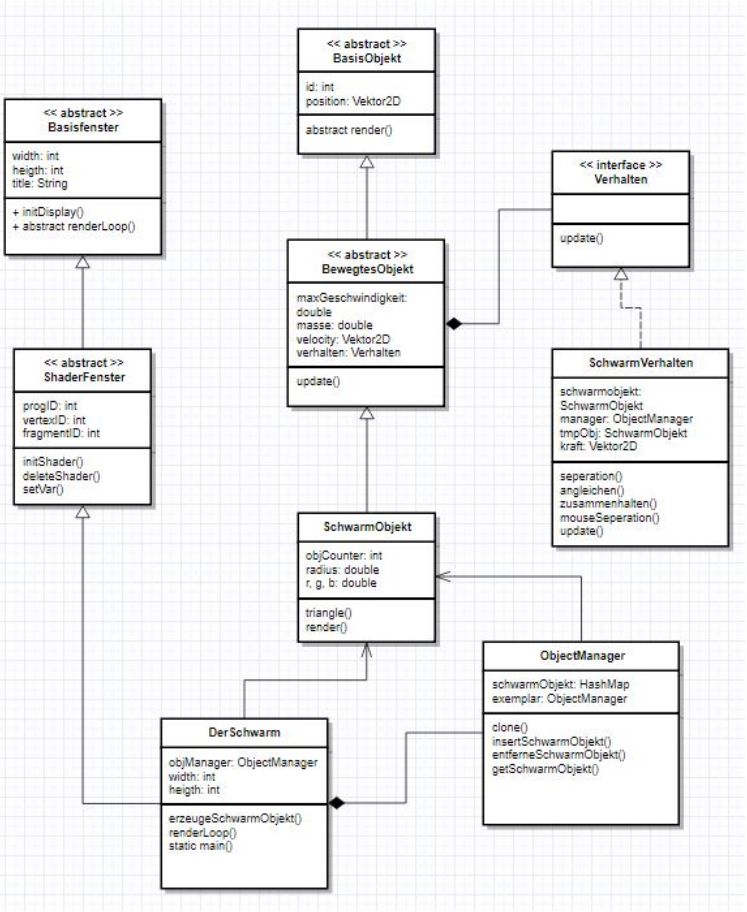
Dokumentation der Belegarbeit Computergrafik/Visualisierung II Sommersemester 2018

Christian Drossel (s75931)

June 26, 2018

Contents



1 Vektoren

Die Klasse Vektor dient als Grundlage für Berechnungen

1.1 Vektor

Die Klasse Vektor beschreibt einen Vektor aus der analytischen Geometrie und sie besitzt grundlegende Operationen zum Berechnen von Vektoren. Außerdem gibt es die Klassen Vektor2D und 3D die von der Klasse Vektor erben. Diese abgeleiteten Klassen stellen Vektoren des zwei- bzw. dreidimensionalen Raum dar und ebenfalls Operationen zur Berechnung.

1.2 Lineare Algebra

Die Klasse beinhaltet mathematische Operationen zur Berechnung der Vektoren. Außerdem besitzt sie die Besonderheit das sie statisch ist und somit nicht als Objekt erzeugt werden muss bzw. werden kann.

1.3 Überprüfe Überlauf

Diese Klasse dient als zusätzliche Hilfe und Schutz für Ausnahmebehandlungen, die bei den Berechnungen eintreten können.

2 Objekte

Das Objekt bildet die Agenten des Programmes die sich im Raum aufhalten. Sie besitzen ein verhalten.

2.1 BasisObjekt

Das BasisObjekt bildet die Grundlage. Sie ist abstract und besitzt nur Konstruktoren die protected sind. Von ihr sollen Subklassen erstellt werden aber es soll keine Instanz erzeugt werden. Der Konstruktor ist überladen und bekommt die Position als double oder als Vektor2D übergeben. Neben den Konstruktor weist die Klasse einen primitiven Datentyp auf und ein Objekt Vektor2D. Die id dient in der Klasse Verhalten eine wesentliche Rolle.

