Lösungsideen der Aufgaben des 26. Bundeswettbewerb Informatik

Markus Schröder
Tim Eberts
Sven Hertling

09. November 2007

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

Inhaltsverzeichnis

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

1. Aufgabe 1: Prämienjagd

1.1. Lösungsidee

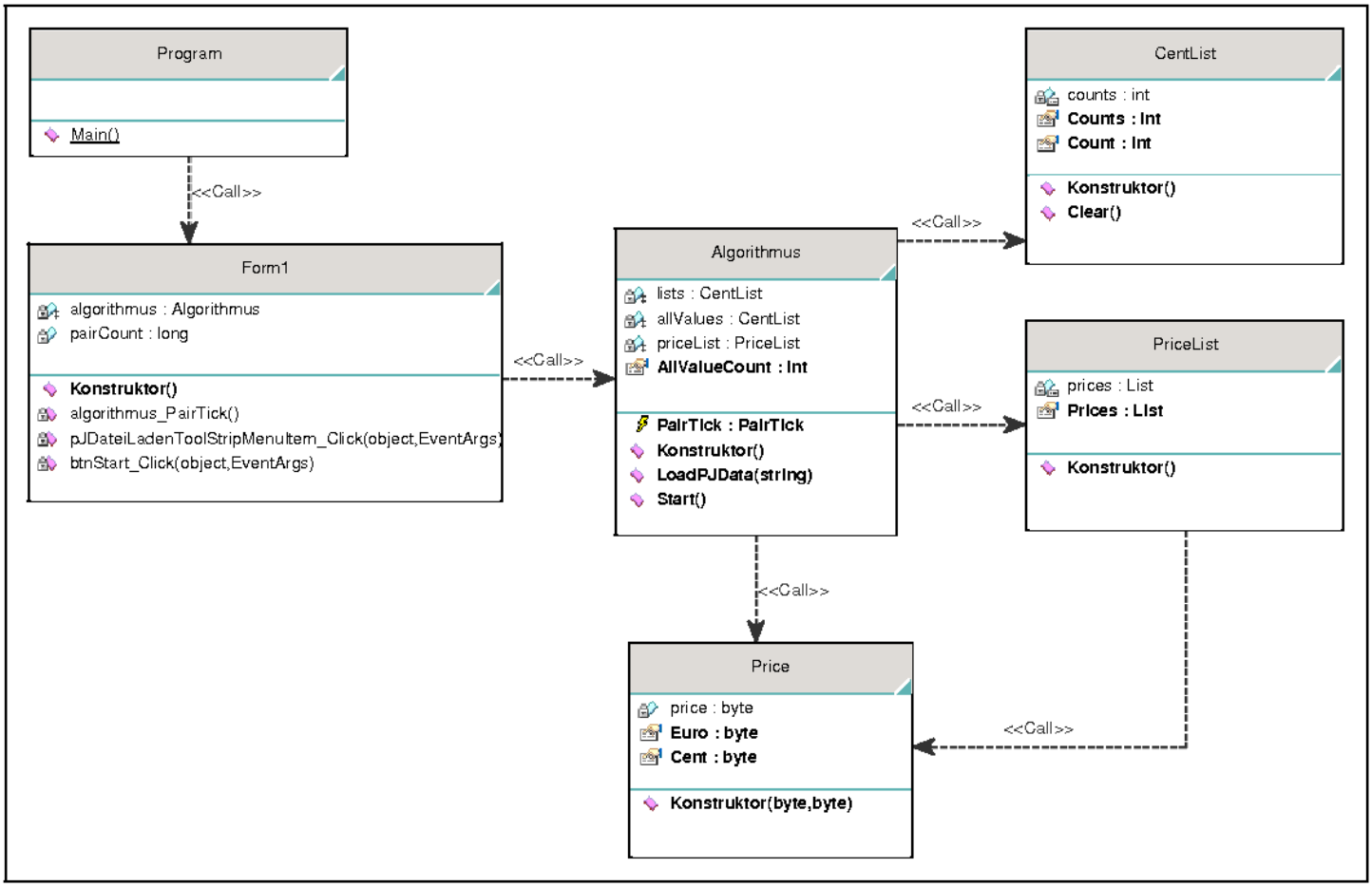
Es sollen Paare gebildet werden die zusammengezählt 11, 33, 55, 77 oder 99 ergeben. Daraus folgt, dass jede Zahl 5 Partner haben kann. Also beispielsweise kann die Zahl 10 folgende Partner haben: 10 + 1 = 11, 10 + 23 = 33, 10 + 45 = 55, 10 + 67 = 77, 10 + 89 = 99. Die 5 möglichen Partner dieser Zahl sind also 1, 23, 45, 67, 89.

Die Zahlen werden, je nachdem wie viele Partner sie haben können, separat zusammengefasst. Die Zahlen mit nur einer Möglichkeit einen Partner zu finden, werden mehr bevorzugt als die Zahlen mit zwei oder mehreren Möglichkeiten (usw.).

Wenn ein Paar gebildet worden ist, kann es sein, das eine Zahl, die vorher zwei Möglichkeiten hatte, jetzt nur noch eine Möglichkeit hat. Dies resultiert daraus, dass diese Preise nicht noch mal für die Paarsuche benutzt werden können. Deshalb muss nach jeder Paarbildung die Möglichkeiten neu ausgerechnet werden.

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

1.2. Programm-Dokumentation



Es gibt zwei Daten Klassen, die die eingelesen Preise speichert: Die *PriceList* und die *Centlist.*

In der *CentList* befindet sich ein Int-Array als Attribut, das 100 Einträge enthält. Der Index des Arrays ist der Centbetrag und der Wert ist die Anzahl des Centbetrages. Damit bleibt die Arbeitsspeicherauslastung gering.

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

In der Klasse *PriceList* befindet sich eine Liste der Klasse *Price*. Die Klasse *Price* enthält als Attribute, *euro* und *cent.* Da der Eurobetrag in den Beispielslisten des Bwinfs nie über 99 geht, kann als Typ *byte* gewählt werden. Dies gilt auch für die Centbeträge. Damit bleibt die Arbeitsspeicherauslastung gering.

Die Klasse *Algorithmus* enthält ein Attribut namens *lists* vom Typ *CentList[].* Dieses Array hat 5 Einträge. Im Index 0 werden später alle Centbeträge gespeichert, die maximal einen Partner haben, im Index 1 dann 2 Partner usw..

Die Centliste *allvalues* enthält alle Centbeträge. Später werden aus dieser Centliste die gepaarten Centbeträge gestrichen, d.h. die Anzahl wird um Eins bei ihnen heruntergesetzt.

Die Methode *Start()* in der Klasse *Algorithmus* startet den Algorithmus.

Der Alogithmus läuft in einer While-Schleife. Diese läuft solange, bis die geladenen Preise einen Partner gefunden haben und damit die Liste leer ist.

Die Liste *lists* werden jedes mal gelöscht. Danach werden die Centwerte neu berechnet. Es wird berechnet, wie viel Partner ein Centwert hat. Die Centwerte, die keine Partner haben, werden aus der Liste gestrichen, also ihr Count wird um eins heruntergesetzt.

Die Listen werden nacheinander durchgearbeitet, wobei *lists[0]* bevorzugt wird.

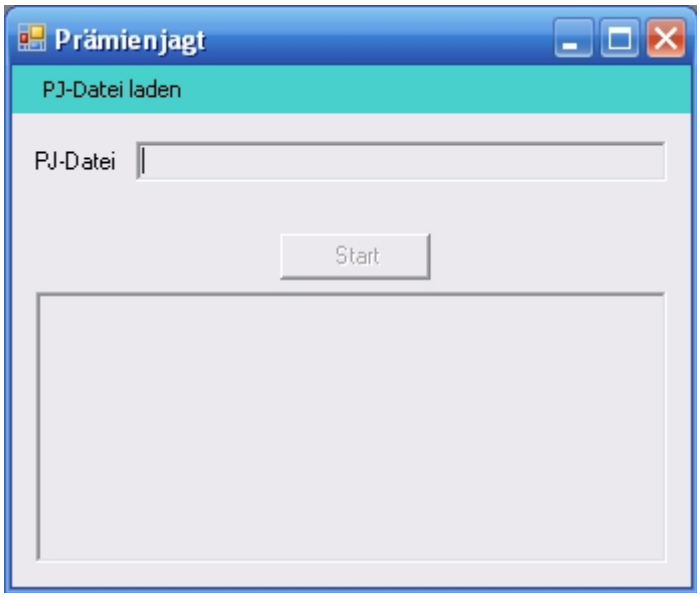
Es wird immer der oberste Centwert genommen und überprüft, welche von seinen Partner am häufigsten vorkommt.

Wenn der linke und rechte Partner gefunden wurde, wird die *PriceList* nach den Preisen durchsucht, die diese Centwerte haben. Die Preise werden dann in eine Datei geschrieben. Der

Count wird bei den gepaarten Centwerten um eins heruntergesetzt.

Nachdem ein Paar gebildet wurde, wird wieder bei *lists[0]* angefangen.

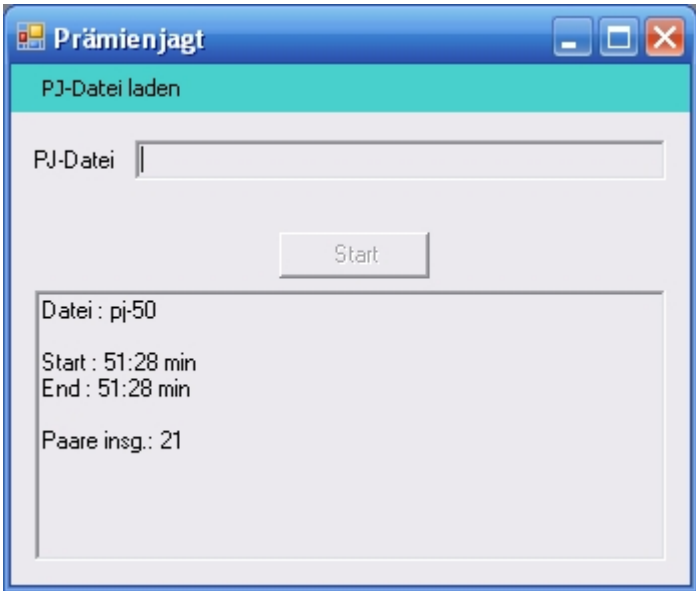
1.3. Programm-Ablaufprotokoll



Dies ist die Hauptform.
Bei einem Klick auf „PJ-Datei laden“ kann der Anwender durch einen OpenFileDialog eine PJ-Datei auswählen.

