | Добавление новой частицы в дерево |
| private void Split() | Вызывается при превышении вместимости. Создает листья данной ветви и распределяет элементы по ним |
| private void AddToLower(IParticle element) | «Логика» добавления элемента. Выбирает в какой из листьев отправить элемент |
| public void DrawContents(Graphics gr, int drawstate) | Отрисовка элементов на поле |
| public void ClearContents(Graphics gr, int drawstate) |
| public void Tick(double coeff) | Логика элементы |
| public void DrawBoundaries(Graphics gr, Color color) | Отрисовка границ дерева |
| public void GetContents(List<IParticle> array) | Получение содержимого дерева |
|  | Таблица 3.2 продолжение |
| public void GetContentsLimited(List<IParticle> array, Rectangle boundaries) | Получение содержимого дерева в границах прямоугольника |

Основные:

IParticle.cs

«Основа» всей системы, а также класс электрона

Таблица 3.3 Абстрактный класс IParticle

|  |  |
| --- | --- |
| **Поля** | **Описание** |
| protected TPoint BasePoint; | Базовая точка объекта |
| protected int Width;  protected int Height; | Размеры объекта |
| **Методы** |  |
| public abstract void Tick(double Coeff) | Логика за кадр |
| public abstract void Draw(Graphics e,int s) | Отрисовка |
| public abstract void Clear(Graphics e, int s) |
| public void SetCoords(double x, double y, double z) | Установление новых координат объекта |
| public TPoint GetCoords() | Получение координат объекта |

Таблица 3.4 Класс Electron: IParticle

|  |  |
| --- | --- |
| **Поля** | **Описание** |
| private double \_baseRadSpeed; | Базовая скорость вращения электрона |
| private double \_angle; | Угол поворота электрона |
| private Point \_location; | Текущее положение электрона |
| private TPoint \_radius; | Радиус вектор относительно базовой точки |
| private readonly int \_level; | «Уровень» электрона |
| private delegate TPoint Rotation(TPoint vector, double angle);  private Rotation \_rotate; | Функция вращения электрона. В зависимости от уровня вращается относительно своей оси |
| **Контруктор** public Electron(double x, double y, double z,int level = 1,double angle =0) | Устанавливает начальные координаты точки, угол и уровень |
| **Методы** |  |
| SetParameters() | Устанавливает базовые параметры частицы |
| public override void Clear(Graphics e, int state) | Отрисовка электрона |
| public override void Draw(Graphics e,int state) |
| public void Tick(double coeff) | Логика элемента |

IAtom.cs

«Основа», вместе с атомами

Таблица 3.5 Абстрактный класс IAtom: IParticle

|  |  |
| --- | --- |
| **Поля** | **Описание** |
| protected Electron[] Electrons; | Массив содержащихся электронов |
| protected Point Location; | Текущее положение |
| protected Color AtomColor; | Цветовое обозначение |
| **Методы** |  |
| public override void Tick(double Coeff) | Логика за кадр |
| public override void Draw(Graphics e,int s) | Отрисовка |
| public override void Clear(Graphics e, int s) |
| private void Wander(double x, double y, double z) | "Шевеление" атома |

В классах самих атомов AHydrogen: IAtom, AOxygen: IAtom, AChlorine: IAtom, APotassium: IAtom определен только конструктор с SetParameters() для инициализации объектов.

IMolecule.cs

«Основа», вместе с молекулами

Таблица 3.6 Абстрактный класс IMolecule:IParticle

|  |  |
| --- | --- |
| **Поля** | **Описание** |
| protected IAtom[] ContainedAtoms; | Массив атомов |
| **Методы** |  |
| SetParameters() | Устанавливает базовые параметры частицы |
| public override void Clear(Graphics e, int state) | Отрисовка электрона |
| public override void Draw(Graphics e,int state) |
| protected void DrawLineBetweenAtoms(Graphics e, Color color) | Линия между атомами для указания связи |
| public void Tick(double coeff) | Логика элемента |
| private void Wander(double x, double y, double z) | "Шевеление" молекулы |

В классах самих молекул MHydrogen : IMolecule, MOxygen : IMolecule, MChlorine : IMolecule, MHCl : IMolecule, MKCl : IMolecule, MKOH : IMolecule, MHOH : IMolecule определены конструкторы и методы SetParameters() и SetAtoms() для инициализации объектов

3.2 Тестирование программы

Объект тестирования – разработанная система классов

Цель тестирования – проверить работоспособность разработанных функций

Средства испытаний – Среда Rider

Методы испытаний – функциональное тестирование (тестирование по входу)

Тестирование проводилось на нескольких наборах данных – в нормальных, граничных и исключительных условиях. Результаты тестирования приведены в таблице 3.6

Для того, чтобы хоть как-то можно было отобразить передвижение частиц на некоторых тестах, правлена отрисовка элементов, чтобы оставался след.

Таблица 3.6 тесты

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Смысл теста** | **Входные данные** | **Результат** |
| **1** | Проверка всех элементов |  | |
|  |  |  | |
|  |  | Таблица 3.6 продолжение | |
| **2** | Проверка вращения электронов | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T2.1.png | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T2.2.png |
| **3** | Проверка притяжения частиц | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T3.1.png | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T3.2.png |
| **4** | Проверка объединения атомов в простейшие молекулы | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T4.1.png | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T4.2.png |
| **5** | Проверка построения дерева | **D:\TEMP_USR\Временные файлы Интернета\Content.Word\T5.1.png** | **D:\TEMP_USR\Временные файлы Интернета\Content.Word\T5.2.png** |
| **6** | Проверка реакций молекул | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T6.1.png | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T6.2.png |
| **7** | Проверка производительности | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T7.1.png | D:\TEMP_USR\Временные файлы Интернета\Content.Word\T7.2.png |

# **Вывод**

Протестированные функции работают по поставленной задаче.