2.2 BewegtesObjekt

Die Klasse BewegtesObjekt ist abgeleitet von der BasisKlasse und ebenfalls abstract. Neben ihren überladenen Konstruktoren, bei denen die Position als double oder Vektor2D aber auch die Geschwindigkeit und die maximale Geschwindigkeit übergeben werden können, besitzt die Klasse zwei primitive Datentypen, die Masse und die maximale Geschwindigkeit als double Werte. Des Weiteren besitzt die Klasse noch zwei Objekte, ein Vektor2D mit dem Namen velocity und das Verhalten. Neben diesen Attributen besitzt die Klasse die Funktionen getvelocity, getMaxGeschwindigkeit und getMasse, die wie der Name es schon vermuten lässt getter Funktionen sind mit der man die Attribute velocity, maxGeschindigkeit und masse abrufen kann. Um das Verhalten zu setzen, besitzt die Klasse auch eine setVerhalten Funktion. Der update Funktion wird die Höhe und weite übergeben, die dann das verhalten "updaten" soll.

2.3 SchwarmObjekt

SchwarmObjekt ist eine von BewegtesObjekt abgeleitete Klasse und repräsentiert ein einzelnes Objekt des Schwarms. Der Konstruktor ist ebenfalls überladen und beinhaltet neben den Aufruf des Superkonstruktors auch die Übergabe der RGB Farbwerte, des Radius und der id. um die Position, id oder den Radius abzufragen besitzt die Klasse getter Methoden. Mit der Render-Funktion wird das Objekt anschließend gerendert, dazu werden die r, g, b Farbwerte und die Position verwendet.

2.4 ObjectManager

Die Klasse ObjectManager ist als Singleton-Pattern realisiert und dient der Verwaltung des erzeugten SchwarmObjektes. Sie beinhaltet eine HashMap zur Verwaltung der SchwarmObjekte. Zusätzlich, neben den Konstruktor gibt es Funktionen für das Hinzufügen und Entfernen von Objekten, sowie getSchwarmObjekt und getSchwarmobjektSize das Schwarmobjekt mit der id oder die Größe des aktuellen SchwarmObjekts zurückgeben. Außerdem gibt es die Funktionen clone die es nicht erlaubt ein Objekt zu „clonen".

3 Verhalten

Das Verhalten beschreibt wie sich ein Objekt im Raum bewegt und reagiert.

3.1 Verhalten

Die Klasse Verhalten ist eine Interface Klasse die eine Funktion update beinhaltet. Somit kann durch ein Strategy-Pattern wie bei dem SchwarmObjekt ein bestimmtes bzw. passendes Verhalten dem Objekt zugeordnet werden. Die Klasse dient also als Schnittstelle zwischen dem Verhalten und dem Objekt.

3.2 SchwarmVerhalten

SchwarmVerhalten erbt von der Klasse Verhalten und besitzt ein SchwarmObjekt, sowie ein ObjectManager und ein Vektor2D der die Kraft repräsentiert. Außerdem besitzt sie folgende Funktionen.

3.2.1 Seperation

...ist ein Verhalten, das auch bei natürlichen Schwärmen auftritt. Es beschreibt den Abstand der Objekte im Schwarm zueinander. Bei der Seperation wird der Abstand zwischen zwei Objekten berechnet, dessen Differenz genommen und anschließend aus der Differenz die Länge berechnet. die Länge wird auf einer variable length-, und die Differenz auf ein helpVec Vektor2D gespeichert. helpVec wird normalisiert und mit der length diverenziert. Am Ende wird der helpVec auf ein Vektor2D mit dem Namen SteeringForce addiert und von der Methode zurückgegeben.

```
1  public Vektor2D seperation() throws OverflowException, InfinityException,
       LengthException {
2      Vektor2D steeringForce = new Vektor2D(0, 0);
3      Vektor2D helpVec = new Vektor2D();
4      double length;
5      for (int i = 1; i <= manager.getSchwarmobjektSize(); i++) {
6          if (schwarmObjekt.getId() == i) {
7              continue;
8          }
9          tmpObj = manager.getSchwarmObjekt(i);
10         if(LineareAlgebra.euklSqrtDistance(tmpObj.getPosition(), schwarmObjekt.
               getPosition()) < (schwarmObjekt.getRadius() * sep)) {
11             if(schwarmObjekt.position.isNotEqual(tmpObj.position)) {

                                                    helpVec.setPosition(
                   LineareAlgebra.sub(schwarmObjekt.position, tmpObj.position));
12                 length = helpVec.length();
13                 helpVec.normalize();
14                 helpVec.div(length);
15                 steeringForce.add(helpVec);
16             }
17         }
18     }
19     return steeringForce;
20 }
```

3.2.2 angleichen

In der Literatur auch als alignment bezeichnet. Dieses Verhalten hat das Ziel, die Geschwindigkeit und Richtung an die anderen Nachbar-Objekte im Schwarm anzupassen. Dabei werden die umliegenden Objekte betrachtet und dessen abstand ermittelt. Ist der Abstand kleiner als der gesetzte Wert, dann passe die Geschwindigkeit an in dem der velocity Vektor vom Nachbarn auf das aktuelle Objekt drauf addiert wird.

