Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/Mathematik
Computergrafik/Visualisierung II

Schwarmverhalten

Submitted by: Marilena Fröhlich
Mathias Gewissen
Sebastian Mischke

Supervised by: Prof. Dr. Marco Block-Berlitz

Dresden, 27. Juni 2016

Inhaltsverzeichnis

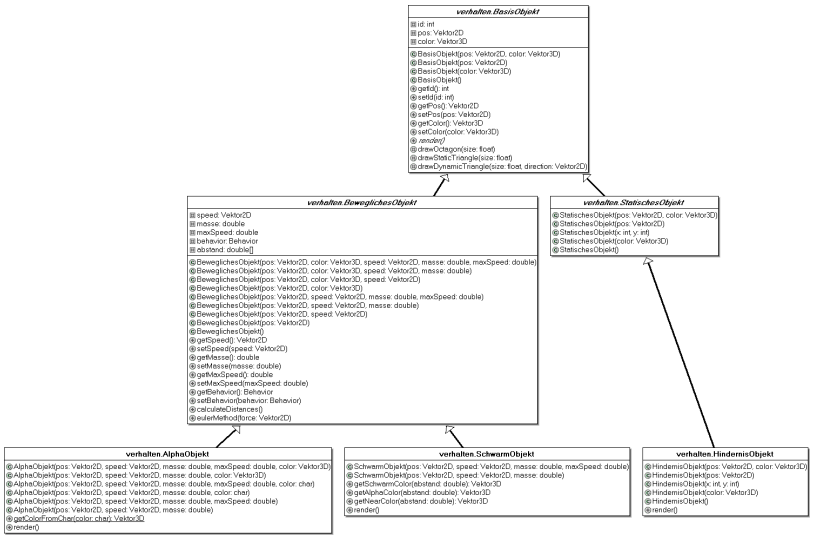

Abbildung 1: Objekte

1 Vektor

Vektoren dienen zur Speicherung von Daten und als Grundlage für Berechnungen.

1.1 Vektor

Die Klasse `Vektor` repräsentiert einen Vektor im mathematischen Sinne. Sie besitzt grundlegende Operationen, die auf Vektoren angewendet werden. Zusätzlich gibt es die Klassen `Vektor2D` und `Vektor3D`, welche von Vektor abgeleitet sind und Vektoren für den zwei- bzw. dreidimensionalen Raum darstellen

1.2 LineareAlgebra

Die Klasse `LineareAlgebra` beinhaltet mathematische Operationen, die für Vektoren ausgeführt werden.

2 Objekte

Im Programm gibt es verschiedene Objekte, die sich im Raum aufhalten und aufeinander wirken. Hierbei werden unterschiedliche Arten von Objekten unterschieden (siehe Abbildung 1).

2.1 BasisObjekt

Die Grundlage der Objekte bildet die Klasse BasisObjekt. Sie ist `abstract` und besitzt `protected` Konstruktoren. Es nicht gewollt ist, eine Instanz von ihr zu erzeugen, allerdings soll es möglich sein, Subklassen von ihr zu erstellen. Die Klasse hat mehrere Attribute - eine ID vom Datentyp `Integer`, eine Position vom Typ `Vektor2D` und eine Farbe vom Typ `Vektor3D` gespeichert. Der Konstruktor wird überladen und bekommt als Parameter Farbe und/oder Position übergeben. Die Zuordnung einer ID erfolgt gesondert im `ObjektManager` (siehe 2.6). Außerdem beinhaltet die Klasse `BasisObjekt` die abstrakte Methode `render`, die zum Anzeigen des Objektes überschrieben wird. Des Weiteren enthält sie Funktionen, die für das Erzeugen der Vertices und Anzeigen des Objektes aufgerufen werden können.

2.2 StatischesObjekt und BeweglichesObjekt

Die Klassen `StatischesObjekt` und `BeweglichesObjekt` sind von `BasisObjekt` abgeleitet und sind ebenfalls `abstract`. `StatischesObjekt` besitzt keine zusätzlichen Funktionen, lediglich Konstruktoren, die die Konstruktoren der Superklasse aufrufen. Ihre Aufgabe ist es, eine Unterscheidung zwischen sich bewegenden und festen Objekten zu ermöglichen. `BeweglichesObjekt` enthält mehrere zusätzliche Attribute.

- `speed` ist die aktuelle Geschwindigkeit des Objektes
- `masse` dient zur Anwendung des zweiten newtonschen Gesetzes $\overrightarrow{F} = m * \overrightarrow{a}$
- `maxSpeed` begrenzt die Geschwindigkeit eines Objektes
- `behavior` ist eine Instanz der Klasse `Behavior` (siehe 3.1) und das Verhalten des Objektes beinhaltet
- `abstand` ist ein Array mit den Abständen des Objektes zu allen anderen im `ObjektManager` (siehe 2.6) registrierten Objekten

Zu diesen Attributen beinhaltet die Klasse `BeweglichesObjekt` die Funktion `calculateDistances`, die das Array `abstand` mit den neu berechneten Werten befüllt. Die Funktion `eulerMethod` nimmt eine Kraft in Form eines `Vektor2D` als Übergabeparameter und wendet das explizite Euler-Verfahren auf das Objekt an.