Разобрался с ООП в C#. Узнал, как работают принципы ООП в C#

# **Приложение. Код программы**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace IndividualTask

{

static class Program

{

public static Thread t;

public static Form1 f;

public static Random r;

[STAThread]

private static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

r = new Random();

f = new Form1();

t = new Thread(f.UpdateContains);

t.Start();

Application.Run(f);

t.Abort();

}

}

}

**IParticle.cs**

using System;

using System.Drawing;

using TDim;

namespace IndividualTask

{

//Основа для всех частиц

public abstract class IParticle

{

//Функция обновляющая состояние объекта

public abstract void Tick(double Coeff);

//Отрисовка объекта

public abstract void Draw(Graphics e,int s);

//"Чистка" предыдущего положения объекта

public abstract void Clear(Graphics e, int s);

//Установление новых координат объекта

public void SetCoords(double x, double y, double z)

{

BasePoint= new TPoint(x,y,z);

}

//Установление новых координат объекта

public void SetCoords(TPoint p)

{

BasePoint= p;

}

//Получение координат объекта

public TPoint GetCoords()

{

return BasePoint;

}

#region Parameters

protected TPoint BasePoint;

protected int Width;

protected int Height;

#endregion

}

//Простейшая частица - электрон

public class Electron : IParticle

{

public Electron()

{

\_level = 1;

SetParameters();

}

public Electron(double x, double y, double z,int level = 1,double angle =0)

{

\_level = level;

SetParameters();

BasePoint = new TPoint(x,y,z);

\_angle = angle;

}

//Параметры электрона

private void SetParameters()

{

Width = 1;

Height = 1;

switch (\_level)

{

case 1:

\_radius = new TPoint(20,0,0);

\_rotate = TTT.RotateZ;

break;

case 2:

\_radius = new TPoint(0,30,0);

\_rotate = TTT.RotateX;

break;

case 3:

\_radius = new TPoint(0,0,40);

\_rotate = TTT.RotateY;

break;

}

\_angle = 0;

BasePoint = new TPoint(0,0,0);

\_baseRadSpeed = Math.PI / 180;

}

public override void Tick(double Coeff)

{

\_angle += \_baseRadSpeed \* Coeff;

if (\_angle >= Math.PI \* 2) \_angle = 0;

\_location = TTT.Projection(BasePoint + \_rotate(\_radius, \_angle));

}

public override void Clear(Graphics e, int state)

{

e.DrawRectangle(new Pen(Color.Black), \_location.X, \_location.Y, 1, 1);

}

public override void Draw(Graphics e,int state)

{

e.DrawRectangle(new Pen(Color.White), \_location.X, \_location.Y, 1, 1);

}

#region AdditionalParameters

private double \_baseRadSpeed;

private double \_angle;

private Point \_location;

private TPoint \_radius;

private readonly int \_level;

private delegate TPoint Rotation(TPoint vector, double angle);

private Rotation \_rotate;

#endregion

}

}

**IAtom.cs**

using System;

using System.Drawing;

using TDim;

namespace IndividualTask

{

//База для атомов, основана на частица

public abstract class IAtom: IParticle

{

#region AdditionalParameters

protected Electron[] Electrons;

protected Point Location;

protected Color AtomColor;

#endregion

public override void Tick(double Coeff)

{

foreach (var electron in Electrons)

{

electron.Tick(Coeff);

}

Wander(Coeff);

}

//"Шевеление" атома

private void Wander(double coeff)

{

var speed = new TPoint(Program.r.NextDouble()+(Program.r.Next()%3-1),Program.r.NextDouble()+(Program.r.Next()%3-1),Program.r.NextDouble()+(Program.r.Next()%3-1));

BasePoint += speed\*coeff;

Location = TTT.Projection(BasePoint);

//Проверка на выходы за пределы поля

if (Location.X>=802)

{

BasePoint= new TPoint(BasePoint.x-3,BasePoint.y,BasePoint.z);

Location = TTT.Projection(BasePoint);

}

if (Location.X<=0)

{

BasePoint= new TPoint(BasePoint.x,BasePoint.y-3,BasePoint.z);

Location = TTT.Projection(BasePoint);

}

if (Location.Y>=367)

{

BasePoint= new TPoint(BasePoint.x,BasePoint.y,BasePoint.z+3);

Location = TTT.Projection(BasePoint);

}

if (Location.Y<=0)

{

BasePoint= new TPoint(BasePoint.x,BasePoint.y,BasePoint.z-3);

Location = TTT.Projection(BasePoint);

}

foreach (var electron in Electrons)

{

electron.SetCoords(BasePoint);

}

}

public override void Clear(Graphics e,int state)

{

if((state&1)==1) foreach (var el in Electrons) el.Clear(e,state);

var pr = new Rectangle(Location.X,Location.Y,Width,Height);

e.DrawEllipse(new Pen(Color.Black), pr);

e.FillEllipse(new SolidBrush(Color.Black), pr);

}

public override void Draw(Graphics e,int state)

{

if((state&1)==1) foreach (var el in Electrons) el.Draw(e,state);

var r = new Rectangle(Location.X,Location.Y,Width,Height);

e.DrawEllipse(new Pen(AtomColor), r);

e.FillEllipse(new SolidBrush(AtomColor), r);

}

}

//Сами электроны

public class AHydrogen : IAtom

{

public AHydrogen()

{

SetParameters();

}

public AHydrogen(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

}

private void SetParameters()

{

Width = 4;

Height = 4;

BasePoint = new TPoint(0,0,0);

Location = TTT.Projection(BasePoint);

Electrons = new Electron[1];

Electrons[0] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z);

AtomColor = Color.Blue;

}

}

public class AOxygen : IAtom

{

public AOxygen()

{

SetParameters();

}

public AOxygen(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

}

private void SetParameters()

{

Width = 4;

Height = 4;

BasePoint = new TPoint(0,0,0);

Location = TTT.Projection(BasePoint);

Electrons = new Electron[8];

AtomColor = Color.Red;

for (var i = 0; i < 8; i++)

{

if (i < 2)

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,1,Math.PI\*i);

}

else

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,2,2\*Math.PI/6\*(i-2));

}

}

}

}

public class AChlorine : IAtom

{

public AChlorine()

{

SetParameters();