Anschließend addiere den Counter um +1. Nachdem die „for“ Schleife sich durch alle Objekte durchgehangelt hat wird der Fall betrachtet, in dem der Counter größer 0 ist. In diesen Fall dividieren wir den counter mit unseren ermittelten Geschwindigkeitswert und subtrahieren ihn anschließend mit dem velocity Vektor des SchwarmObjektes.

```
1  public Vektor2D angleichen() throws OverflowException, InfinityException,
       LengthException {
2      Vektor2D steeringForce = new Vektor2D(0, 0);
3      int counter = 0;
4
5      for (int i = 1; i <= manager.getSchwarmobjektSize(); i++) {
6          if (schwarmObjekt.getId() == i) {
7              continue;
8          }
9          tmpObj = manager.getSchwarmObjekt(i);
10
11         if (LineareAlgebra.euklSqrtDistance(tmpObj.getPosition(), schwarmObjekt.
               getPosition()) < ang * ang) {
12             steeringForce.add(tmpObj.getVelocityObj());
13          counter++;
14         }
15     }
16     if (counter > 0) {
17         steeringForce.div(counter){
18         steeringForce.sub(schwarmObjekt.getVelocityObj());
19     }
20     return steeringForce;
21 }
```

3.2.3 zusammenhalten

Zusammenhalten beschreibt klar den Zusammenhalt des Schwarmes also der einzelnen Objekte zueinander. Dazu werden, wie bei Angleichen und Seperation, die Umliegenden Objekte und deren Position betrachtet. Auch hier wird die Länge ermittelt und anschließend mit der Masse multipliziert und durch 1 gerechnet, dieser Vektor wird mit der Position multipliziert. Schließlich wird dieser neu ermittelte Vektor zurückgegeben. Auch hier wird der Fall, dass der Counter größer null ist mit einbezogen. In diesen Fall subtrahieren wir den counter mit der ermittelten Kraft SteeringForce und subtrahieren ihn nochmal mit der Position des SchwarmObjektes.

```
1  public Vektor2D zusammenhalten() throws OverflowException, InfinityException,
       LengthException {
2          Vektor2D steeringForce = new Vektor2D(0, 0);
3          int counter = 0;
4          double lenght;
5          double[] tmp;
6          for (int i = 1; i <= manager.getSchwarmobjektSize(); i++) {
7              if (schwarmObjekt.getId() == i) {
8                  continue;
9              }
10             tmpObj = manager.getSchwarmObjekt(i);
11             tmp = tmpObj.getPosition();
12             lenght = LineareAlgebra.length(tmp);
13             LineareAlgebra.mult(tmp, (1 / (lenght * schwarmObjekt.getMasse())));
```

```
14              steeringForce.add(tmp);
15              counter++;
16          }
17          if (counter > 0) {
18              steeringForce.div(counter);
19              steeringForce.sub(schwarmObjekt.getPosition());
20          }
21          return steeringForce;
22  }
```

3.2.4 mouseSeperation

Bei der mouseSeperation wird die Richtung, der Radius und die Höhe übergeben. Die Mausposition wird auf einem double Array gespeichert, wobei hier schon die Höhe - Y Position der Maus gerechnet wird. Als nächstes wird betrachtet ob der Abstand von der Position zu Maus mit dem übergebenen Radius übereinstimmt, tut er das wird weiter überprüft, ob das Objekt sich schon auf der Position der Maus befindet. Wenn sich das Objekt nicht dort befindet, dann berechnet man die Länge aus der Maus Position und der Position des SchwarmObjektes, die man zuvor miteinander subtrahiert hat. Anschließend normalisiert man diesen ermittelten Vektor helpVec und multipliziert ihn nochmal mit der übergebenen Richtung und der ermittelten Länge. Zum Schluss gibt man diesen ermittelten Wert zurück.

```
1   public Vektor2D mouseSeperation(int richtung, int radius, int height) throws
        OverflowException, InfinityException, LengthException {
2       double length;
3       Vektor2D steeringForce = new Vektor2D(0, 0);
4       Vektor2D helpVec = new Vektor2D();
5       double[] mousePos = new double[]{Mouse.getX(), height - Mouse.getY()};
6       if (LineareAlgebra.euklDistance(schwarmObjekt.getPosition(), mousePos) <
            radius) {
7           if (schwarmObjekt.position.isNotEqual(mousePos)) {
8               helpVec.setPosition(LineareAlgebra.sub(schwarmObjekt.position,
                    mousePos));
9               length = helpVec.length();
10              helpVec.normalize();
11              helpVec.mult(richtung * length);
12              steeringForce.add(helpVec);
13          }
14      }
15      return steeringForce;
16  }
```

