Dokumentation Abgabe 1

Gruppe B7

Annalena Diederich,

Jordan Grant

Inhalt dieses Dokumentes:

Aspekte der Abgabe im Bezug auf die Simulationspipeline

Die Kugel bewegt sich in einer beliebigen Richtung

Die Grundlage der vorliegenden Simulation soll eine sich bewegenden Kugel sein.

Um diese Umzusetzen sind wir die Schritte der Simulationspipeline durchgegangen.

Im ersten Schritt, der **Modellierung** haben wir einen Raum erstellt indem eine Kugel platziert wird. Auf diese sollten verschiedene Parameter einwirken können.

Diese Parameter wurden im zweiten Schritt, der **Berechnung**, eingefügt. Hierbei handelt es sich um einfache Formeln wie Geschwindigkeit oder Einwirkung von Schwerkraft.

Nachdem diese Schwerpunkte alle feststanden fingen wir an mit dem dritten Schritt der Simulationspipeline, der **Implementierung**. Diese wurde in einem in C# programmiertem Projekt umgesetzt.

Dieses Projekt ist ausführbar und soll die Idee einer sich bewegenden Kugel für den Nutzer verdeutlichen, welches der vierte Schritt, die **Visualisierung**, erfüllt.

Um dies in der **Validierung**, dem fünften Schritt zu überprüfen haben wir das Programm ausgeführt und

Sobald die Ergebnisse die Anforderungen erfüllten konnten wir diese fertig in das Modell **Einbetten**. Somit wird auch der letzte Schritt der Simulationspipeline erfüllt.

Eine oder mehrere Beschleunigungen können wirken (z.B. Gravitation und Wind).

In dem zweiten Aspekt wird versucht zu verdeutlichen wie eine Kugel beschleunigt wird. Wichtig ist dies, da es keine Konstante Bewegung gibt. Verschiedene Faktoren werden einen Einfluss ausüben, der verdeutlicht werden muss.

In der **Modellierung** wird erkannt, dass Gravitation zum Beispiel Einfluss auf die Bewegung einer Kugel besitzt.

Durch die Formeln zum **Berechnen** der Beschleunigung, können wir das in unserem Raum nachahnen.

Implementiert wird dies dann im Code. Dies führt auch zur **Visualisierung** bei.

Nachdem dies abgeschlossen ist, wurde das Programm ausgeführt um zu **validieren**, dass die Implementierung und Visualisierung infolge der Berechnungen erfolgt sind. Danach wurde dieser schritt ebenfalls in das Projekt **eingebettet**.

Empfohlen: Rollen auf der schiefen Ebene

Der nächste Aspekt soll das rollen auf der schiefen Ebene verdeutlichen. Hierbei ist in der **Modellierung** zu erkennen, dass es dabei vor allem um die Abhängigkeit der Kugel von der Erdanziehungskraft, der Gravitation, geht. Bei der **Berechnung** wird zur Vereinfachung die Gravitation mit 9,81N/kg, berechnet.

In unserem Fall haben wir diesen Teil in dieser Abgabe noch nicht **implementiert**. Eine fertige Ausarbeitung würde in der **Visualisierung** dann eine Kugel zeigen, welche auf einer Schiefen Ebene, zum Beispiel einem angewinkelten Brett, rollt. Zum **Validieren** der daten müsste eine Beschleunigung zu erkennen sein. Wenn dies fertig umgesetzt wird, könnte man es final ins Projekt **einbetten**.

Einstellbar: Startposition, Startbewegung, Beschleunigungseinflüsse

Zuletzt sollen einstellbare Parameter hinzugefügt werden. In der **Modellierung** würde man sich die GIU des Projektes vorstellen und überlegen welche Auswirkungen eine Anpassbarkeit dieser Parameter auf die Simulation haben würden. Beim **Berechnen** der Parameter sollten diese durch Variablen in den Formeln verwendet werden. Eine Startposition soll frei gewählt werden können.