}

public AChlorine(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

}

private void SetParameters()

{

Width = 4;

Height = 4;

BasePoint = new TPoint(0,0,0);

Location = TTT.Projection(BasePoint);

Electrons = new Electron[17];

AtomColor = Color.Yellow;

for (var i = 0; i < 17; i++)

{

if (i < 2)

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,1,Math.PI\*i);

}

else if (i<10)

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,2,2\*Math.PI/8\*(i-2));

}

else

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,3,2\*Math.PI/7\*(i-10));

}

}

}

}

public class APotassium : IAtom

{

public APotassium()

{

SetParameters();

}

public APotassium(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

}

private void SetParameters()

{

Width = 4;

Height = 4;

BasePoint = new TPoint(0,0,0);

Location = TTT.Projection(BasePoint);

Electrons = new Electron[19];

AtomColor = Color.Gray;

for (var i = 0; i < 19; i++)

{

if (i < 2)

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,1,Math.PI\*i);

}

else if (i<10)

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,2,2\*Math.PI/8\*(i-2));

}

else

{

Electrons[i] = new Electron(BasePoint.x,BasePoint.y,BasePoint.z,3,2\*Math.PI/9\*(i-10));

}

}

}

}

}

**IMolecule.cs**

using System.Drawing;

using TDim;

namespace IndividualTask

{

//Основа для всех молкул

public abstract class IMolecule:IParticle

{

public override void Tick(double Coeff)

{

Wander(Coeff);

foreach (var atom in ContainedAtoms)

{

atom.Tick(Coeff);

var atomLocation = atom.GetCoords();

if (TTT.Radius(atomLocation, BasePoint) > 8) atom.SetCoords(atomLocation - (atomLocation-BasePoint)/8);

if (TTT.Radius(atomLocation, BasePoint) < 2) atom.SetCoords(atomLocation + (atomLocation-BasePoint)\*1.5);

}

}

public override void Clear(Graphics e, int state)

{

DrawLineBetweenAtoms(e,Color.Black);

foreach (var atom in ContainedAtoms)

{

atom.Clear(e,state);

}

}

public override void Draw(Graphics e, int state)

{

DrawLineBetweenAtoms(e,Color.White);

foreach (var atom in ContainedAtoms)

{

atom.Draw(e,state);

}

}

//Линия, "соединяющая" атомы в молекулы

protected void DrawLineBetweenAtoms(Graphics e, Color color)

{

for (int i = 0; i < ContainedAtoms.Length-1; i++)

{

var fLocation = TDim.TTT.Projection(ContainedAtoms[i].GetCoords());

fLocation = new Point(fLocation.X+3,fLocation.Y+2);

var sLocation = TDim.TTT.Projection(ContainedAtoms[i+1].GetCoords());

sLocation = new Point(sLocation.X,sLocation.Y+2);

e.DrawLine(new Pen(color),fLocation,sLocation);

}

}

private void Wander(double coeff)

{

var speed = new TPoint(Program.r.NextDouble()+(Program.r.Next()%3-1),Program.r.NextDouble()+(Program.r.Next()%3-1),Program.r.NextDouble()+(Program.r.Next()%3-1));

BasePoint += speed\*coeff;

foreach (var atom in ContainedAtoms)

{

atom.SetCoords(atom.GetCoords()+speed\*coeff);

}

var location = TTT.Projection(BasePoint);

if (location.X>=802)

BasePoint= new TPoint(BasePoint.x-3,BasePoint.y,BasePoint.z);

if (location.X<=0)

BasePoint= new TPoint(BasePoint.x,BasePoint.y-3,BasePoint.z);

if (location.Y>=367)

BasePoint= new TPoint(BasePoint.x,BasePoint.y,BasePoint.z+3);

if (location.Y<=0)

BasePoint= new TPoint(BasePoint.x,BasePoint.y,BasePoint.z-3);

}

#region AdditonalParameters

protected IAtom[] ContainedAtoms;

#endregion

}

//Молекулы

public class MHydrogen : IMolecule

{

public MHydrogen()

{

SetParameters();

SetAtoms();

}

public MHydrogen(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[2];

ContainedAtoms[0]= new AHydrogen(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AHydrogen(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

public class MOxygen : IMolecule

{

public MOxygen()

{

SetParameters();

SetAtoms();

}

public MOxygen(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[2];

ContainedAtoms[0]= new AOxygen(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AOxygen(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

public class MChlorine : IMolecule

{

public MChlorine()

{

SetParameters();

SetAtoms();

}

public MChlorine(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[2];

ContainedAtoms[0]= new AChlorine(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AChlorine(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

public class MHCl : IMolecule

{

public MHCl()

{

SetParameters();

SetAtoms();

}

public MHCl(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[2];

ContainedAtoms[0]= new AHydrogen(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AChlorine(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

public class MKCl : IMolecule

{

public MKCl()

{

SetParameters();

SetAtoms();

}

public MKCl(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[2];

ContainedAtoms[0]= new APotassium(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AChlorine(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

public class MKOH : IMolecule

{

public MKOH()

{

SetParameters();

SetAtoms();

}

public MKOH(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[3];

ContainedAtoms[0]= new APotassium(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AOxygen(BasePoint.x,BasePoint.y,BasePoint.z);

ContainedAtoms[2]= new AHydrogen(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

public class MHOH : IMolecule

{

public MHOH()

{

SetParameters();

SetAtoms();

}

public MHOH(double x, double y, double z)

{

SetParameters();

BasePoint = new TPoint(x,y,z);

SetAtoms();

}

private void SetParameters()

{

BasePoint = new TPoint(0,0,0);

}

private void SetAtoms()

{

ContainedAtoms = new IAtom[3];

ContainedAtoms[0]= new AHydrogen(BasePoint.x-10,BasePoint.y,BasePoint.z);

ContainedAtoms[1]= new AOxygen(BasePoint.x,BasePoint.y,BasePoint.z);

ContainedAtoms[2]= new AHydrogen(BasePoint.x,BasePoint.y-10,BasePoint.z);