Da der PC des Anwenders evt. wenig Arbeitsspeicher haben kann, wird er darauf aufmerksam gemacht, dass die größte zu ladende Datei mit 5000000 Einträgen ca. 200 MB Arbeitsspeicher beansprucht.

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181



Nach Beendigung des Rechenvorgangs wird das Ergebnis in einer Textbox dargestellt.

Die erstellten Paare sind in der Datei „output.txt“ im Ordner des Programms zu finden.

1.3.1. Gegebenes Beispiel pj-50

```
98.30
55.88
 9.64
83.96
35.88
 7.64
73.82
54.85
63.07
 3.98
48.93
50.90
70.35
 3.23
22.74
 6.98
70.72
74.80
95.84
60.95
87.12
75.33
20.41
91.42
94.32
76.41
 7.38
84.92
86.06
 0.11
40.92
 0.19
55.99
66.37
99.96
 7.69
74.00
73.79
78.35
37.07
93.58
```

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

```
27.28
 3.79
63.93
46.32
42.35
86.73
17.04
32.97
82.57
```

Ergebnis der Berechnung:

```
( 83.96   66.37 )
( 54.85   84.92 )
( 95.84   48.93 )
( 63.93   86.6 )
( 75.33   74.0 )
( 55.99   87.12 )
( 98.30   7.69 )
( 9.64   70.35 )
( 7.64   78.35 )
( 0.11   55.88 )
( 35.88   3.23 )
( 94.32   73.79 )
( 46.32   3.79 )
( 3.98   82.57 )
( 6.98   42.35 )
( 32.97   74.80 )
( 73.82   60.95 )
( 7.38   86.73 )
( 17.4   63.7 )
( 37.7   40.92 )
( 20.41   93.58 )
```

1.3.2. Eigenes Beispiel

```
9.48
2.47
73.11
88.79
13.54
0.74
77.33
49.12
48.26
```

Ergebnis der Berechnung:

```
( 88.79   13.54 )
```

1.4. Programm-Text

```
static class Program
{
    [STAThread]
    static void Main()
    {
        Application.Run(new Form1());
    }
}

public partial class Form1 : Form
{
    private Algorithmus algorithmus;
    private long pairCount;

    public Form1()
    {
        InitializeComponent();
        algorithmus = new Algorithmus();
        algorithmus.PairTick += new PairTick(algorithmus_PairTick);
    }

    private void algorithmus_PairTick()
    {
        pairCount++;
    }

    private void pJDateiLadenToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Da alle Preise mit Eurobeträgen geladen werden, wird viel
Arbeitsseicher benötigt (max. 200MB).", "Warnung!", MessageBoxButtons.OK,
MessageBoxIcon.Warning);

        OpenFileDialog ofd = new OpenFileDialog();
        ofd.InitialDirectory = Application.StartupPath;
        ofd.ShowDialog();

        if (ofd.FileName == "")
        {
            return;
        }

        algorithmus.LoadPJData(ofd.FileName);

        tboxLoadData.Text = ofd.FileName.Substring(ofd.FileName.LastIndexOf('\\') +
1,ofd.FileName.Length - ofd.FileName.LastIndexOf('\\') - 1);
        btnStart.Enabled = true;
    }

    private void btnStart_Click(object sender, EventArgs e)
    {
```

```
        btnStart.Enabled = false;
        tboxInformation.Text = "";

        tboxInformation.Text += "Datei : " + tboxLoadData.Text + "\r\n\r\n";

        tboxInformation.Text += "Start : " + DateTime.Now.Minute + ":" +
DateTime.Now.Second + " min\r\n";
        algorithmus.Start();
        tboxInformation.Text += "End : " + DateTime.Now.Minute + ":" +
DateTime.Now.Second + " min\r\n\r\n";

        tboxInformation.Text += "Paare insg.: " + pairCount;

        tboxLoadData.Text = "";
    }
}

public delegate void PairTick();

class Algorithmus
{
    private CentList[] lists;
    private CentList allValues;

    public event PairTick PairTick;
    private PriceList priceList;


    public Algorithmus()
    {
        this.lists = new CentList[5];
        this.lists[0] = new CentList();
        this.lists[1] = new CentList();
        this.lists[2] = new CentList();
        this.lists[3] = new CentList();
        this.lists[4] = new CentList();
        this.allValues = new CentList();
        this.priceList = new PriceList();
    }

    public void LoadPJData(string path)
    {
        allValues = new CentList();

        StreamReader strR = new StreamReader(path);

        while (!strR.EndOfStream)
        {
            string[] euroAndCent = strR.ReadLine().Split(new char[] { '.' });
            priceList.Prices.Add(new
Price(byte.Parse(euroAndCent[0]),byte.Parse(euroAndCent[1])));
            allValues.Counts[int.Parse(euroAndCent[1])]++;
        }

        strR.Close();
    }

    public void Start()
    {
        StreamWriter strW = new StreamWriter(Application.StartupPath +
"\\output.txt");

        int index = 0;

        while (allValues.Count != 0)//wenn es keine werte mehr gibt, beende
        {
            //clear unterlisten und berechne neu
            this.lists[0].Clear();
            this.lists[1].Clear();
            this.lists[2].Clear();
            this.lists[3].Clear();
            this.lists[4].Clear();

            //erstelle unterlisten aus der allvalue
            for (int i = 0; i < 100; i++)
            {
                //i ist der centwert
                //nur wenn i überhaupt da ist, also count > 0 hat
                if (allValues.Counts[i] != 0)
                {
                    int count = 0;
```

```
            int partner = -1;

            for (int j = 0; j < 5; j++)
            {
                partner = (99 - 22 * j) - i;
                if (partner < 0)
                {
                    partner += 100;
                }

                if (allValues.Counts[partner] != 0)
                {
                    count++;
                }
            }

            //anhand von count, ordne in liste ein
            //also erhöhe count von i (= centbetrag)
            if (count != 0)
            {
                lists[count - 1].Counts[i]++;
            }
            else
            {
                //wenn count 0 ist, dann steiche von der liste, da
                //niemals mit i ein paar gebildet werden kann
                allValues.Counts[i]--;
            }
        }
    }


    if (lists[index].Count > 0)
    {
        int leftValue = -1;
        //suche erste stelle, von lists[index] die nicht 0 count hat
        for (int i = 0; i < 100; i++)
        {
            if (lists[index].Counts[i] != 0)
            {
                leftValue = i;
            }
        }

        //suche die möglichkeit, die am größten vorhanden ist
        int maxcount = 0;
        int rightValue = -1;
        int partner = -1;
        for (int j = 0; j < 5; j++)
        {
            partner = (99 - 22 * j) - leftValue;
            if (partner < 0)
            {
                partner += 100;
            }

            if (maxcount < allValues.Counts[partner])
            {
                rightValue = partner;
                maxcount = allValues.Counts[partner];
            }
        }



        //prüfe ob rightvalue vorhanden war
        if (rightValue != -1)
        {
            foreach(Price p in priceList.Prices)
            {
                if(p.Cent == leftValue)
                {
                     strW.Write("( " + p.Euro + "." + p.Cent + "    ");
                     priceList.Prices.Remove(p);
                     break;
                }
            }

            foreach(Price p in priceList.Prices)
            {
                if(p.Cent == rightValue)
```

```
                        {
                            strW.Write(p.Euro + "." + p.Cent + " )\r\n");
                            priceList.Prices.Remove(p);
                            break;
                        }
                    }

                    //nimm die partner und erniedrige den count wert um 1
                    allValues.Counts[leftValue]--;
                    allValues.Counts[rightValue]--;

                    //tick beim paarerstellen
                    PairTick();

                    //fang von vorne an
                    index = 0;
                }
            }
            else
            {
                //wenn die liste keine values mehr hat springe zu der nächsten liste
                index++;
            }

            if (index > 4)
            {
                index = 0;
            }
        }

        strW.Close();
    }

    public int AllValueCount
    {
        get
        {
            return allValues.Count;
        }
    }
}

class CentList
{
    private int[] counts;

    public CentList()
    {
        this.counts = new int[100];
    }

    public void Clear()
    {
        for (int i = 0; i < counts.Length; i++)
        {
            counts[i] = 0;
        }
    }

    public int[] Counts
    {
        get
        {
            return counts;
        }
    }
    public int Count
    {
        get
        {
            //zähle alle zusammen
            int back = 0;
            foreach (int c in counts)
            {
                back += c;
            }
            return back;
        }
    }

}
```

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

```
class PriceList
{
    private List<Price> prices;

    public PriceList()
    {
        prices = new List<Price>();
    }

    public List<Price> Prices
    {
        get
        {
            return prices;
        }
    }
}

class Price
{
    private byte[] price;

    public Price(byte euro, byte cent)
    {
        price = new byte[2];
        price[0] = euro;
        price[1] = cent;
    }

    public byte Euro
    {
        get
        {
            return price[0];
        }
    }
    public byte Cent
    {
        get
        {
            return price[1];
        }
    }
}
```

