Bericht zur 01 Vorbereitung

Marcus Baetz, Andreas Kiauka, Robert Dziuba

02. November 2015

1 Aufgabenstellung

1.1

Es sollten zwei Programme angefertigt werden, die Bilder anzeigen und speichern.

1.1.1

Das erste Programm soll ein Bild von der Festplatte anzeigen können.

1.1.2

Das zweite Programm soll ein schwarzes Bild mit einer roten Diagonale anzeigen und speichern können. Bei beiden Programmen sollten wir mit JavaFX arbeiten.

1.2

Der zweite Teil der Aufgabenstellung bestand darin, dass wir vier mathematische Klassen implementieren sollten. Diese vier Klassen sollten eine Normale, einen Vektor, einen Punkt und eine 3x3-Matrix repräsentieren.

1.3

Der letzte Teil bestand darin, die mathematischen Klassen zu testen.

2 Lösungstrategien

Wir haben die Aufgabenstellung in 3 Teilaufgaben geteilt und jedes Gruppenmitglied hat eine dieser Aufgaben bearbeitet. Unsere Kommunikation während der Bearbeitung fand meistens via Facebook statt. Wir haben mit der Entwicklungsumgebung IntelliJ Idea gearbeitet, da die Einbindung von Git damit vergleichsweise einfach ist. Ansonsten haben wir unsere Vorlesungsaufzeichnungen des Moduls Mathematik 2 angeschaut.

3 Implementierung

3.1 Image Canvas

Diese Klasse zeichnet ein schwarzes Bild mit einer roten Diagonalen. Dazu wird die Klasse Canvas genutzt und die Methode paint überschrieben.

```
1  /**
2  * Fills the image with the defined color for each pixel.
3  */
4  private void render() {
5    writableImage = new WritableImage(imgWidth, imgHeight);
6    image.setImage(writableImage);
7    final PixelWriter myPixelWriter = writableImage.
         getPixelWriter();
8    final double wh = (double) imgHeight / (double) imgWidth;
9    for (int y = 0; y < imgHeight; y++) {
10     for (int x = 0; x < imgWidth; x++) {
11       if (y == ((int) ((wh) * x))) myPixelWriter.setColor(x
             , y, Color.RED);
12       else myPixelWriter.setColor(x, y, Color.BLACK);
13     }
14   }
15
16 }
```

3.2 Normal3

Die Klasse Normal3 stellt eine Normale dar. Sie hat Methoden für die mathematischen Grundrechenarten Addition, Subtraktion und Multiplikation. Zudem hat sie die Methode dot(), die das Skalarprodukt berechnet. Um diese Methoden zu implementieren reicht das Grundwissen über Vektoren aus dem Modul Mathematik 2

```
1  public double dot(final Vector3 v){
2    if(v==null) throw new IllegalArgumentException("Null as
         parameter");
3      return ((x*v.x) + (y*v.y) + (z*v.z));
4  }
```

3.3 Point3

Die Klasse Point3 stellt einen Punkt in einem dreidimensionalen Koordinatensystem dar. Sie hat ausschließlich Methoden zur Subtraktion und Addition, die dazu genutzt werden können, den Vektor zwischen zwei Punkten zu errechnen. Auch hier traten keine größeren Probleme auf.

```
1  public Point3 add(final Vector3 v){
2    if(v==null) throw new IllegalArgumentException("Null as
         parameter");
3    final double x = this.x + v.x;
4    final double y = this.y + v.y;
5    final double z = this.z + v.z;
6    return new Point3(x,y,z);
7  }
```

3.4 Vector3

Die Klasse Vector3 ist ähnlich aufgebaut wie die Klasse Normal3. Sie hat zusätzlich die Methoden x(), reflectedOn(), normalized() und asNormal(). Bei der Methode reflectedOn() haben wir zunächst verschieden Formeln gefunden, am Ende stellte sich jedoch heraus, dass es sich nur um eine unterschiedliche Interpretation von reflektieren handelte.

```
1  public Vector3 reflectedOn(final Normal3 n){
2  /*  if(n==null) throw new IllegalArgumentException("Null
       as parameter");
3    final Normal3 n1 = n.mul(2);
4    final double s = n.dot(this);
5    final Normal3 n2 = n1.mul(s);
6    final Vector3 v1 = this.mul(-1);
7    return v1.add(n2);*/
8    return (this.mul(-1)).add(n.mul(2).mul(n.dot(this)));
9  }
```

3.5 Mat3x3

Bei der Klasse Mat3x3, die eine zweidimensionale Matrix repräsentiert, irritierte am Anfang die Darstellung der Elemente als einzelne Attribute, später erleichterte dies jedoch die Rechnungen sehr.

```
1  public Mat3x3(final double m11, final double m21, final
       double m31,
2  final double m12, final double m22, final double m32,
3  final double m13, final double m23, final double m33) {
4    this.m11 = m11 + 0.0;
5    this.m12 = m12 + 0.0;
6    this.m13 = m13 + 0.0;
7    this.m21 = m21 + 0.0;
8    this.m22 = m22 + 0.0;
9    this.m23 = m23 + 0.0;
10   this.m31 = m31 + 0.0;
11   this.m32 = m32 + 0.0;
12   this.m33 = m33 + 0.0;
13
14   this.determinant = (m11*m22*m33)+ (m21*m32*m13)+ (m31*
       m12*m23)- (m13*m22*m31)-(m23*m32*m11)-(m33*m12*m21);
15 }
```

3.6 Tests

Die in den Akzeptanzkriterien vorgegebenen Rechnungen wurden mittels Junit getestet.

```
1  @Test
2  public void testAddNormal() throws Exception {
3    boolean worked= true;
4    final Random rnd = new Random();
5      for (int i = 0; i < 500; i++) {
6      final double x1= (rnd.nextDouble()*4000) - 2000;
7      final double y1= (rnd.nextDouble()*4000) - 2000;
8      final double z1= (rnd.nextDouble()*4000) - 2000;
9      final double x2= (rnd.nextDouble()*4000) - 2000;
10     final double y2= (rnd.nextDouble()*4000) - 2000;
11     final double z2= (rnd.nextDouble()*4000) - 2000;
12     final Vector3 v1 = new Vector3(x1,y1,z1);
13     final Normal3 n1 = new Normal3(x2,y2,z2);
14     final Vector3 v2 = new Vector3(x1+x2,y1+y2,z1+z2);
15     final Vector3 v3 = v1.add(n1);
16     if(!v2.equals(v3)) worked =false;
17   }
18   assertTrue(worked);
19 }
```

4 Probleme bei der Bearbeitung

Bei der Bearbeitung der Aufgabenstellung traten kaum Probleme auf.