}

}

}

**MainWindow.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Threading;

using System.Windows.Forms;

using IndTsk;

using TDim;

using Tree;

namespace IndividualTask

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

Additional();

}

private void Additional()

{

\_tree = new QuadTree(new Rectangle(0, 0, field.Width, field.Height));

\_graphics = field.CreateGraphics();

\_drawState = 0;

\_tempCoeff = 0;

\_update = true;

NewParticle = null;

field.MouseClick += field\_Click;

}

//Функция на обновление и отрисовку частиц

public void UpdateContains()

{

while (\_update)

{

var tStart = DateTime.Now;

var drawBoundaries = (\_drawState & 2) == 2;

if (drawBoundaries) \_tree.DrawBoundaries(\_graphics, Color.Black);

RebuildTree();

\_tree.ClearContents(\_graphics,\_drawState);

if (drawBoundaries) \_tree.DrawBoundaries(\_graphics, Color.Aquamarine);

\_tree.Tick(\_tempCoeff);

\_tree.DrawContents(\_graphics, \_drawState);

Thread.Sleep(10);

var tPassed = DateTime.Now.Subtract(tStart);

if (tPassed.TotalSeconds!=0) FPSLabel.Text = (int)(1.0 / tPassed.TotalSeconds) + " FPS";

}

}

private int \_drawState;

private delegate void coveredtsk();

private bool \_update;

private QuadTree \_tree;

private Graphics \_graphics;

private double \_tempCoeff;

public IParticle NewParticle;

//Рисовать электроны или нет

private void ChangeDrawingState()

{

if (!showElectronsCheckBox.Checked && (\_drawState & 1) == 1)

{

\_tree.ClearContents(\_graphics, \_drawState);

\_drawState -= 1;

}

if (showElectronsCheckBox.Checked) \_drawState |= 1;

if (!changeDrawTreeBox.Checked && (\_drawState & 2) == 2)

{

\_tree.DrawBoundaries(\_graphics, Color.Black);

\_drawState -= 2;

}

if (changeDrawTreeBox.Checked)

{

\_drawState |= 0b10;

}

}

//Добавление частиц

private static void Run(object o)

{

Application.Run((Form) o);

}

private void AddParticleCoords(TPoint coords)

{

var choose = new ChooseItemWindow();

var t = new Thread(new ParameterizedThreadStart(Run));

t.Start(choose);

t.Join();

if (NewParticle != null)

{

DebugLabel.Text = "Создано:" + NewParticle.GetType().ToString().Split('.')[1];

NewParticle.SetCoords(coords);

\_tree.Append(NewParticle);

}

NewParticle = null;

}

//Вызов нужных функций

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

DoWithStoppingThread(ChangeDrawingState);

}

private void field\_Click(object sender, MouseEventArgs e)

{

\_update = false;

Program.t.Join();

var particleCoords = new TPoint(((e.X-field.Width/2)/Math.Cos(Math.PI/6)),0,-e.Y+field.Height/2+(e.X-field.Width/2)\*Math.Cos(Math.PI/6));

AddParticleCoords(particleCoords);

\_update = true;

Program.t = new Thread(UpdateContains);

Program.t.Start();

}

//Обложка на приостановку потока и выполнение самой функции

private void DoWithStoppingThread(coveredtsk tsk)

{

\_update = false;

Program.t.Join();

tsk();

\_update = true;

Program.t = new Thread(UpdateContains);

Program.t.Start();

}

//Дерево необходимо перестраивать, т.к. элементы меняют свою позицию

private void RebuildTree()

{

var newTree = new QuadTree(new Rectangle(0, 0, field.Width, field.Height));

var contents = new List<IParticle>();

\_tree.GetContents(contents);

ApplyForce(contents);

SetReactions(contents);

foreach (var element in contents) newTree.Append(element);

\_tree = newTree;

}

private void ClearAll()

{

\_tree.DrawBoundaries(\_graphics,Color.Black);

\_tree.ClearContents(\_graphics,\_drawState);

\_tree = new QuadTree(new Rectangle(0, 0, field.Width, field.Height));

}

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

{

DoWithStoppingThread(ClearAll);

DebugLabel.Visible = true;

DebugLabel.Text = "";

}

private void TemperatureControl\_Scroll(object sender, ScrollEventArgs e)

{

\_tempCoeff = TempertureControl.Value / 10.0;

DebugLabel.Text = \_tempCoeff.ToString();

}

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

{

DebugLabel.Visible = false;

}

}

}

**MainWindow.Designer.cs**

namespace IndividualTask

{

partial class Form1

{

/// <summary>

/// Required designer variable.

/// </summary>

private System.ComponentModel.IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.field = new System.Windows.Forms.Panel();

this.FPSLabel = new System.Windows.Forms.Label();

this.TempertureControl = new System.Windows.Forms.HScrollBar();

this.showElectronsCheckBox = new System.Windows.Forms.CheckBox();

this.changeDrawTreeBox = new System.Windows.Forms.CheckBox();

this.DebugLabel = new System.Windows.Forms.Label();

this.clearButton = new System.Windows.Forms.Button();

this.label1 = new System.Windows.Forms.Label();

this.field.SuspendLayout();

this.SuspendLayout();

//

// field

//

this.field.BackColor = System.Drawing.Color.Black;

this.field.Controls.Add(this.FPSLabel);

this.field.Location = new System.Drawing.Point(1, 1);

this.field.Name = "field";

this.field.Size = new System.Drawing.Size(802, 367);

this.field.TabIndex = 2;

//

// FPSLabel

//

this.FPSLabel.ForeColor = System.Drawing.Color.Lime;

this.FPSLabel.Location = new System.Drawing.Point(0, 0);

this.FPSLabel.Name = "FPSLabel";

this.FPSLabel.Size = new System.Drawing.Size(77, 18);

this.FPSLabel.TabIndex = 0;

this.FPSLabel.Text = "fps";