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

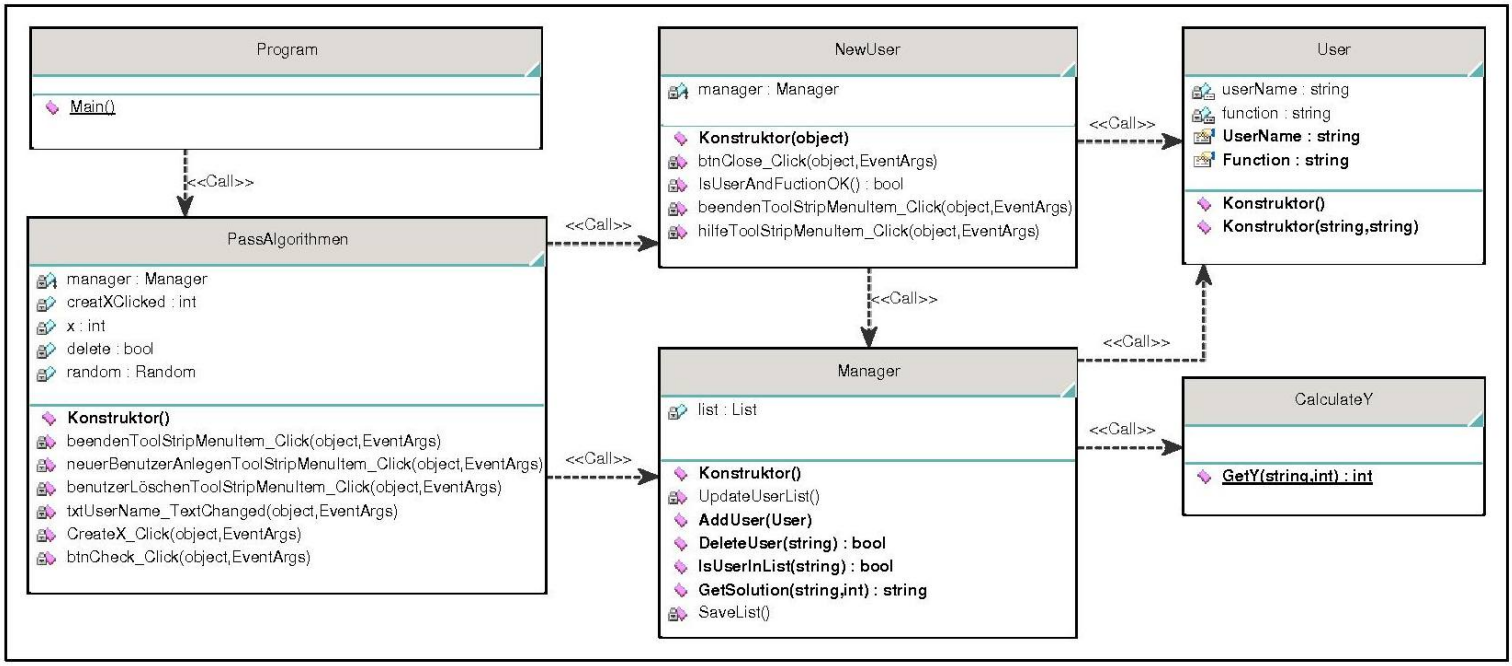
3. Aufgabe 4: Pass-Algorithmen

3.1. Lösungsidee

Die Lösungsidee zur Aufgabe Pass-Algorithmen ist, dass man Benutzer mit ihrem Benutzernamen und Funktionen in einer Datei abspeichert. So kann man auch beim wiederholten Aufruf der Anwendung sich immer noch mit seinem Benutzernamen anmelden. Da man die Funktionen in einer Datei nur als Zeichenkette speichern kann, habe ich überlegt, wie man die Funktion für ein bestimmtes x für f(x) ausrechnen kann.
Beim anlegen eines Benutzers wird weiterhin überprüft, ob die eingegebene Funktion ein „x“ enthält und ob der Benutzername schon vorhanden ist. Falls dies der Fall ist, wird der Benutzer darauf aufmerksam gemacht und es wird kein neuer Benutzer erzeugt. Beim Anmelden selbst kann man nur einmal ein zufälliges x erzeugen und muss das Ergebnis auswerten lassen. Erst danach kann er sich wieder ein neues x erzeugen lassen. Dies funktioniert aber nicht mehr, wenn sich der Benutzer dreimal hintereinander falsch angemeldet hat. Diese Eigenschaft habe ich in mein Programm eingebaut, da man sich dadurch

nicht so oft ein neues x erzeugen kann, bis das x erscheint zu dem man zufälligerweise f(x) weiß. Die Definitionsgrenze habe ich von 0 bis 30 gesetzt und nicht vom Benutzer eingeben lassen, da man dann nur einen kleinen Definitionsbereich auswählen kann, bei dem man f(x) für x kennt.

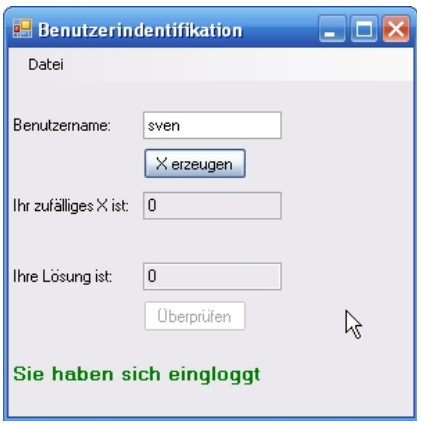
3.2. Programm-Dokumentation



Windows-Form Klasse *PassAlgorithmen*. In dieser Hauptklasse befindet sich ein Objekt der Klasse *Manager,* das eine Liste aller Benutzer enthält. Diese Liste wird im Konstuktor mit dem Inhalt aus der Datei, die die Benutzer mit ihren Funktionen enthält, gefüllt. Sobald das *neuerBenutzerAnlegenToolStrip* Click-Event auftritt, wird die Windows-Form Klasse *NewUser* angezeigt. Dieser wird die Referenz auf das Managerobjekt übergeben, sodass die Klasse durch die Methode *AddUser(User)*, einen neuen Benutzer in die Liste eintragen kann. Wenn dies abgeschlossen ist (das heißt: am Ende der Methode) wird die private Methode *SaveList()* aufgerufen, dass dadurch die Datei immer mit der Liste übereinstimmt. Der Methode *AddUser(User)* übergibt man als Parameter ein Objekt der Klasse *User*, die lediglich als Datenklasse fungiert und den Benutzer, sowie die Funktion als string enthält. Die Methode *GetSolution(string, int)* in *Manager* wird in *PassAlgorithmen*

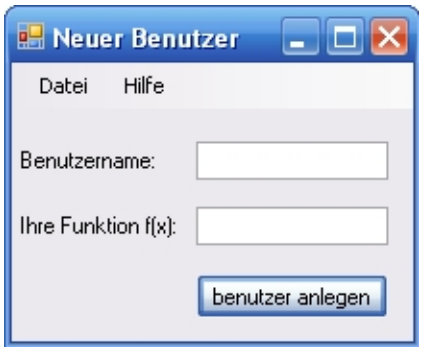
aufgerufen, wenn das *btnCheck_Click*-Event eintritt. Dieser Methode muss man ein string (Benutzername) und int (x) mitgeben. Es wird dann in der Liste nach dem Benutzernamen gesucht und falls dies gelingt, wird die statische Methode *GetY(string, int)* mit der Funktion und dem x aufgerufen. Dieser Rückgabewert wird dann in einen string umgewandelt und wieder der Klasse *PassAlgorithmen* zurückgeben. Dort wird dann das Ergebnis ausgewertet und falls das Suchen in *Manager* fehlschlägt, wird ein konstanter Ausdruck zurückgegeben. Beim Löschvorgang wird die gleiche Prozedur wie beim Anmelden ausgeführt, nur dass dann die Methode *DeleteUser(string)* aufgerufen wird. Diese such wiederum den Benutzer in der Liste und löscht ihn gegebenenfalls. Der Rückgabewert bestimmt, ob der Benutzername vorhanden war und ob der Benutzer gelöscht werden konnte.

3.3. Programm-Ablaufprotokoll



<- Dies ist die Hauptform *PassAlgorithmen*. In die erste Textbox wird der Benutzername geschrieben (z.B. sven), dadurch wird der obere Button aktiviert und man kann damit ein zufälliges x erzeugen. Dieses x wird in die zweite Textbox geschrieben, in die man als Benutzer nur lesen und nicht verändern kann. Wenn man die Lösung in die dritte Textbox eingetragen hat und auf den unteren Button klickt, erscheint unter diesem Button ein Label, ob das Ergebnis richtig oder falsch ist.

Wenn man den Button „X erzeugen“ angeklickt hat, wird er deaktiviert und durch den Button „Überprüfen“ aktiviert, sodass man sich nicht immer solange ein x erzeugen kann, bis das x erscheint, bei dem man f(x) weiß.



Wenn ein neuer Benutzer angelegt wird, öffnet sich diese Form(siehe rechts). Man kann in die erste Textbox sein beliebigen Benutzernamen schreiben und in die zweite Textbox seine Funktion. Falls man dazu Hilfe braucht gibt es neben „Datei“ ein Link zu einer MessageBox, die angibt, welche Bedingungen erfüllt sein müssen, um einen Benutzer anzulegen. Diese Bedingungen werden beim Klick auf den Button „benutzer anlegen“ nochmals überprüft und gegebenenfalls auch mit einer MessageBox darauf aufmerksam gemacht, dass die Eingabe falsch oder ungültig ist.