Dies wird ebenfalls im Code **implementiert**. Hierbei wird der Fokus auf die **Visualisierung** in der GUI gelenkt. So wird in unserem Projekt eine Kugel frei vom Nutzer platziert. Dieser bekommt ein visuelles Feedback, wo sich die Kugel derzeit befindet. Die Beschleunigungseinflüsse kann er anhand von x- und y-Variablen einstellen. Dies wird ebenfalls die Startbewegung beeinflussen. Um diesen Schritt zu **validieren** muss wird

überprüft, ob die Kugel gesetzt wird und sich so bewegt, wie die Parameter es verlangen. Ebenfalls wird dieser Teil nachdem er funktioniert in das Projekt **eingebettet**.

Screenshots mit Erläuterung

```
//Calculate S
1 reference
public Vector2 NextDistance(Vector2 a,float t)
{
    Vector2 S = new Vector2();
    S = Position + Velocity * t + 0.5f * a* (float)Math.Pow(t,2);
    return S;
}
```

In diesem Teil des Codes wird die Strecke s, die der Ball hinter sich legen soll berechnet.

Die Strecke wird als ein Vektor dargestellt.

```
// Calculate V
1 reference
public void AddInfluenceVector (Vector2 a)
{

    // Berechnung der Geschwindigkeit des sich bewegenden Objektes

    Vector2 newV = new Vector2();

    newV = Velocity + a * Deltat;

    Influences.Add(newV);
}
```

Hier wird ein „Influence Vector“, also ein Vektor der Einfluss auf die Kugel ausüben soll, berechnet und einer Liste hinzugefügt.

```
// Velocity ------------------------------------------------------------------
//Calculate actual V
1 reference
public Vector2 NextVelocity()
{
    Vector2 influence = new Vector2();
    foreach (var item in Influences)
    {
        influence += item;
    }
    Vector2 truevelocity = Velocity + influence;
    return truevelocity;
}
```

Alle „Influence Vektoren“ werden zusammen berechnet um die tatsächliche Geschwindigkeit der Kugel zu berechnen.

```
public Ball(Vector2 startposition, Vector2 startvelocity, List<int> size,float t) {
    Velocity = startvelocity;
    Position = startposition;
    Size = size;
    Deltat = t;
    Influences = new List<Vector2>();
    //Erdanziehung in mm/S
    AddInfluenceVector(new Vector2(0,9810f));
}
```

Hier werden die Initialen Werte und Einstellungen der Kugel berechnet. Dabei ist zu beachten, dass die Gravitation fest zugeteilt und als eigener „Influence Vektor“ hinzugefügt wird.

```
public void BallUpdate()
{
    Vector2 v = NextVelocity();
    Vector2 s = NextDistance(Acceleration, 0.02f);

    Velocity = v;
    SetPosition((int)s.X,(int)s.Y);

}
```

Zuletzt werden die Werte einer Kugel zugeteilt und dann die alte Position und Geschwindigkeit auf den neuesten Stand gebracht.

Aufgaben

Annalena Diederich: GUI, Simulationsabbildung

Jordan Grant: Berechnungen und Element Kontrollen

Quellcode

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace PhysiksModell
{
   /// <summary>
   /// Interaction logic for App.xaml
   /// </summary>
   public partial class App : Application
   {
   }
}

using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;

namespace PhysiksModell
{
```

```
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
    }
}

using System.Windows;

[assembly: ThemeInfo(
    ResourceDictionaryLocation.None, //where theme specific resource dictionaries are
located
                                     //(used if a resource is not found in the page,
                                     // or application resource dictionaries)
    ResourceDictionaryLocation.SourceAssembly //where the generic resource dictionary is
located
                                          //(used if a resource is not found in the page,
                                          // app, or any theme specific resource dictionaries)
)]