//

// TempertureControl

//

this.TempertureControl.LargeChange = 1;

this.TempertureControl.Location = new System.Drawing.Point(51, 380);

this.TempertureControl.Name = "TempertureControl";

this.TempertureControl.Size = new System.Drawing.Size(544, 18);

this.TempertureControl.TabIndex = 3;

this.TempertureControl.Scroll += new System.Windows.Forms.ScrollEventHandler(this.TemperatureControl\_Scroll);

//

// showElectronsCheckBox

//

this.showElectronsCheckBox.Location = new System.Drawing.Point(30, 414);

this.showElectronsCheckBox.Name = "showElectronsCheckBox";

this.showElectronsCheckBox.Size = new System.Drawing.Size(179, 22);

this.showElectronsCheckBox.TabIndex = 4;

this.showElectronsCheckBox.Text = "Показ электронов";

this.showElectronsCheckBox.UseVisualStyleBackColor = true;

this.showElectronsCheckBox.CheckedChanged += new System.EventHandler(this.checkBox1\_CheckedChanged);

//

// changeDrawTreeBox

//

this.changeDrawTreeBox.Location = new System.Drawing.Point(205, 414);

this.changeDrawTreeBox.Name = "changeDrawTreeBox";

this.changeDrawTreeBox.Size = new System.Drawing.Size(179, 22);

this.changeDrawTreeBox.TabIndex = 4;

this.changeDrawTreeBox.Text = "Показывать границы дерева";

this.changeDrawTreeBox.UseVisualStyleBackColor = true;

this.changeDrawTreeBox.CheckedChanged += new System.EventHandler(this.checkBox1\_CheckedChanged);

//

// DebugLabel

//

this.DebugLabel.Location = new System.Drawing.Point(423, 414);

this.DebugLabel.Name = "DebugLabel";

this.DebugLabel.Size = new System.Drawing.Size(195, 21);

this.DebugLabel.TabIndex = 5;

this.DebugLabel.Click += new System.EventHandler(this.DebugLabel\_Click);

//

// clearButton

//

this.clearButton.Location = new System.Drawing.Point(643, 417);

this.clearButton.Name = "clearButton";

this.clearButton.Size = new System.Drawing.Size(126, 30);

this.clearButton.TabIndex = 6;

this.clearButton.Text = "Очистить";

this.clearButton.UseVisualStyleBackColor = true;

this.clearButton.Click += new System.EventHandler(this.clearButton\_Click);

//

// label1

//

this.label1.Location = new System.Drawing.Point(612, 376);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(171, 38);

this.label1.TabIndex = 7;

this.label1.Text = "Нажмите на поле, чтобы создать новую частицу";

//

// Form1

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.BackColor = System.Drawing.SystemColors.Control;

this.ClientSize = new System.Drawing.Size(800, 450);

this.Controls.Add(this.label1);

this.Controls.Add(this.clearButton);

this.Controls.Add(this.DebugLabel);

this.Controls.Add(this.changeDrawTreeBox);

this.Controls.Add(this.showElectronsCheckBox);

this.Controls.Add(this.TempertureControl);

this.Controls.Add(this.field);

this.Name = "Form1";

this.Text = "Симуляция";

this.field.ResumeLayout(false);

this.ResumeLayout(false);

}

private System.Windows.Forms.Label FPSLabel;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.Button clearButton;

private System.Windows.Forms.CheckBox changeDrawTreeBox;

private System.Windows.Forms.Label DebugLabel;

private System.Windows.Forms.CheckBox showElectronsCheckBox;

private System.Windows.Forms.HScrollBar TempertureControl;

private System.Windows.Forms.Panel field;

#endregion

}

}

**ChooseItemWindow.cs**

using System;

using System.Windows.Forms;

using IndividualTask;

using TDim;

namespace IndTsk

{

public partial class ChooseItemWindow : Form

{

public ChooseItemWindow()

{

InitializeComponent();

}

//Создание нового объекта на основе выбора пользователя

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

{

if (\_isMolecule) Program.f.NewParticle = new MHydrogen();

else Program.f.NewParticle = new AHydrogen();

Close();

}

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

{

if (\_isMolecule) Program.f.NewParticle = new MChlorine();

else Program.f.NewParticle = new AChlorine();

Close();

}

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

{

if (\_isMolecule) Program.f.NewParticle = new MOxygen();

else Program.f.NewParticle = new AOxygen();

Close();

}

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

{

Program.f.NewParticle = new APotassium();

Close();

}

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

{

Program.f.NewParticle = new MHOH();

Close();

}

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

{

Program.f.NewParticle = new MHCl();

Close();

}

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

{

Program.f.NewParticle = new MKCl();

Close();

}

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

{

Program.f.NewParticle = new MKOH();

Close();

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

\_isMolecule = !\_isMolecule;

}

private bool \_isMolecule = false;

}

}

**ChooseItemWindow.Designer.cs**

using System.ComponentModel;

namespace IndTsk

{

partial class ChooseItemWindow

{

/// <summary>

/// Required designer variable.

/// </summary>

private IContainer components = null;

/// <summary>

/// Clean up any resources being used.

/// </summary>

/// <param name="disposing">true if managed resources should be disposed; otherwise, false.</param>

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

#region Windows Form Designer generated code

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.groupBox1 = new System.Windows.Forms.GroupBox();

this.radioButton2 = new System.Windows.Forms.RadioButton();

this.radioButton1 = new System.Windows.Forms.RadioButton();

this.buttonH = new System.Windows.Forms.Button();

this.buttonO = new System.Windows.Forms.Button();

this.buttonCl = new System.Windows.Forms.Button();

this.buttonK = new System.Windows.Forms.Button();

this.buttonW = new System.Windows.Forms.Button();

this.buttonKCl = new System.Windows.Forms.Button();

this.buttonHCl = new System.Windows.Forms.Button();

this.buttonKOH = new System.Windows.Forms.Button();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.groupBox1.SuspendLayout();

this.SuspendLayout();

//

// groupBox1

//

this.groupBox1.Controls.Add(this.radioButton2);

this.groupBox1.Controls.Add(this.radioButton1);

this.groupBox1.Location = new System.Drawing.Point(12, 12);

this.groupBox1.Name = "groupBox1";

this.groupBox1.Size = new System.Drawing.Size(372, 46);

this.groupBox1.TabIndex = 0;

this.groupBox1.TabStop = false;