3.4. Programm-Text

```
static class Program
    {
        /// <summary>
        /// Der Haupteinstiegspunkt für die Anwendung.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new PassAlgorithmen());
        }
    }
public partial class PassAlgorithmen : Form
{
    private Manager manager;
    private int creatXClicked;
    private int x;
    private bool delete;
    private Random random;
    public PassAlgorithmen()
    {
        InitializeComponent();
        //Initialisieren der Variablen
        manager = new Manager();
        random = new Random();
        delete = false;
        creatXClicked = 0;
        x = 0;
    }

    private void beendenToolStripMenuItem_Click(object sender, EventArgs e)
```

```
    {
        this.Close();
    }

    private void neuerBenutzerAnlegenToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        NewUser form2 = new NewUser(manager);
        form2.Show();
    }

    private void benutzerLöschenToolStripMenuItem_Click(object sender, EventArgs e)
    {
        lblDelete.ForeColor = Color.Red;
        lblDelete.Text = "Bei erfolgreicher Anmeldung wird der Benutzer gelöscht";
        lblDelete.Visible = true;
        delete = true; // das Attribut false setzen, dass man auch in anderen Methoden
                       // nachsehen kann, ob gelöscht werden soll
    }

    private void txtUserName_TextChanged(object sender, EventArgs e)
    {
        creatXClicked = 0;//wenn man einen anderen Benutzernamen angegeben hat, spielt
                          //die Anzahl der falschen Ergebnisse keine Rolle...es wird
neu
                          //angefangen zu zählen
        lblLogIn.Text = "";
        if (txtUserName.Text != "")
            CreateX.Enabled = true;
        else
            CreateX.Enabled = false;
    }

    private void CreateX_Click(object sender, EventArgs e)
    {
        x = random.Next(0, 30);
        txtX.Text = x.ToString();//das zufällig erzeugte x wird in die TextBox
geschrieben
        creatXClicked++;
        txtSolution.ReadOnly = false;
        lblLogIn.Text = "";
        CreateX.Enabled = false;
        btnCheck.Enabled = true;
    }

    private void btnCheck_Click(object sender, EventArgs e)
    {
        CreateX.Enabled = true;
        txtSolution.ReadOnly = true;
        btnCheck.Enabled = false;
        if (creatXClicked ==3)
        {
            lblLogIn.ForeColor = Color.OrangeRed;
            lblLogIn.Text = "Zu oft das falsche Ergebnis eingegeben!";
            lblLogIn.Visible = true;
            CreateX.Enabled = false;
            return;
        }
        string tmp=manager.GetSolution(txtUserName.Text, x);
        if (tmp != "NA")//wenn der Benutzername in der Liste ist (oder nicht)
        {
            if (txtSolution.Text == tmp)
            {
                if (delete == false)
                {
                    lblLogIn.ForeColor = Color.Green;
                    lblLogIn.Text = "Sie haben sich eingloggt";
                    lblLogIn.Visible = true;
                }
                else
                {
                    bool isDeleted=manager.DeleteUser(txtUserName.Text);
                    if (isDeleted == true)
                    {
                        lblLogIn.ForeColor = Color.Green;
                        lblLogIn.Text = "Der Benutzer ist gelöscht";
                        lblLogIn.Visible = true;
                    }
                }
            }
            else
```

```
            {
                lblLogIn.ForeColor = Color.Red;
                lblLogIn.Text = "Falsches Ergebnis";
                lblLogIn.Visible = true;
            }
        }
        else
        {
            lblLogIn.ForeColor = Color.OrangeRed;
            lblLogIn.Text = "Der Benutzername ist nicht in der Liste";
            lblLogIn.Visible = true;
        }
        lblDelete.Visible = false;
        delete = false;
    }
}


public partial class NewUser : Form
{
    private Manager manager;
    public NewUser(object tmp)
    {
        InitializeComponent();
        manager = (Manager)tmp;
    }

    private void btnClose_Click(object sender, EventArgs e)
    {
        if (IsUserAndFuctionOK())
        {
            manager.AddUser(new User(txtUserName.Text, txtFunction.Text));
            this.Close();
        }
        else
        {
            MessageBox.Show("Entweder ist der Name schon vorhanden, oder Ihre Funktion
enthält kein x", "Kann den Benutzer nicht speichern", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
    private bool IsUserAndFuctionOK()
    {
        if (txtFunction.Text.Contains("x"))
        {
            if (!manager.IsUserInList(txtUserName.Text))
            {
                return true;
            }
        }
        return false;
    }

    private void beendenToolStripMenuItem_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void hilfeToolStripMenuItem_Click(object sender, EventArgs e)
    {
        MessageBox.Show("-Sie müssen in ihrer Funktion ein x einbinden \n-Für
hochzahlen verwnden Sie bitte ein ^ z.b. x^2\n-Klammer werden nicht berücksichtigt",
"Hilfe zur Eingabe der Funktion", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

class Manager
{
    private List<User> list;
    public Manager()
    {
        list = new List<User>();
        UpdateUserList();
    }

    /// <summary>
    /// Die Benutzer Liste, die als Attribut in dieser KLasse gespeichert ist, wird
neu eingelesen
    /// </summary>
    private void UpdateUserList()
    {
```

```
        try
        {
            StreamReader str = new StreamReader(Directory.GetCurrentDirectory() +
"/test.txt");
            while (!str.EndOfStream)
            {
                list.Add(new User(str.ReadLine(), str.ReadLine()));
            }
            str.Close();
            //für die Implemenation der Aufgabe:
            list.Add(new User("aufgabe","x^2-3"));
        }
        catch(Exception e){}
    }

    /// <summary>
    /// Ein User-Objekt wird der Liste hinzugefügt
    /// </summary>
    /// <param name="user">Ein Objekt der Klasse User, das zur Liste hinzugefügt
werden soll</param>
    public void AddUser(User user)
    {
        list.Add(user);
        SaveList();
    }

    /// <summary>
    /// Der zu übergebende Benutzername wird aus der Liste gesucht und gelöscht
    /// </summary>
    /// <param name="userName">Der Benutzername, der gelöscht werden soll</param>
    /// <returns>Gibt zurück, ob die Person gelöscht wurde</returns>
    public bool DeleteUser(string userName)
    {
        for (int i = 0; i < list.Count; i++)
        {
            if (list[i].UserName == userName)
            {
                list.RemoveAt(i);
                SaveList();
                return true;
            }
        }
        return false;
    }

    /// <summary>
    /// Überprüft, ob der Benutzername schon in der Liste ist
    /// </summary>
    /// <param name="userName">Der zu überprüfende Benutzername</param>
    /// <returns>Gibt zurück, ob der Benutzer vorhanden ist</returns>
    public bool IsUserInList(string userName)
    {
        for (int i = 0; i < list.Count; i++)
        {
            if (list[i].UserName == userName)
            {
                return true;
            }
        }
        return false;
    }
    /// <summary>
    /// Rechnet den Y-Wert aus
    /// </summary>
    /// <param name="userName">Benutzername der überprüft werden soll</param>
    /// <param name="x">Der x-Wert für den Y berechnet werden soll</param>
    /// <returns>Gibt das Ergebnis als String zurück</returns>
    public string GetSolution(string userName, int x)
    {
        for (int i = 0; i < list.Count; i++)
        {
            if (list[i].UserName == userName)
            {
               return CalculateY.GetY(list[i].Function, x).ToString();
            }
        }
        return "NA";//Not Available
    }

    /// <summary>
    /// Die Liste der Benutzer wird in einer Datei gespeichert
```

```
    /// </summary>
    private void SaveList()
    {
        StreamWriter stw = new StreamWriter(Directory.GetCurrentDirectory() +
"/test.txt", false);

        for (int i = 0; i < list.Count; i++)
        {
            stw.WriteLine(list[i].UserName);
            stw.WriteLine(list[i].Function);
        }
        stw.Close();
    }
}

class User
{
    private string userName;
    private string function;
    public User()
    {
        userName = "";
        function = "";

    }
    public User(string userName, string function)
    {
        this.userName = userName;
        this.function = function;
    }
    public string UserName
    {
        get
        {
            return userName;
        }
        set
        {
            userName = value;
        }
    }
    public string Function
    {
        get
        {
            return function;
        }
        set
        {
            function = value;
        }
    }
}
public enum Operator
{
    Plus, Minus, Geteilt, Mal
}

class CalculateY
{
    /// <summary>
    /// Liefert f(x) für die Funktion und x zurück
    /// </summary>
    /// <param name="function">Die Funktion als string</param>
    /// <param name="x">Das X für das die Funktion berechnet werden soll</param>
    /// <returns>Das Ergebnis aus f(x)</returns>
    static public int GetY(string function,int x)
    {
        List<Operator> operatoren = new List<Operator>();
        string[] container = function.Split(new char[] { '+', '-', '/', '*' });

        for (int i = 0; i < function.Length; i++)
        {
            if (function[i] == '+')
            {
                operatoren.Add(Operator.Plus);
            }
            else if (function[i] == '-')
            {
                operatoren.Add(Operator.Minus);
            }
```

```
            else if (function[i] == '*')
            {
                operatoren.Add(Operator.Mal);
            }
            else if (function[i] == '/')
            {
                operatoren.Add(Operator.Geteilt);
            }
        }



        //suche und ersetzt x
        for (int i = 0; i < container.Length; i++)
        {
            if (container[i].Contains("x"))
            {
                container[i] = container[i].Replace("x", Convert.ToString(x));
            }
        }

        List<int> zahlen = new List<int>();

        //zusammenzählen
        for (int i = 0; i < container.Length; i++)
        {
            //wandle um und speichere in zahl


            //sonderfall bsp.: 3^2!
            if (container[i].Contains("^"))
            {
                string[] basisUndExponent = container[i].Split(new char[] { '^' });
                container[i] =
Convert.ToString(Math.Pow(double.Parse(basisUndExponent[0]),
double.Parse(basisUndExponent[1])));
            }

            //speichern
            zahlen.Add(int.Parse(container[i]));
        }



        //je nach reinheinfolge * oder / oder + oder -

        int endergebnis = zahlen[0];

        for (int i = 1; i < zahlen.Count; i++)
        {
            if (operatoren[i - 1] == Operator.Plus)
            {
                endergebnis += zahlen[i];
            }
            else if (operatoren[i - 1] == Operator.Minus)
            {
                endergebnis -= zahlen[i];
            }
            else if (operatoren[i - 1] == Operator.Mal)
            {
                endergebnis *= zahlen[i];
            }
            else if (operatoren[i - 1] == Operator.Geteilt)
            {
                endergebnis /= zahlen[i];
            }
        }



        return endergebnis;
    }
}
```

3.5. Aufgaben

3.5.1. Aufgabe 1

Im folgenden Text werden die in der Aufgabe genannten Eigenschaften in der Hinsicht überprüft, warum gerade diese wünschenswert sind und es werden noch weitere genannt.
Die erste Eigenschaft ist, dass die Anzahl möglicher Pass-Algorithmen groß ist. Dies sollte der Fall sein, da man sonst aus zu wenigen Algorithmen auswählen kann und es dann sehr leicht ist, Rückschlüsse auf die Funktion zu ziehen.
Eine weitere Eigenschaft ist, dass ein Pass-Algorithmus mit wenig Mühe im Kopf in kurzer Zeit ausgeführt werden kann. Er darf nicht zu kompliziert sein, da man sonst verleitet wird, diesen auf ein Blatt zu schreiben, dass man eventuell auch verlieren kann.
Die letzte in der Aufgabe genannte Eigenschaft ist, dass die Beobachtung weniger Paare kaum Rückschlüsse auf den benutzen Algorithmus erlaubt. Falls dies so wäre, würde ein solches System kein Sinn machen, denn dann wäre jedem Dieb bei Beobachtung eines Paares der Algorithmus gleich klar und er könnte den Zugang ausnutzen.
Der Algorithmus muss zudem nicht nur schnell ausrechenbar , sondern auch einfach zu merken sein, sonst wird man verleitet ihn einfach aufzuschreiben .