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
```

```
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Windows.Threading;
using System.Numerics;
using System.Text.RegularExpressions;

namespace PhysiksModell
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class View : Window
    {
        // Timers---------------------------------------------------------------
        private readonly DispatcherTimer animationInterval = new DispatcherTimer();
        private readonly DispatcherTimer simulationInterval = new DispatcherTimer();
        // Flags----------------------------------------------------------------
        bool placementMode, placementClick, simActive;
        // List<UpdateBase> updates = new List<UpdateBase>();
        // Object Tracking Lists ------------------------------------------
        List<Ball> ballsInPlay;
        List<Rectangle> squaresInPlay;
        // GUI Elements ---------------------------------------------------
        float tbAccellerationValueX, tbAccellerationValueY;
        Canvas cPreviewLayer, cPlacementLayer;
        public View()
        {
            //Default Global Variable Settings//----------------------------
            placementMode = false;
```

```
    placementClick = false;
    simActive = false;
    ballsInPlay = new List<Ball>();
    squaresInPlay = new List<Rectangle>();
    //GUI Initialization//----------------------------------------

    InitializeComponent();
    //Grid Settup//-------------------------------------------------
    cPreviewLayer = new Canvas();
    cPlacementLayer = new Canvas();

    cSimArea.Children.Add(cPlacementLayer);
    cSimArea.Children.Add(cPreviewLayer);
    //Timer setup for animation Frames//----------------------------
    animationInterval.Interval = TimeSpan.FromMilliseconds(10);
    animationInterval.Tick += AnimationUpdate;
    animationInterval.Start();
    //Timer setup for Simulation cycles//---------------------------
    simulationInterval.Interval = TimeSpan.FromMilliseconds(20);
    simulationInterval.Tick += SimulationUpdate;

}
/*
public abstract class UpdateBase
{
    public abstract void Update();
}
public class PlaceballUpdate : UpdateBase
{
    private readonly View view;
```

```
    public PlaceballUpdate(View view)
    {
        this.view = view;
    }
    public override void Update()
    {
        view.cPreviewLayer.Children.Clear();

        int[] pos = view.CurrentMousePosition();
        bool oncanvas = view.WithinCanvas(pos[0], pos[1]);
        //Labels------------------------------------------------------
        if (oncanvas)
        {
            view.lplacementX.Content = "|X| :" + pos[0];
            view.lplacementY.Content = "|Y| :" + pos[1];
        }
        else
        {
            view.lplacementX.Content = "|X| : Invalid";
            view.lplacementY.Content = "|Y| : Invalid";
        }

        //Ellipses----------------------------------------------------
        if (oncanvas)
        {
            if (!view.WithinCanvas(pos[0] - 25, pos[1] - 25)) { return; }
            if (!view.WithinCanvas(pos[0] + 25, pos[1] + 25)) { return; }
            view.CreateBall(pos[0], pos[1]);
        }
    }
}
```

```
*/
private void AnimationUpdate(object sender, EventArgs e)
{
   /*
    lock (updates)
    {
       foreach (var item in updates)
       {
          item.Update();
       }
       updates.Clear();
    }
   */
   if (placementMode) {PlacePreview();}
   if (simActive) { PlaceSimObj(); }
}
private void SimulationUpdate(object sender, EventArgs e)
{

   if (simActive) {

       foreach (var item in ballsInPlay)
       {
          if (!WithinCanvas((int)item.Position.X, (int)item.Position.Y))
          {
            item.destroy = true;
          }
          else
          {
             item.BallUpdate();
          }
```

```
        }
    }
}
private void PlacePreview()
{
    cPreviewLayer.Children.Clear();

    int[] pos = CurrentMousePosition();
    bool oncanvas = WithinCanvas(pos[0], pos[1]);
    //Labels------------------------------------------------------
    if (oncanvas)
    {
        lplacementX.Content = "|X| :" + pos[0];
        lplacementY.Content = "|Y| :" + pos[1];
    }
    else
    {
        lplacementX.Content = "|X| : Invalid";
        lplacementY.Content = "|Y| : Invalid";
    }

    //Ellipses----------------------------------------------------
    if (oncanvas)
    {
        if (!WithinCanvas(pos[0] - 25, pos[1] - 25)) { return; }
        if (!WithinCanvas(pos[0] + 25, pos[1] + 25)) { return; }
        List<int> sizevalues = new List<int>();
        sizevalues.Add(50);