//

// radioButton2

//

this.radioButton2.Location = new System.Drawing.Point(251, 17);

this.radioButton2.Name = "radioButton2";

this.radioButton2.Size = new System.Drawing.Size(87, 23);

this.radioButton2.TabIndex = 1;

this.radioButton2.Text = "Молекула";

this.radioButton2.UseVisualStyleBackColor = true;

//

// radioButton1

//

this.radioButton1.Checked = true;

this.radioButton1.Location = new System.Drawing.Point(27, 17);

this.radioButton1.Name = "radioButton1";

this.radioButton1.Size = new System.Drawing.Size(87, 23);

this.radioButton1.TabIndex = 0;

this.radioButton1.TabStop = true;

this.radioButton1.Text = "Атом";

this.radioButton1.UseVisualStyleBackColor = true;

this.radioButton1.CheckedChanged += new System.EventHandler(this.radioButton1\_CheckedChanged);

//

// buttonH

//

this.buttonH.Location = new System.Drawing.Point(48, 78);

this.buttonH.Name = "buttonH";

this.buttonH.Size = new System.Drawing.Size(103, 27);

this.buttonH.TabIndex = 1;

this.buttonH.Text = "Водород";

this.buttonH.UseVisualStyleBackColor = true;

this.buttonH.Click += new System.EventHandler(this.buttonH\_Click);

//

// buttonO

//

this.buttonO.Location = new System.Drawing.Point(229, 151);

this.buttonO.Name = "buttonO";

this.buttonO.Size = new System.Drawing.Size(103, 27);

this.buttonO.TabIndex = 1;

this.buttonO.Text = "Кислород";

this.buttonO.UseVisualStyleBackColor = true;

this.buttonO.Click += new System.EventHandler(this.buttonO\_Click);

//

// buttonCl

//

this.buttonCl.Location = new System.Drawing.Point(229, 78);

this.buttonCl.Name = "buttonCl";

this.buttonCl.Size = new System.Drawing.Size(103, 27);

this.buttonCl.TabIndex = 1;

this.buttonCl.Text = "Хлор";

this.buttonCl.UseVisualStyleBackColor = true;

this.buttonCl.Click += new System.EventHandler(this.buttonCl\_Click);

//

// buttonK

//

this.buttonK.Location = new System.Drawing.Point(48, 151);

this.buttonK.Name = "buttonK";

this.buttonK.Size = new System.Drawing.Size(103, 27);

this.buttonK.TabIndex = 1;

this.buttonK.Text = "Калий";

this.buttonK.UseVisualStyleBackColor = true;

this.buttonK.Click += new System.EventHandler(this.buttonK\_Click);

//

// buttonW

//

this.buttonW.Location = new System.Drawing.Point(48, 299);

this.buttonW.Name = "buttonW";

this.buttonW.Size = new System.Drawing.Size(103, 27);

this.buttonW.TabIndex = 1;

this.buttonW.Text = "Вода";

this.buttonW.UseVisualStyleBackColor = true;

this.buttonW.Click += new System.EventHandler(this.buttonW\_Click);

//

// buttonKCl

//

this.buttonKCl.Location = new System.Drawing.Point(48, 357);

this.buttonKCl.Name = "buttonKCl";

this.buttonKCl.Size = new System.Drawing.Size(103, 27);

this.buttonKCl.TabIndex = 1;

this.buttonKCl.Text = "Хлорид Калия";

this.buttonKCl.UseVisualStyleBackColor = true;

this.buttonKCl.Click += new System.EventHandler(this.buttonKCl\_Click);

//

// buttonHCl

//

this.buttonHCl.Location = new System.Drawing.Point(229, 299);

this.buttonHCl.Name = "buttonHCl";

this.buttonHCl.Size = new System.Drawing.Size(103, 27);

this.buttonHCl.TabIndex = 1;

this.buttonHCl.Text = "Соляная кислота";

this.buttonHCl.UseVisualStyleBackColor = true;

this.buttonHCl.Click += new System.EventHandler(this.buttonHCl\_Click);

//

// buttonKOH

//

this.buttonKOH.Location = new System.Drawing.Point(229, 357);

this.buttonKOH.Name = "buttonKOH";

this.buttonKOH.Size = new System.Drawing.Size(103, 27);

this.buttonKOH.TabIndex = 1;

this.buttonKOH.Text = "Гидроксид Калия";

this.buttonKOH.UseVisualStyleBackColor = true;

this.buttonKOH.Click += new System.EventHandler(this.buttonKOH\_Click);

//

// label1

//

this.label1.Location = new System.Drawing.Point(48, 238);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(259, 28);

this.label1.TabIndex = 2;

this.label1.Text = "Молекулы:";

this.label1.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;

//

// label2

//

this.label2.Location = new System.Drawing.Point(48, 131);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(100, 17);

this.label2.TabIndex = 3;

this.label2.Text = "Только атом:";

//

// ChooseItemWindow

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(396, 411);

this.Controls.Add(this.label2);

this.Controls.Add(this.label1);

this.Controls.Add(this.buttonKOH);

this.Controls.Add(this.buttonK);

this.Controls.Add(this.buttonHCl);

this.Controls.Add(this.buttonCl);

this.Controls.Add(this.buttonKCl);

this.Controls.Add(this.buttonW);

this.Controls.Add(this.buttonO);

this.Controls.Add(this.buttonH);

this.Controls.Add(this.groupBox1);

this.Name = "ChooseItemWindow";

this.Text = "ChooseItemWindow";

this.groupBox1.ResumeLayout(false);

this.ResumeLayout(false);

}

private System.Windows.Forms.Label label2;

private System.Windows.Forms.Button buttonKOH;

private System.Windows.Forms.Button buttonKCl;

private System.Windows.Forms.Button buttonHCl;

private System.Windows.Forms.Button buttonW;

private System.Windows.Forms.Button buttonCl;

private System.Windows.Forms.Button buttonH;

private System.Windows.Forms.Button buttonK;

private System.Windows.Forms.Button buttonO;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.GroupBox groupBox1;

private System.Windows.Forms.RadioButton radioButton1;

private System.Windows.Forms.RadioButton radioButton2;