3.2.5 update

Mit der Methode update, wie der Name schon sagt wird das zum Schluss gezeigt Bild immer aktuell gehalten. Hier werden die Kräfte aus dem verhalten und einem Festen Wert multipliziert. Außerdem ist hier die Fallunterscheidung welche Maustaste man drückt. Dabei wird unterschieden ob rechte oder linke Maustaste. Mit dem drücken der linken Maustaste (Zeile 6) wird ein negativer Wert für die Richtung übergeben. Da sich das Objekt auf den Mauszeiger zu bewegen soll. Also muss die Distanz zwischen Objekt und Maus abnehmen. Wenn die rechte Maustaste gedrückt wird, soll sich das SchwarmObjekt entfernen, also wird ein positiver Wert an mouseSeperation übergeben.

```
1  @Override
2  public void update(int width, int height) throws OverflowException,
       InfinityException, LengthException {
3          kraft.add(LineareAlgebra.mult(seperation(), 10000));
4          kraft.add(LineareAlgebra.mult(angleichen(), 80));
5          kraft.add(LineareAlgebra.mult(zusammenhalten(), (0.2)));
6          if (Mouse.isButtonDown(0)) {
7              kraft.add(mouseSeperation(-5, 500, height));
8          }
9          if (Mouse.isButtonDown(1)) {
10             kraft.add(mouseSeperation(5, 200, height));
11         }
12         kraft.round();
13         kraft.div(schwarmObjekt.getMasse());
14         schwarmObjekt.getVelocityObj().add(kraft);
15
16         schwarmObjekt.getVelocityObj().kuerzen(schwarmObjekt.
               getMaxGeschwindigkeit());
17         schwarmObjekt.position.add(schwarmObjekt.getVelocityObj());
18
19
20         schwarmObjekt.position.setXPosition((schwarmObjekt.position.getX() +
               width) % width);
21         schwarmObjekt.position.setYPosition((schwarmObjekt.position.getY() +
               height) % height);
22     }
23 }
```

4 Anzeige

Die Anzeige ist verantwortlich für die Darstellung des Schwarmverhaltens.

4.1 BasisFenster

Die Klasse Basis Fenster ist abstract und bildet die Grundlage für die grafische Ausgabe der Objekte. Sie besitzt eine Höhe, eine Breite und den Titel des Fensters als Attribute. Als Methoden beinhaltet sie die Initialisierung eines LWJGL Displays und das Starten des Renderloops.