        Ball previewball = new Ball(new Vector2(pos[0], pos[1]),sizevalues);
```

```
            cPreviewLayer.Children.Add(CreateBall(previewball));
        }
    }
    private void PlaceInSim()
    {
        int[] pos = CurrentMousePosition();
        bool oncanvas = WithinCanvas(pos[0], pos[1]);

        if (oncanvas)
        {
            if (!WithinCanvas(pos[0] - 25, pos[1] - 25)) { return; }
            if (!WithinCanvas(pos[0] + 25, pos[1] + 25)) { return; }
            List<int> sizevalues = new List<int>();
            sizevalues.Add(50);
            Ball previewball =
                new Ball(
                    new Vector2(pos[0], pos[1]),
                    new Vector2(tbAccellerationValueX, tbAccellerationValueY),
                    sizevalues,0.02f);
            cPlacementLayer.Children.Add(CreateBall(previewball));
            ballsInPlay.Add(previewball);
        }

    }
    private void PlaceSimObj()
    {
        cPlacementLayer.Children.Clear();
        //Rectangles
        //Ellipses
        foreach (var item in ballsInPlay)
        {
```

```
            cPlacementLayer.Children.Add(CreateBall(item));
        }
    }
    private void bPlaceObject_Click(object sender, RoutedEventArgs e)
    {
        //Set flipflop for Update flag
        switch (placementMode)
        {
            case false:
                placementMode = true;
                break;
            case true:
                placementMode = false;
                break;
        }
    }
    private int[] CurrentMousePosition()
    {
        int[] mPosition = new int[] { (int)Mouse.GetPosition(cSimArea).X,
(int)Mouse.GetPosition(cSimArea).Y };


        return mPosition;
    }
    private void CreateBox(int xpos, int ypos)
    {
    }
    private Ellipse CreateBall(Ball ellipse)
    {


        // Creates a circle Of desired size at Desired Location
        // Creating Object----------------------------------------------
```

```
    Ellipse nextCircle = new Ellipse();
    // Characteristics of the Ellipse------------------------------
    nextCircle.Width = ellipse.Size[0];
    nextCircle.Height = ellipse.Size[0];
    nextCircle.Stroke = Brushes.White;
    nextCircle.Fill = Brushes.Blue;
    // Placement / Position----------------------------------------
    if (WithinCanvas((int)ellipse.Position.X, (int)ellipse.Position.Y))
    {
        Canvas.SetLeft(nextCircle, ellipse.Position.X - nextCircle.Width / 2);
        Canvas.SetTop(nextCircle, ellipse.Position.Y - nextCircle.Height / 2);
    }

    return nextCircle;

}
private void LeftClick(object sender, MouseButtonEventArgs e)
{
    if (!placementMode) { return; }
    PlaceInSim();
}
private void MouseMoveUpdate(object sender, MouseEventArgs e)
{
    /* if (!placementMode) { return; }
     PlaceballUpdate update = new PlaceballUpdate(new View());
     updates.Add(update);*/
}
private void bReset_Click(object sender, RoutedEventArgs e)
{
    cPlacementLayer.Children.Clear();
}
```

```
private void tbAccellerationNumCheck(object sender, TextCompositionEventArgs e)
{
   //Überprüfung mit einer Rgex ob der user eine Zahl eingegeben hat
   Regex regex = new Regex("[^0-9]+");
   e.Handled = regex.IsMatch(e.Text);
}
private void tbAccellerationX_GotFocus(object sender, RoutedEventArgs e)
{
   tbAccellerationX.Text = "0";
}
private void tbAccellerationY_GotFocus(object sender, RoutedEventArgs e)
{
   tbAccellerationY.Text = "0";
}
private void tbAccelerationY_KeyUp(object sender, KeyEventArgs e)
{
   try {
   tbAccellerationValueY = -(float)Convert.ToDecimal(tbAccellerationY.Text);
   }
   catch (Exception ex)
   {