3.5.2. Aufgabe 2

Das Beispielsystem ist in meinen Augen eher negativ zu bewerten, da man die ganzen Zahlen die in der Tabelle (Matrix) stehen auswendig kennen müsste. Man wird dadurch verleitet diese Tabelle auf ein Papier zu schreiben und geht damit das Risiko ein, diesen Zettel , sozusagen mit den Zugangsdaten, zu verlieren. Zudem werden die Positionen preisgegeben: Das heißt ein x und ein y wert. Dies macht es noch schwieriger den Wert zu finden - und das für zwei Werte, die man auch noch addieren müsste.

3.5.3. Aufgabe 3

Dieses Zugangssystem kann bei allen möglichen Einsatzgebieten benutzt werden, bei dem man eine sichere Benutzeridentifikation braucht.
Das erste System, das ich benutzt habe, ist ein recht einfacher, leicht zu merkender Algorithmus: x^2-3 . Er lässt durch die Subtraktion der Zahl 3 nicht so schnell Rückschlüsse auf den Algorithmus ziehen. Dies ist ein großer Vorteil. Er ist, wie oben erwähnt, leicht zu merken und nicht so komplex. Ein Nachteil ist das Quadrieren. Bei großen Zahlen wird man dies nicht im Kopf ausführen können - und ein Taschenrechner hat man nicht immer dabei. Eine erweiterte Funktion ist zum Beispiel 2*x-3. Dies kann man auch mit höheren Zahlen rechnen und es erlaubt weiterhin durch die Subtraktion keine Rückschlüsse. Dieses System könnte man z.B. als Zugangsberechtigung bei der Bank einsetzten, da in diesen Bereichen viele Ausspähungen vorgenommen werden.
Eine weitere Funktion, die noch weniger Rückschlüsse ziehen lässt könnte lauten: x^2-x. Die Subtraktion hängt nun auch noch vom gewählten x ab. Dieses System könnte man auch in einem Flughafen einsetzen, sodass bestimmte Türen nur für bestimmte Mitarbeiter geöffnet werden können.
Die Implementation habe ich insofern vorgenommen, da man den Algorithmus frei wählen kann und somit auch eine sehr große Anzahl möglicher Pass-Algorithmen gibt. Dadurch kann man auch die Beispiele die ich oben genannt habe durch das Benutzerinterface eingeben. Die Funktion $f(x)=x^2-3$ habe ich im Quellcode dem Benutzernamen „aufgabe“ zugeordnet.

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

4. Aufgabe 5: Kosmischer Tanz

4.1. Lösungsidee

Wie in der Aufgabe steht, soll ein Simulationssystem entwickelt werden. Dabei müssen zwei unterschiedliche Aspekte berücksichtigt werden: Die Anzeige und die Berechnung.

Man kann den Körper nur in Pixelbreichen zeichnen. Wenn man auch in Pixelbreichen rechnet, wird die Berechnung ungenau. Also muss die Berechnung im Hintergrund mit Fließkommazahlen berechnet werden, damit die Flugbahn genau wird.

Der Körper hat insgesamt 3 Vektoren. Diese Vektoren drücken die Geschwindigkeit, sowie die Richtung aus.

Der erste Vektor zeigt stetig auf den Stern. Die Länge des Vektors wird mit der Formel $G*M*m / r^2$ und dann mit $a = F / m$ und $v = a\ t$ berechnet.

Der zweite Vektor ist der Benutzervektor. Der Benutzer kann den Vektor selbst verändern.

Der dritte und letzte Vektor ist der Resultierende Vektor. Addiert man den ersten und zweiten Vektor miteinander, erhält man diesen.

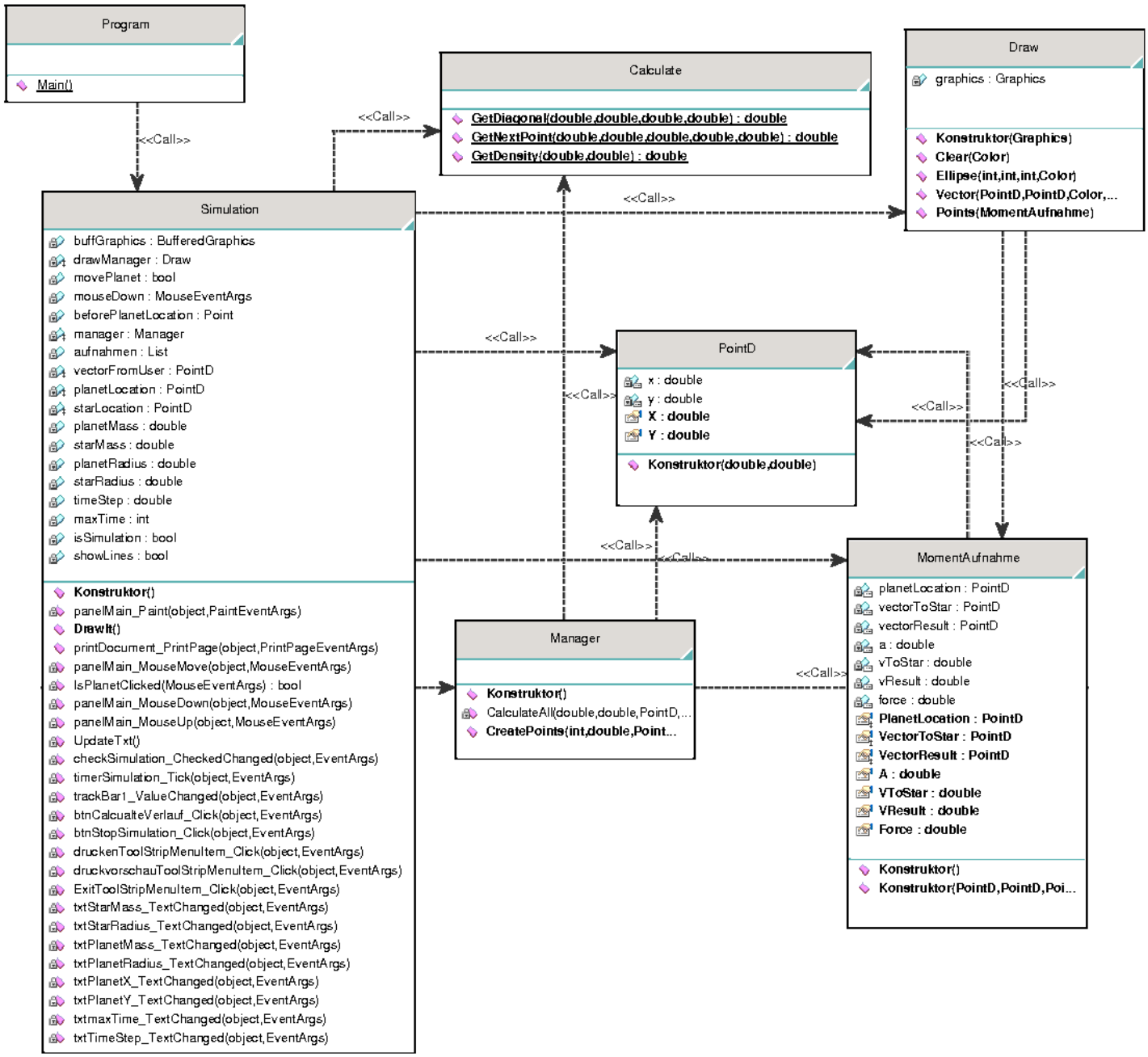
Am Ende hat man einen Vektor, der in eine bestimmte Richtung zeigt und eine bestimmte Geschwindigkeit hat. Da das Ergebnis sehr genau sein sollte, kann man nun ausrechnen, wie viel Meter der Körper in einem bestimmten Zeitintervall zurücklegt. Um die Genauigkeit zu erhöhen, kann man das Zeitintervall auf 0,001 oder niedriger verringern.

Nach dem der Körper auf seine neue Position gesetzt wurde, werden die Vektoren neu berechnet.

Damit bekommt man eine Ansammlung von Punkten, die man abspeichert. Außerdem speichert man die gesamten Daten wie Beschleunigung, Geschwindigkeit usw.. Diese Punkte kann man dann als Linie verbinden. Sie zeigt den genauen Weg, der vom dem Körper zurückgelegt wird.

4.2. Programm-Dokumentation

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181



Die Klasse *Simulation* dient dazu, den Körper sowie den Stern zu zeichnen und bietet eine Eingabeoberfläche für den Anwender. Außerdem sind einige Attribute enthalten, die den Körper, Stern und deren Vektoren beschreiben.

Die Hauptklasse ist die *Manager*. In ihr gibt es die *Methode CalculateAll()*. Der *CalculateAll()*- Methode werden alle Daten gegeben, die sie braucht, um Momentaufnahmen zu erstellen.

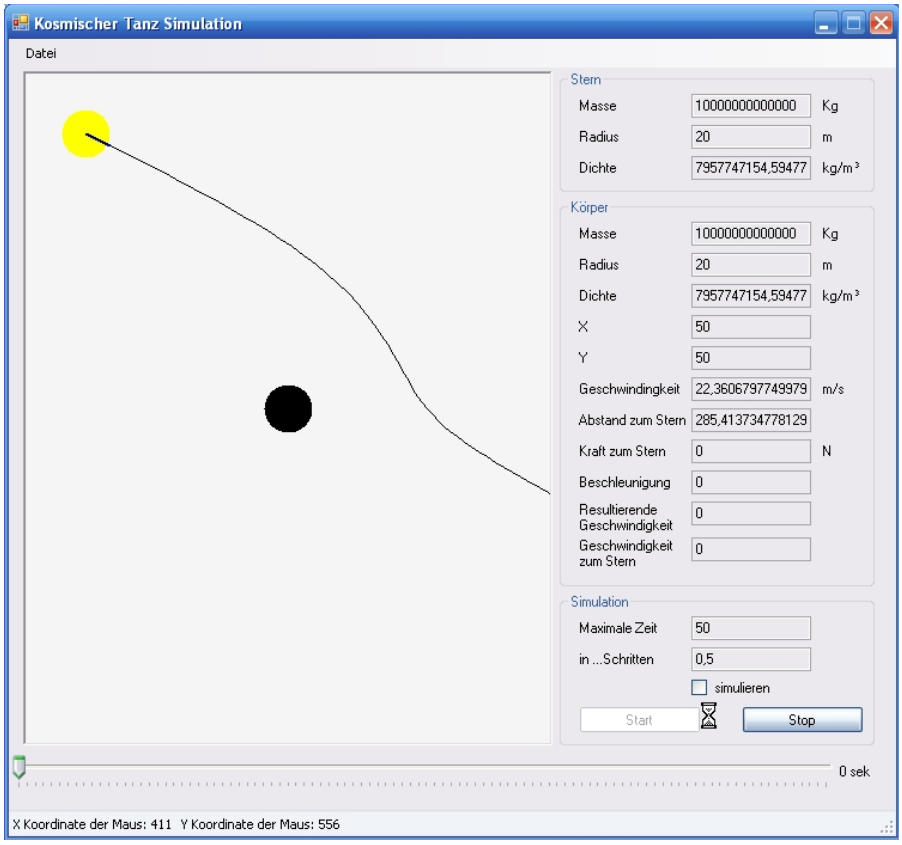
Die Klasse *Momentaufnahme* enthält Attribute, die zu einem bestimmten Zeitpunkt Geschwindigkeit, Beschleunigung und Vektoren (usw.) speichern.