4.2 ShaderFenster

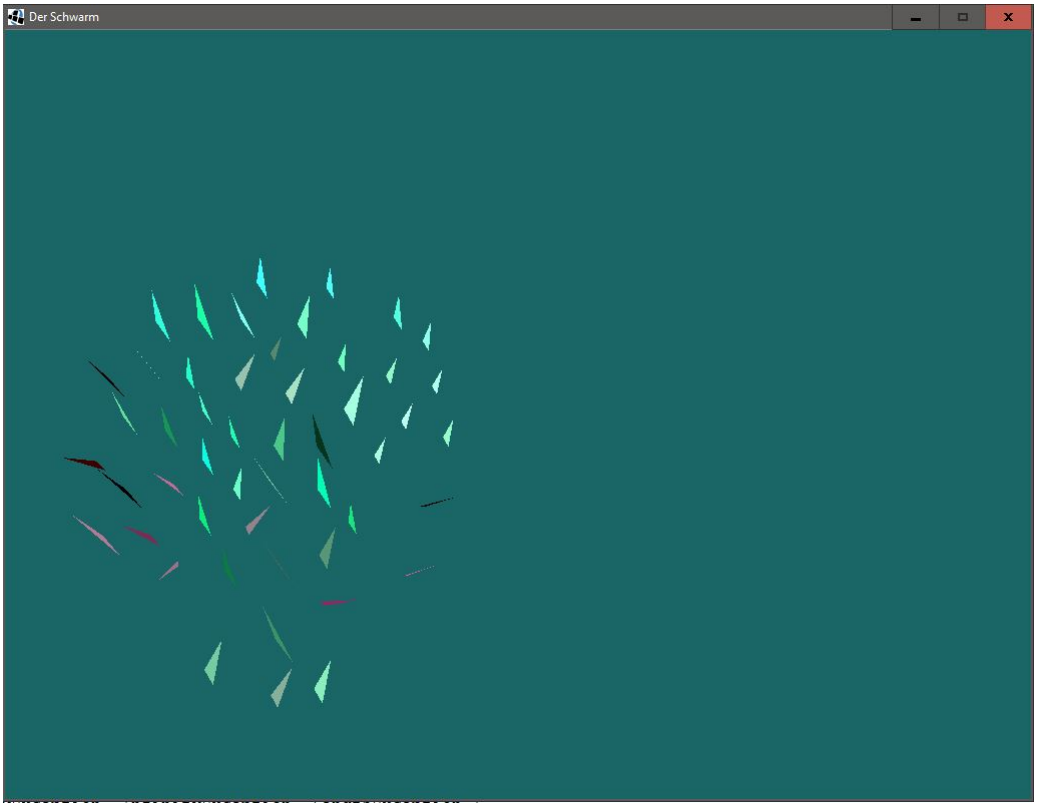
Die Klasse ShaderFenster dient dazu, die Berechnung der Anzeige der Objekte auf der Grafikkarte vorzunehmen. Dafür wird bei der Erstellung des Shaderprogramms ein Vertex- und ein Fragmentshader erzeugt. Deren Shadercode vorher als String festgelegt wurde und anschliesend an den glShaderSource übergeben wurde. In dem String wird Programmcode übergeben, der unter anderem die RGB Werte setzt und aus der Position die Farbe berechnet.

```
1  public void initShader() {
2      progID = glCreateProgram();
3      vertexID = glCreateShader(GL_VERTEX_SHADER);
4      fragmentID = glCreateShader(GL_FRAGMENT_SHADER);
```

```
5
6       String vertexCode = "uniform vec2 position;" +
7                   "varying vec3 color;" +
8                   "void main() {" +
9                   "gl_Position = ftransform();" +
10                  "float r = position.x - int(position.x);" +
11                  "float g = position.y - int(position.x);" +
12                  "float b = ( r + g ) / 2;" +
13                  "color.xyz = vec3(r,g,b);" +
14                  "}";
15          String fragmentCode = "varying vec3 color;" +
16                  "void main() {" +
17                  "gl_FragColor = vec4(color, 1);" +
18                  "}";
19
20      glShaderSource(vertexID, vertexCode);
21      glCompileShader(vertexID);
22      glShaderSource(fragmentID, fragmentCode);
23      glCompileShader(fragmentID);
24      glAttachShader(progID, vertexID);
25      glAttachShader(progID, fragmentID);
26      glLinkProgram(progID);
27      glValidateProgram(progID);
28      glUseProgram(progID);
```

4.3 DerSchwarm

In dieser Klasse wird im Grunde alles Zusammengebaut und erzeugt, denn sie besitzt die Main Methode. Sie erbt von der Klasse Shaderfenster und generiert ein Fenster. Ebenfalls erzeugt sie mit der Methode erzeugeSchwarmObjekt eine bestimmte Anzahl an SchwarmObjekte mit deren Position, Masse, Geschwindigkeit, Radius, Farbwerten und deren Verhalten. Außerdem setzt sie die SchwarmObjekte in den ObjektManager. Mit der Methode renderLoop wird unter anderem die Hintergrundfarbe gewählt, die Matrix gezeichnet und die SchwarmObjekte gezeichnet. In der Main Methode wird das ganze Programm gestartet durch den Aufruf new DerSchwarm().start(); .



5 Exception

Sind zu Deutsch Ausnahmebehandlungen. Sie sind gerade bei unkalkulierbaren Situationen besonders wichtig.

5.1 InfinityException

Diese Klasse hält Negative aber auch positive Unendlichkeit (im englischen Infinity) ab.

5.2 LengthException

Die Klasse LengthException hält falsche Längenangaben ab. Wenn zum Beispiel zwei Längen verglichen werden.

5.3 OverflowException

Diese Klasse fängt den Überlauf ab.