#endregion

}

}

**Interactions.cs**

using System;

using System.Collections.Generic;

using System.Drawing;

namespace IndividualTask

{

partial class Form1

{

private void SetReactions(List<IParticle> reagents)

{

var newContents = new List<IParticle>();

int i = 0;

while (i < reagents.Count)

{

var thisReagent = reagents[i];

var thisReagentLocation = TDim.TTT.Projection(thisReagent.GetCoords());

var closestReagents = new List<IParticle>();

\_tree.GetContentsLimited(closestReagents,

new Rectangle(thisReagentLocation.X - 20, thisReagentLocation.Y - 20, 40, 40));

i++;

foreach (var anotherReagent in closestReagents)

{

if (!thisReagent.Equals(anotherReagent) && \_tempCoeff>0)

{

var thisType = thisReagent.GetType().ToString().Split('.')[1];

var anotherType = anotherReagent.GetType().ToString().Split('.')[1];

var center = (thisReagent.GetCoords() + anotherReagent.GetCoords()) / 2;

bool reactionCompleted = true;

if (thisType[0] == anotherType[0])

{

#region AtomReactions

if (thisType[0] == 'A')

{

if (thisType == anotherType && thisType[1] != 'P')

{

switch (thisType[1])

{

case 'H':

newContents.Add(new MHydrogen(center.x, center.y, center.z));

break;

case 'O':

newContents.Add(new MOxygen(center.x, center.y, center.z));

break;

case 'C':

newContents.Add(new MChlorine(center.x, center.y, center.z));

break;

}

}

else reactionCompleted = false;

}

#endregion

#region MoleculeReactions

else

{

switch (thisType + anotherType)

{

case "MHydrogenMOxygen":

case "MOxygenMHydrogen":

newContents.Add(new MHOH(center.x+10,center.y,center.z));

newContents.Add(new AOxygen(center.x,center.y+10,center.z));

break;

case "MHydrogenMChlorine":

case "MChlorineMHydrogen":

newContents.Add(new MHCl(center.x+10,center.y,center.z));

newContents.Add(new MHCl(center.x,center.y+10,center.z));

break;

case "MHOHMKCl":

case "MKClMHOH":

newContents.Add(new AHydrogen(center.x,center.y+10,center.z));

newContents.Add(new AChlorine(center.x+10,center.y,center.z));

newContents.Add(new MKOH(center.x,center.y,center.z));

break;

default:

reactionCompleted = false;

break;

}

}

#endregion

}

else

{

if (thisType + anotherType == "MHClAPotassium" || thisType + anotherType == "APotassiumMHCl")

{

newContents.Add(new MKCl(center.x+10, center.y, center.z));

newContents.Add(new AHydrogen(center.x, center.y+10, center.z));

}

else reactionCompleted = false;

}

if (reactionCompleted)

{

i--;

reagents.Remove(thisReagent);

reagents.Remove(anotherReagent);

thisReagent.Clear(\_graphics,\_drawState);

anotherReagent.Clear(\_graphics,\_drawState);

break;

}

}

}

}

foreach (var item in newContents)

{

reagents.Add(item);

}

}

private void ApplyForce(List<IParticle> particles)

{

foreach (var particle in particles)

{

foreach (var nearbyParticle in particles)

{

if (!nearbyParticle.Equals(particle) && \_tempCoeff>0)

{

var dS = (particle.GetCoords()-nearbyParticle.GetCoords())\*Math.Pow(3/TDim.TTT.Radius(particle.GetCoords(), nearbyParticle.GetCoords()),2);

if (TDim.TTT.Radius(particle.GetCoords(), nearbyParticle.GetCoords())>=10)

{

particle.SetCoords(particle.GetCoords() - dS);

nearbyParticle.SetCoords(nearbyParticle.GetCoords() + dS);

}

else

{

particle.SetCoords(particle.GetCoords() + dS\*2);

nearbyParticle.SetCoords(nearbyParticle.GetCoords() - dS\*2);

}

}

}

}

}

}

}

**QuadTree.cs**

using System.Collections.Generic;

using System.Drawing;

using IndividualTask;

namespace Tree

{

public class QuadTree

{

//Дереву передаем его "границы"

public QuadTree(Rectangle boundaries)

{

ul = null;

ur = null;

ll = null;

lr = null;

\_boundaries = boundaries;

\_elements = new List<IParticle>();

}

//Элемент добавляем основе. Если превысили лимит - разделяем. Если уже разделен - то листьям

public void Append(IParticle element)

{

if (ul != null) AddToLower(element);

else

{

\_elements.Add(element);

if (\_elements.Count > Capacity) Split();

}

}

//Разделение корня на 4 ветви

private void Split()

{

ll = new QuadTree(new Rectangle(\_boundaries.X,\_boundaries.Y,\_boundaries.Width/2,\_boundaries.Height/2));

lr = new QuadTree(new Rectangle(\_boundaries.X+\_boundaries.Width/2,\_boundaries.Y,\_boundaries.Width/2,\_boundaries.Height/2));

ul = new QuadTree(new Rectangle(\_boundaries.X,\_boundaries.Y+\_boundaries.Height/2,\_boundaries.Width/2,\_boundaries.Height/2));

ur = new QuadTree(new Rectangle(\_boundaries.X+\_boundaries.Width/2,\_boundaries.Y+\_boundaries.Height/2,\_boundaries.Width/2,\_boundaries.Height/2));

foreach (var element in \_elements)

{

AddToLower(element);

}

\_elements.Clear();

\_elements = null;

}

//Добавление элемента листьям

private void AddToLower(IParticle element)