Der Anwender legt in dem Fenster fest, welche Eigenschaften Stern und Körper haben soll und kann den Körper durch Drag&Drop auf der Oberfläche bewegen. Außerdem kann er den Geschwindigkeitsvektor des Körpers bestimmen.

Nachdem der Anwender alle Einstellung vorgenommen hat, kann er mit einem Button die Simulation starten. Die Methode *CalculateAll()* wird ausgeführt und berechnet die Momentaufnahmen. Durch eine Eingabe kann der Anwender die maximale Zeit und den Zeitintervall bestimmen. Durch eine Trackbar ist es möglich den Zeitpunkt der Simulation zu verändern. Der Anwender kann dann nachvollziehen, welche Eigenschaften (Geschwindigkeit usw.) der Körper zu einem bestimmten Zeitpunkt hat.

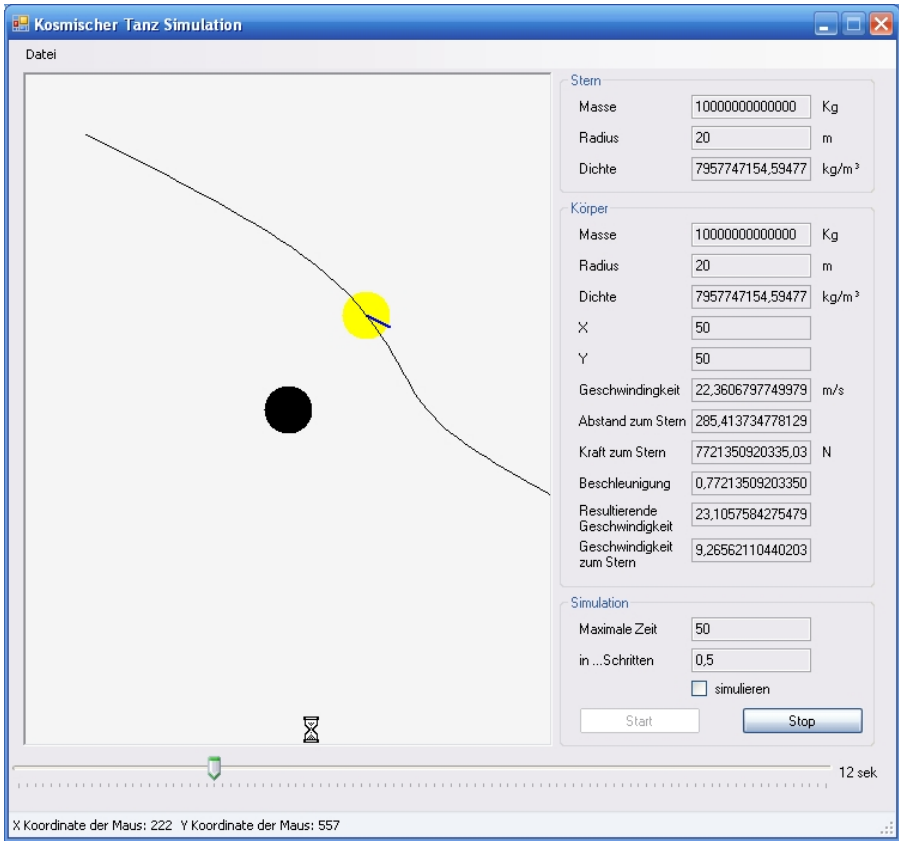
4.3. Programm-Ablaufprotokoll

Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181



Der Anwender kann zuerst die auf der rechten Seite gelegenen Textboxen mit werten füllen. Diese werden dann für die Berechnung der Simulation eingesetzt. Des Weiteren kann der Benutzer per Drag & Drop den Körper verschieben.
Durch einen Rechtsklick auf die grafische Oberfläche kann der Geschwindigkeitsvektor gesetzt werden.

Man muss für die Simulation eine Maximale Zeit angeben und die Zeitintervalle festlegen. Ein Klick auf den Startbutton, startet die Berechnung und zeigt dann eine Verlaufslinie an.

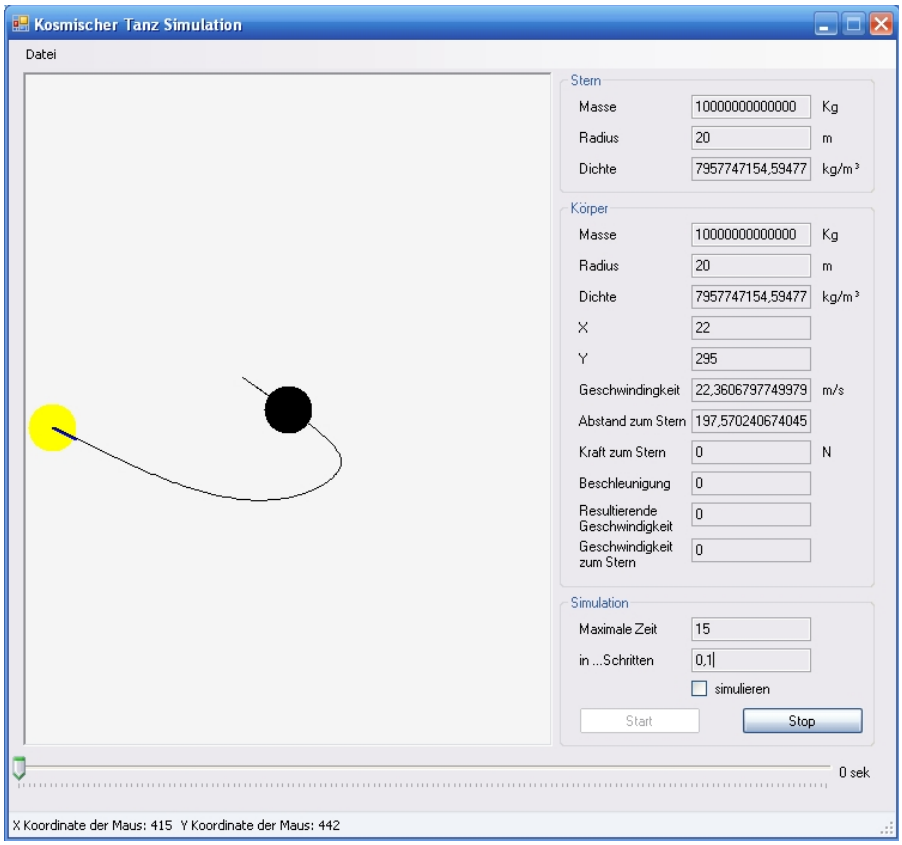


Mit der Trackbar kann man den Körper auf seiner Verlaufslinie bewegen.

Wie man sieht, wird der Körper leicht von dem Stern angezogen, aber seine Geschwindigkeit

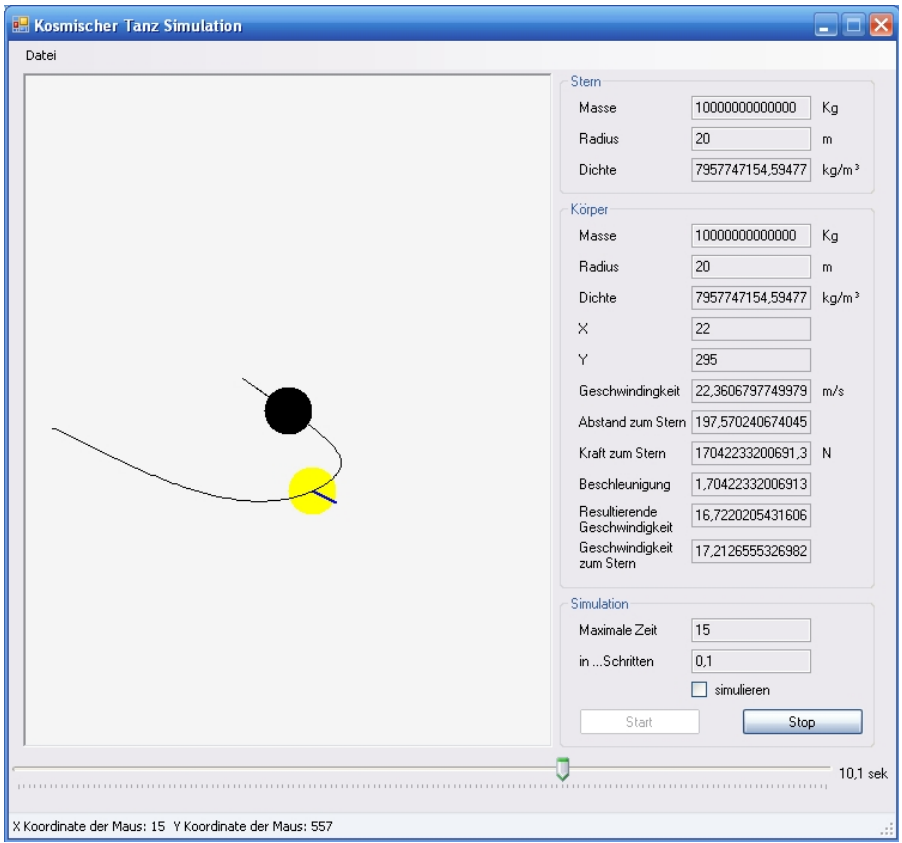
Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

ist so groß und er ist so weit entfernt, dass er nicht in den Stern fliegt.