   }

}
private void tbAccelerationX_KeyUp(object sender, KeyEventArgs e)
{
   try
   {
      tbAccellerationValueX = (float)Convert.ToDecimal(tbAccellerationX.Text);
   }
```

```
    catch (Exception ex)
    {

    }
}
private void bStartStop_Click(object sender, RoutedEventArgs e)
{
    switch (simActive)
    {
        case false:
            simActive = true;
            bStartStop.Content = "Stop";
            simulationInterval.Start();
            break;
        case true:
            simActive = false;
            bStartStop.Content = "Start";
            simulationInterval.Stop();
            break;
    }
}
private void RightClick(object sender, MouseButtonEventArgs e)
{
    if (placementMode)
    {
        placementMode = false;
        cPreviewLayer.Children.Clear();
    }
}
private bool WithinCanvas(int xCoord,int yCoord)
{
```

```
            if (xCoord > cSimArea.Width | xCoord < 0) { return false; }
            if (yCoord > cSimArea.Width | yCoord < 0) { return false; }
            return true;
        }

    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Numerics;

namespace PhysiksModell
{
    enum Material
    {
        Steel,
        Wood,
        Rubber
    }
    internal abstract class Object
    {
        private Material _material;
        private Vector2 _position;
        private List<int> _size;

        public void SetPosition(int X,int Y)
        {
```

```
        Position = new Vector2(X,Y);
    }
    public Vector2 Position { get => _position; set => _position = value; }
    internal Material Material1 { get => _material; set => _material = value; }
    public List<int> Size { get => _size; set => _size = value; }
}
internal abstract class DynamicObject : Object
{

    private Vector2 _velocity,_acceleration;
    private List<Vector2> _influences;
    private float _deltat;
    // Velocity ----------------------------------------------------------------
    //Calculate actual V
    public Vector2 NextVelocity()
    {
        Vector2 influence = new Vector2();
        foreach (var item in Influences)
        {
            influence += item;
        }
        Vector2 truevelocity = Velocity + influence;
        return truevelocity;
    }
    // Calculate V
    public void AddInfluenceVector (Vector2 a)
    {

        // Berechnung der Geschwindigkeit des sich bewegenden Objektes

        Vector2 newV = new Vector2();
```

```
        newV = Velocity + a * Deltat;

        Influences.Add(newV);
    }

    //Distance-----------------------------------------------------------------
    //Calculate S
    public Vector2 NextDistance(Vector2 a,float t)
    {
        Vector2 S = new Vector2();
        S = Position + Velocity * t + 0.5f * a* (float)Math.Pow(t,2);
        return S;
    }
    //GetSet-------------------------------------------------------------------
    public Vector2 Velocity { get => _velocity; set => _velocity = value; }
    public List<Vector2> Influences { get => _influences; set => _influences = value; }
    public Vector2 Acceleration { get => _acceleration; set => _acceleration = value; }
    public float Deltat { get => _deltat; set => _deltat = value; }
}
internal class Ball : DynamicObject
{
    public bool destroy;
    public Ball(Vector2 startposition, Vector2 startvelocity, List<int> size,float t) {

        Position = startposition;
        Size = size;
        Deltat = t;
        Influences = new List<Vector2>();
        //Erdanziehung in mm/S
        AddInfluenceVector(new Vector2(0,9810f));
```

```
            AddInfluenceVector(startvelocity);
            destroy = false;
        }
        public Ball(Vector2 startposition,List<int> size)
        {
            Position = startposition;
            Size = size;
        }
        public void BallUpdate()
        {
            Vector2 v = NextVelocity();
            Vector2 s = NextDistance(Acceleration, 0.02f);

            Velocity = v;
            SetPosition((int)s.X,(int)s.Y);

        }

    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PhysiksModell
{
    internal abstract class Update
    {
```

```
    }
    internal class PreviewBallUpdate:Update
    {
        Ball subject;
        public void BallUpdate(Ball ball)
        {
            subject = ball;
        }


    }

}
```