{

var location = TDim.TTT.Projection(element.GetCoords());

if (location.X<=\_boundaries.X+\_boundaries.Width/2 && location.Y<=\_boundaries.Y+\_boundaries.Height/2) ll.Append(element);

if (location.X>\_boundaries.X+\_boundaries.Width/2 && location.Y<=\_boundaries.Y+\_boundaries.Height/2) lr.Append(element);

if (location.X<=\_boundaries.X+\_boundaries.Width/2 && location.Y>\_boundaries.Y+\_boundaries.Height/2) ul.Append(element);

if (location.X>\_boundaries.X+\_boundaries.Width/2 && location.Y>\_boundaries.Y+\_boundaries.Height/2) ur.Append(element);

}

//Отрисовка элемнтов

public void DrawContents(Graphics gr, int drawstate)

{

if (ul==null)

foreach (var element in \_elements)

{

element.Draw(gr, drawstate);

}

else

{

ul.DrawContents(gr,drawstate);

ur.DrawContents(gr,drawstate);

ll.DrawContents(gr,drawstate);

lr.DrawContents(gr,drawstate);

}

}

public void ClearContents(Graphics gr, int drawstate)

{

if (ul==null)

foreach (var element in \_elements)

{

element.Clear(gr, drawstate);

}

else

{

ul.ClearContents(gr,drawstate);

ur.ClearContents(gr,drawstate);

ll.ClearContents(gr,drawstate);

lr.ClearContents(gr,drawstate);

}

}

//Логика

public void Tick(double coeff)

{

if (ul==null)

foreach (var element in \_elements)

{

element.Tick(coeff);

}

else

{

ul.Tick(coeff);

ur.Tick(coeff);

ll.Tick(coeff);

lr.Tick(coeff);

}

}

//Границы

public void DrawBoundaries(Graphics gr, Color color)

{

if (ul != null)

{

ul.DrawBoundaries(gr,color);

ur.DrawBoundaries(gr,color);

ll.DrawBoundaries(gr,color);

lr.DrawBoundaries(gr,color);

gr.DrawLine(new Pen(color),\_boundaries.X+\_boundaries.Width/2,\_boundaries.Y,\_boundaries.X+\_boundaries.Width/2,\_boundaries.Y+\_boundaries.Height);

gr.DrawLine(new Pen(color),\_boundaries.X,\_boundaries.Y+\_boundaries.Height/2,\_boundaries.X+\_boundaries.Width,\_boundaries.Y+\_boundaries.Height/2);

}

}

//Элементы дерева

public void GetContents(List<IParticle> array)

{

if (array==null) array = new List<IParticle>();

if (ul == null)

{

foreach (var element in \_elements) array.Add(element);

}

else

{

ul.GetContents(array);

ur.GetContents(array);

ll.GetContents(array);

lr.GetContents(array);

}

}

//Элементы, ограниченные определенной областью

public void GetContentsLimited(List<IParticle> array, Rectangle boundaries)

{

if (array==null) array = new List<IParticle>();

if (boundaries.IntersectsWith(\_boundaries))

{

if (ul == null) foreach (var element in \_elements)

{

var location = TDim.TTT.Projection(element.GetCoords());

if (location.X<boundaries.X+boundaries.Width && location.X>boundaries.X &&

location.Y<boundaries.Y+boundaries.Height && location.Y>boundaries.Y) array.Add(element);

}

else

{

ul.GetContentsLimited(array,boundaries);

ur.GetContentsLimited(array,boundaries);

ll.GetContentsLimited(array,boundaries);

lr.GetContentsLimited(array,boundaries);

}

}

}

#region Leafs

private QuadTree ul;

private QuadTree ur;

private QuadTree ll;

private QuadTree lr;

#endregion

#region Parameters

private Rectangle \_boundaries;

private List<IParticle> \_elements;

private const int Capacity =4;

#endregion

}

}

**ThreeDim.cs**

using System;

using System.Drawing;

namespace TDim

{

public struct TPoint

{

public double x;

public double y;

public double z;

public TPoint(double \_x, double \_y, double \_z)

{

x = \_x;

y = \_y;

z = \_z;

}

public static TPoint operator +(TPoint lhs, TPoint rhs) => new TPoint(lhs.x+rhs.x,lhs.y+rhs.y,lhs.z+rhs.z);

public static TPoint operator -(TPoint lhs) => new TPoint(-lhs.x, -lhs.y, -lhs.z);

public static TPoint operator -(TPoint lhs, TPoint rhs) => lhs+(-rhs);

public static TPoint operator \*(TPoint lhs, double rhs) => new TPoint((lhs.x\*rhs),(lhs.y\*rhs),(lhs.z\*rhs));

public static TPoint operator /(TPoint lhs, double rhs) => lhs \* (1 / rhs);

}

public class TTT

{

public static Point Projection(TPoint tDim)

{

double x = (tDim.x - tDim.y) \* Math.Cos(Math.PI/6);

double y = (tDim.x + tDim.y) \* Math.Sin(Math.PI/6) - tDim.z;

return new Point((int)x+401,(int)y+183);

}

public static TPoint RotateZ(TPoint startVector,double angle)

{

return new TPoint((int)(startVector.x\*Math.Cos(angle)-startVector.y\*Math.Sin(angle)),

(int)(startVector.x\*Math.Sin(angle)+startVector.y\*Math.Cos(angle)),

startVector.z

);

}

public static TPoint RotateX(TPoint startVector,double angle)

{

return new TPoint(startVector.x,

(int)(startVector.y\*Math.Cos(angle)-startVector.z\*Math.Sin(angle)),

(int)(startVector.y\*Math.Sin(angle)+startVector.z\*Math.Cos(angle))

);

}

public static TPoint RotateY(TPoint startVector,double angle)

{

return new TPoint((int)(startVector.x\*Math.Cos(angle)-startVector.z\*Math.Sin(angle)),

startVector.y,

(int)(startVector.x\*Math.Sin(angle)+startVector.z\*Math.Cos(angle))

);

}

public static double Radius(TPoint lhs, TPoint rhs)

{

return Math.Sqrt(Math.Pow((lhs.x - rhs.x), 2) + Math.Pow((lhs.y - rhs.y), 2) + Math.Pow((lhs.z - rhs.z), 2));

}

}

}