In dem 2.Verlaufsbeispiel kann man erkennen, dass der Körper stark von dem Stern angezogen wird. Dies resultiert aus den großen Massen der beiden Objekte.

Soll die Trackbar automatisch jeden Zeitintervall um ein Wert erhöhen, dann muss man die Checkbox „simulieren“ drücken.



Markus Schröder
Tim Eberts
Sven Hertling
Einsendenummer: 1181

4.4. Programm-Text

```
/// <summary>
/// Der Haupteinstiegspunkt für die Anwendung.
/// </summary>
static class Program
    {
        /// <summary>
        /// Der Haupteinstiegspunkt für die Anwendung.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Simulation());
        }
    }

class PointD
{
    private double x;
    private double y;

    public PointD(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public double X
    {
        get
        {
            return x;
        }
        set
        {
            x = value;
        }
    }
    public double Y
    {
        get
        {
            return y;
        }
        set
        {
            y = value;
        }
    }
}

class MomentAufnahme
{
    private PointD planetLocation;
    private PointD vectorToStar;
    private PointD vectorResult;
    private double a;
    private double vToStar;
    private double vResult;
    private double force;

    public MomentAufnahme()
    {
```

```
            planetLocation = new PointD(0,0);
            vectorResult = new PointD(0,0);
            vectorToStar = new PointD(0,0);
        }
        public MomentAufnahme(PointD planetLocation, PointD vectorToStar,PointD
    vectorResult, double a, double vToStar, double vResult, double force)
        {
            this.planetLocation = planetLocation;
            this.vectorToStar = vectorToStar;
            this.vectorResult = vectorResult;
            this.a = a;
            this.vToStar = vToStar;
            this.vResult = vResult;
            this.force = force;
        }
        public PointD PlanetLocation
        {
            get
            {
                return planetLocation;
            }
            set
            {
                planetLocation = value;
            }
        }
        public PointD VectorToStar
        {
            get
            {
                return vectorToStar;
            }
            set
            {
                vectorToStar = value;
            }
        }
        public PointD VectorResult
        {
            get
            {
                return vectorResult;
            }
            set
            {
                vectorResult = value;
            }
        }
        public double A
        {
            get
            {
                return a;
            }
            set
            {
                a = value;
            }
        }
        public double VToStar
        {
            get
            {
                return vToStar;
            }
            set
            {
                vToStar = value;
            }
        }
        public double VResult
        {
            get
            {
                return vResult;
            }
            set
            {
                vResult = value;
            }
        }
```

```
    public double Force
    {
        get
        {
            return force;
        }
        set
        {
            force = value;
        }
    }
}

class Draw
{
    Graphics graphics;

    public Draw(Graphics graphics)
    {
        this.graphics = graphics;
    }

    public void Clear(Color color)
    {
        graphics.Clear(color);
    }

    public void Ellipse(int x, int y, int radius, Color color)
    {
        graphics.FillEllipse(new SolidBrush(color), new Rectangle(x, y, radius * 2,
radius * 2));
    }

    public void Vector(PointD start, PointD vector, Color penColor, float penWidth)
    {
        graphics.DrawLine(new Pen(new SolidBrush(penColor), penWidth), new
Point((int)start.X, (int)start.Y), new Point((int)(start.X + vector.X), (int)(start.Y
+ vector.Y)));
    }

    public void Points(MomentAufnahme[] points)
    {
        for(int i = 1; i < points.Length; i++)
        {
            try
            {
                graphics.DrawLine(new Pen(Brushes.Black, 1), new Point((int)points[i -
1].PlanetLocation.X, (int)points[i - 1].PlanetLocation.Y), new
Point((int)points[i].PlanetLocation.X, (int)points[i].PlanetLocation.Y));
            }
            catch
            {

            }
        }
    }
}

class Calculate
{
    /// <summary>
    /// Berechnet die Diagonale zweier beliebiger Punkte.
    /// </summary>
    /// <param name="x1">x-koordinate des ersten Punktes.</param>
    /// <param name="y1">y-koordinate des ersten Punktes.</param>
    /// <param name="x2">x-koordinate des zweiten Punktes.</param>
    /// <param name="y2">y-koordinate des zweiten Punktes.</param>
    /// <returns>Gibt die Diagonale zurück.</returns>
    public static double GetDiagonal(double x1, double y1, double x2, double y2)
    {
        double diagon = Math.Sqrt((Math.Pow(x1-x2,2.0)+Math.Pow(y1-y2,2.0)));
        return diagon;
    }

    /// <summary>
    /// Gibt den nächsteliegenden Punkt auf eine linie wieder.
    /// </summary>
    /// <param name="startX">x-koordinate des Anfangs der Linie.</param>
    /// <param name="startY">y-koordinate des Anfangs der Linie.</param>
    /// <param name="endX">x-koordinate des Endes der Linie.</param>
    /// <param name="endY">y-koordinate des Endes der Linie.</param>
```

```
    /// <param name="radius">Die weite des nächsten Punktes vom Startpunkt.</param>
    /// <returns>Gibt eine double-Array zurück. double[0] = x-koordinate, double[1] =
y-koordiante</returns>
    public static double[] GetNextPoint(double startX, double startY, double endX,
double endY, double radius)
    {
        double[] back = new double[2];

        double x = endX - startX;
        double y = endY - startY;

        if (y == 0 && x == 0)
        {
            return new double[] { startX, startY };
        }

        double rad = Math.Atan(y / x);
        double cos = Math.Cos(rad);
        double sin = Math.Sin(rad);

        if (x >= 0)
        {
            back[0] = (cos * radius) + startX;
            back[1] = (sin * radius) + startY;
            return back;
        }

        back[0] = -(cos * radius) + startX;
        back[1] = -(sin * radius) + startY;
        return back;
    }

    public static double GetDensity(double radius, double mass)
    {
        double volume = (4 / 3) * Math.PI * Math.Pow(radius, 2.0);
        return (mass / volume);
    }
}

class Manager
{

    //Points
    public Manager()
    {
    }

    private MomentAufnahme CalculateAll(double t,double zeitStep, PointD
vectorFromUser, PointD planetLocation, PointD starLocation, double planetMass, double
starMass)
    {
        MomentAufnahme momentAufnahme = new MomentAufnahme();

        double tmp = planetMass * starMass * (Math.Pow(6.67428, -11));
        double force = (tmp /
Math.Pow(Calculate.GetDiagonal(planetLocation.X,planetLocation.Y,starLocation.X,starLo
cation.Y), 2.0));

        momentAufnahme.Force = force;

        //F = m * a
        //a = F / m
        double a = force / planetMass;

        momentAufnahme.A = a;

        //v = a * t
        double vToStar = a * t;

        momentAufnahme.VToStar = vToStar;

        double[] endPointOfVectorToStar = Calculate.GetNextPoint(planetLocation.X,
planetLocation.Y, starLocation.X, starLocation.Y, vToStar);

        momentAufnahme.VectorToStar = new PointD(endPointOfVectorToStar[0] -
planetLocation.X, endPointOfVectorToStar[1] - planetLocation.Y);

        momentAufnahme.VectorResult = new PointD(vectorFromUser.X  + planetLocation.X
+ momentAufnahme.VectorToStar.X, vectorFromUser.Y + planetLocation.Y +
momentAufnahme.VectorToStar.Y);
```

```
        double vResult = Calculate.GetDiagonal(planetLocation.X, planetLocation.Y,
momentAufnahme.VectorResult.X, momentAufnahme.VectorResult.Y);

        momentAufnahme.VResult = vResult;

        //v = s / t
        //s = v * t
        double s = vResult * zeitStep;

        double[] newPoints = Calculate.GetNextPoint(planetLocation.X,
planetLocation.Y, momentAufnahme.VectorResult.X, momentAufnahme.VectorResult.Y, s);

        momentAufnahme.PlanetLocation = new PointD(newPoints[0], newPoints[1]);

        return momentAufnahme;
    }

    public List<MomentAufnahme> CreatePoints(int length, double zeitStep, PointD
vectorFromUser, PointD planetLocation, PointD starLocation, double planetMass, double
starMass)
    {
        List<MomentAufnahme> aufnahmen = new List<MomentAufnahme>();

        aufnahmen.Add(new MomentAufnahme());
        aufnahmen[0].PlanetLocation = new PointD(planetLocation.X, planetLocation.Y);

        for (double i = zeitStep; i < length; i += zeitStep)
        {
            aufnahmen.Add(CalculateAll(i, zeitStep, vectorFromUser,
aufnahmen[aufnahmen.Count - 1].PlanetLocation, starLocation, planetMass, starMass));
        }

        return aufnahmen;
    }
}

public partial class Simulation : Form
{
    BufferedGraphics buffGraphics;
    Draw drawManager;
    bool movePlanet;
    MouseEventArgs mouseDown;
    Point beforePlanetLocation;

    //new
    private Manager manager;
    private List<MomentAufnahme> aufnahmen;
    private PointD vectorFromUser;

    private PointD planetLocation;
    private PointD starLocation;

    private double planetMass;
    private double starMass;
    private double planetRadius;
    private double starRadius;

    private double timeStep;
    private int maxTime;
    private bool isSimulation;

    //darstellung
    private bool showLines;


    public Simulation()
    {
        InitializeComponent();

buffGraphics=BufferedGraphicsManager.Current.Allocate(panelMain.CreateGraphics(),panel
Main.ClientRectangle);
        drawManager=new Draw(buffGraphics.Graphics);
        manager = new Manager();
        aufnahmen = new List<MomentAufnahme>();

        timerSimulation.Enabled = false;

        trackBar1.Maximum = 0;
        trackBar1.Enabled = false;
        checkSimulation.Enabled = false;
        btnStopSimu.Enabled = false;
```

```
        printDocument.DocumentName = "Kosmischer Tanz";
        printDocument.PrintPage += new
System.Drawing.Printing.PrintPageEventHandler(printDocument_PrintPage);

        //init
        vectorFromUser = new PointD(20, 10);
        planetLocation = new PointD(0,0);
        starLocation = new PointD(panelMain.ClientSize.Width / 2,
panelMain.ClientSize.Height / 2);
        starMass = 1000;
        planetMass = 1000;
        planetRadius = 20;
        starRadius = 20;

        showLines = true;

        timeStep = 0;
        txtTimeStep.Text = "0";
        maxTime = 0;
        txtmaxTime.Text = "0";
        isSimulation = false;
        UpdateTxt();
    }

    private void panelMain_Paint(object sender, PaintEventArgs e)
    {
        DrawIt();
    }

    public void DrawIt()
    {
        drawManager.Clear(Color.WhiteSmoke);

        drawManager.Ellipse((int)(planetLocation.X - planetRadius), (int)
(planetLocation.Y - planetRadius), (int)planetRadius, Color.Yellow);
        drawManager.Ellipse((int)(starLocation.X - starRadius), (int)(starLocation.Y -
starRadius), (int)starRadius, Color.Black);
        drawManager.Vector(planetLocation, vectorFromUser, Color.Blue, 2);
        if (showLines)
        {
            drawManager.Points(aufnahmen.ToArray());
        }

        buffGraphics.Render();
    }
    void printDocument_PrintPage(object sender, PrintPageEventArgs e)
    {
        e.Graphics.DrawRectangle(new
Pen(Color.Black,2),0,0,panelMain.Size.Width,panelMain.Size.Height);

        Draw drawPrint = new Draw(e.Graphics);

        drawPrint.Ellipse((int)(planetLocation.X - planetRadius), (int)
(planetLocation.Y - planetRadius), (int)planetRadius, Color.Yellow);
        drawPrint.Ellipse((int)(starLocation.X - starRadius), (int)(starLocation.Y -
starRadius), (int)starRadius, Color.Black);
        drawPrint.Vector(planetLocation, vectorFromUser, Color.Blue, 2);
        if (showLines)
        {
            drawPrint.Points(aufnahmen.ToArray());
        }
    }

    private void panelMain_MouseMove(object sender, MouseEventArgs e)
    {
        lblSMouseX.Text = "X Koordinate der Maus: " + Convert.ToString(e.X);
        lblSMouseY.Text = "Y Koordinate der Maus: " + Convert.ToString(e.Y);

        if (movePlanet)
        {
            if (e.X > 0 && e.X <
panelMain.Size.Width&&e.Y>0&&e.Y<panelMain.Size.Height)
            {
                planetLocation.X = beforePlanetLocation.X + (e.X - mouseDown.X);
                planetLocation.Y = beforePlanetLocation.Y + (e.Y - mouseDown.Y);
                DrawIt();
                UpdateTxt();
            }
        }
    }
    private bool IsPlanetClicked(MouseEventArgs e)
```

```
    {
        if(planetLocation.X+planetRadius>e.X&&
            planetLocation.X-planetRadius<e.X&&
            planetLocation.Y+planetRadius>e.Y&&
            planetLocation.Y-planetRadius<e.Y)
        {
            return true;
        }
        return false;

    }
    private void panelMain_MouseDown(object sender, MouseEventArgs e)
    {
        if (!isSimulation)
        {
            if (IsPlanetClicked(e) && e.Button == MouseButtons.Left)
            {
                movePlanet = true;
                mouseDown = e;
                beforePlanetLocation = new Point((int)planetLocation.X,
(int)planetLocation.Y);
            }
            if (e.Button == MouseButtons.Right)
            {
                vectorFromUser.X = e.X - planetLocation.X;
                vectorFromUser.Y = e.Y - planetLocation.Y;
                UpdateTxt();
                DrawIt();
            }
        }
    }
    private void panelMain_MouseUp(object sender, MouseEventArgs e)
    {
        movePlanet = false;
    }

    private void UpdateTxt()
    {
        if (txtStarMass.Text != Convert.ToString(starMass))
            txtStarMass.Text = Convert.ToString(starMass);
        if (txtStarRadius.Text != Convert.ToString(starRadius))
            txtStarRadius.Text = Convert.ToString(starRadius);
        if (txtStarDensity.Text != Convert.ToString(Calculate.GetDensity(starRadius,
starMass)))
            txtStarDensity.Text = Convert.ToString(Calculate.GetDensity(starRadius,
starMass));
        if (txtPlanetMass.Text != Convert.ToString(planetMass))
            txtPlanetMass.Text = Convert.ToString(planetMass);
        if (txtPlanetRadius.Text != Convert.ToString(planetRadius))
            txtPlanetRadius.Text = Convert.ToString(planetRadius);
        if (txtPlanetDensity.Text !=
Convert.ToString(Calculate.GetDensity(planetRadius, planetMass)))
            txtPlanetDensity.Text =
Convert.ToString(Calculate.GetDensity(planetRadius, planetMass));
        if (txtPlanetX.Text != Convert.ToString(planetLocation.X))
            txtPlanetX.Text = Convert.ToString(planetLocation.X);
        if (txtPlanetY.Text != Convert.ToString(planetLocation.Y))
            txtPlanetY.Text = Convert.ToString(planetLocation.Y);

        if (txtPlanetSpeed.Text != Convert.ToString(Calculate.GetDiagonal(0, 0,
vectorFromUser.X, vectorFromUser.Y)))
            txtPlanetSpeed.Text = Convert.ToString(Calculate.GetDiagonal(0, 0,
vectorFromUser.X, vectorFromUser.Y));

        if (txtPlanetDistance.Text !=
Convert.ToString(Calculate.GetDiagonal(planetLocation.X, planetLocation.Y,
starLocation.X, starLocation.Y)))
            txtPlanetDistance.Text =
Convert.ToString(Calculate.GetDiagonal(planetLocation.X, planetLocation.Y,
starLocation.X, starLocation.Y));

    }

    private void checkSimulation_CheckedChanged(object sender, EventArgs e)
    {
        if (checkSimulation.Checked)
        {
            timerSimulation.Start();
        }
        else
        {
```

```
            timerSimulation.Stop();
        }
    }
    private void timerSimulation_Tick(object sender, EventArgs e)
    {
        trackBar1.Value ++;
    }


    private void trackBar1_ValueChanged(object sender, EventArgs e)
    {
        planetLocation.X = aufnahmen[trackBar1.Value].PlanetLocation.X;
        planetLocation.Y = aufnahmen[trackBar1.Value].PlanetLocation.Y;
        lblTrackBar.Text = trackBar1.Value * timeStep + " sek";
        txtA.Text= aufnahmen[trackBar1.Value].A.ToString();
        txtSpeedToStar.Text=aufnahmen[trackBar1.Value].VToStar.ToString();
        txtResultingSpeed.Text=aufnahmen[trackBar1.Value].VResult.ToString();
        txtForceToStar.Text = aufnahmen[trackBar1.Value].Force.ToString();

        DrawIt();
    }

    private void btnCalcualteVerlauf_Click(object sender, EventArgs e)
    {
        this.aufnahmen = manager.CreatePoints(maxTime, timeStep, vectorFromUser,
planetLocation, starLocation, planetMass, starMass);
        isSimulation = true;
        trackBar1.Maximum = aufnahmen.Count - 1;
        trackBar1.Enabled = true;
        checkSimulation.Enabled = true;
        txtPlanetMass.ReadOnly = true;
        txtPlanetRadius.ReadOnly = true;
        txtPlanetSpeed.ReadOnly = true;
        txtPlanetX.ReadOnly = true;
        txtPlanetY.ReadOnly = true;
        txtStarMass.ReadOnly = true;
        txtStarRadius.ReadOnly = true;

        btnStopSimu.Enabled = true;
        btnCalcualteVerlauf.Enabled = false;

        txtmaxTime.ReadOnly = true;
        txtTimeStep.ReadOnly = true;

        DrawIt();
    }
    private void btnStopSimulation_Click(object sender, EventArgs e)
    {
        timerSimulation.Stop();
        checkSimulation.Checked = false;
        isSimulation = false;
        trackBar1.Maximum = 0;
        trackBar1.Enabled = false;
        checkSimulation.Enabled = false;
        txtPlanetMass.ReadOnly = false;
        txtPlanetRadius.ReadOnly = false;
        txtPlanetSpeed.ReadOnly = false;
        txtPlanetX.ReadOnly = false;
        txtPlanetY.ReadOnly = false;
        txtStarMass.ReadOnly = false;
        txtStarRadius.ReadOnly = false;

        txtmaxTime.ReadOnly = false;
        txtTimeStep.ReadOnly = false;


        btnStopSimu.Enabled = false;
        btnCalcualteVerlauf.Enabled = true;
    }

    private void druckenToolStripMenuItem_Click(object sender, EventArgs e)
    {
        printDialog.Document = printDocument;
        if (printDialog.ShowDialog() == DialogResult.OK)
        {
            printDocument.Print();
        }
    }
    private void druckvorschauToolStripMenuItem_Click(object sender, EventArgs e)
    {
        printPreviewDialog1.Document = printDocument;
```

```
    printPreviewDialog1.ShowDialog();
}


private void ExitToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}

private void txtStarMass_TextChanged(object sender, EventArgs e)
{
    try
    {
        starMass = double.Parse(txtStarMass.Text);
    }
    catch
    {

    }
    UpdateTxt();
}
private void txtStarRadius_TextChanged(object sender, EventArgs e)
{
    try
    {
        starRadius = double.Parse(txtStarRadius.Text);
    }
    catch
    {

    }
    UpdateTxt();
    DrawIt();
}
private void txtPlanetMass_TextChanged(object sender, EventArgs e)
{
    try
    {
        planetMass = double.Parse(txtPlanetMass.Text);
    }
    catch
    {

    }
    UpdateTxt();
}
private void txtPlanetRadius_TextChanged(object sender, EventArgs e)
{
    try
    {
        planetRadius = double.Parse(txtPlanetRadius.Text);
    }
    catch
    {

    }
    DrawIt();
    UpdateTxt();
}
private void txtPlanetX_TextChanged(object sender, EventArgs e)
{
    try
    {
        planetLocation.X = double.Parse(txtPlanetX.Text);
    }
    catch
    {

    }
    DrawIt();
}
private void txtPlanetY_TextChanged(object sender, EventArgs e)
{
    try
    {
        planetLocation.Y = double.Parse(txtPlanetY.Text);
    }
    catch
    {

    }
```

```
            DrawIt();
        }

        private void txtmaxTime_TextChanged(object sender, EventArgs e)
        {

            try
            {
                maxTime = int.Parse(txtmaxTime.Text);
            }
            catch
            {

            }
        }
        private void txtTimeStep_TextChanged(object sender, EventArgs e)
        {
            try
            {
                timeStep = double.Parse(txtTimeStep.Text);
            }
            catch
            {

            }
        }
    }
```