2.3 SchwarmObjekt

`SchwarmObjekt` ist eine von `BeweglichesObjekt` abgeleitete Klasse und repräsentiert ein Subjekt des Schwarms. Die Konstruktoren beinhalten neben dem Aufruf des Superkonstruktors auch das Einfügen des Objektes in den `ObjektManager` (siehe 2.6). Außerdem wird hier das Attribut `behavior` mit einem Objekt der Klasse `SchwarmVerhalten` (siehe 3.3) initialisiert. Für das Anzeigen der `SchwarmObjekte` wird die Funktion `render` überschrieben. Darin wird zunächst `color` durch den Aufruf von `getSchwarmColor` geändert. Danach folgt der OpenGL Aufruf mit dem setzen der Farbe der Anzeige auf den im Objekt gespeicherten color Wert und anschließendem Ausführen von `drawDynamicTriangle`, welches ein gleichschenkliges Dreieck erzeugt, das in Richtung des eingegebenen Geschwindigkeitsvektors ausgerichtet ist.

2.4 AlphaObjekt

Auch die Klasse `AlphaObjekt` wurde von `BeweglichesObjekt` abgeleitet, jedoch enthält sie, im Unterschied zum `SchwarmObjekt`, neben Konstruktoren und dem überschriebenen `render` keine zusätzlichen Funktionen. Bereits im Konstruktor wird behavior ein neues Objekt von `AlphaVerhalten` (siehe 3.4) zugewiesen.

2.5 HindernisObjekt

Die Klasse `HindernisObjekt` wird von `StatischesObjekt` abgeleitet. Sie besitzt neben den Konstruktoren, welche die Konstruktoren der Superklasse verwenden, eine überschriebene Funktion `render`. In ihr wird die Farbe für die Anzeige auf schwarz gesetzt und ein regelmäßiges Achteck durch die Funktion `drawOctagon` an die Position des Objektes gezeichnet.

2.6 ObjektManager

Die Klasse `ObjektManager` ist als Singleton-Pattern realisiert und dient der Verwaltung der erzeugten Objekte. Sie beinhaltet drei Arrays, diese speichern jeweils Instanzen von `SchwarmObjekt`, `AlphaObjekt` und `HindernisObjekt`. Zusätzlich gibt es Funktionen für das Hinzufügen und Entfernen von Objekten, sowie das Überprüfen, ob ein Objekt bereits im `ObjektManager` vorhanden ist. Außerdem gibt es die Funktionen `update` und `render`. Diese rufen die `update` bzw. `render`-Funktion einzeln für jedes im `ObjektManager` befindliche Objekt auf.

3 Verhalten

Verhalten ist die Art, wie sich ein bewegliches Objekt im Raum bewegt und auf andere Objekte reagiert.

3.1 Behavior

`Behavior` ist ein Interface, welches die Funktion `update` bereitstellt. Damit wird sichergestellt, dass in den Klassen, die `Behavior` implementieren, die Funktion `update` aufgerufen werden kann.

3.2 BasisVerhalten

Die Klasse `BasisVerhalten` ist eine abstrakte Klasse, die `Behavior` implementiert. Zudem besitzt sie als Attribut ein Objekt der Klasse `BeweglichesObjekt` und die folgenden Funktionen.

3.2.1 Kohäsion

Jedes bewegliche Objekt hat ein festgelegtes Umfeld. Alle beweglichen Objekte, die sich in diesem Umfeld befinden, nehmen Einfluss auf das Objekt. Die Größe des Feldes wird dem Verhalten als Parameter übergeben. Für die Berechnung der Kohäsion wird für jedes im `ObjektManager` befindliche `SchwarmObjekt` überprüft, welche beweglichen Objekte sich innerhalb des Umfeldes befinden. Für diese Objekte wird anschließend die durchschnittliche Position berechnet. Die Differenz aus berechnetem Mittelwert und der Position des im Verhalten gespeicherten Objektes liefert die Kohäsion.

```
1  public Vektor getCohesion(double abstand) {
2      Vektor2D average = new Vektor2D();
3      int count = 0;
4      for (int i = 0; i < om.getObjectCount(); i++) {
5          if (obj.abstand[i] < abstand) {
6              average.add(om.getObject(i).pos);
7              count++;
8          }
9      }
10     return count == 0 ? average : average.div(count).sub(obj.pos);
11 }
```

3.2.2 Separation

Wie bei der Kohäsion nehmen nur bewegliche Objekte in einem bestimmten Umkreis eines Objektes Einfluss auf das Objekt. Von diesen Objekten wird der durchschnittliche Abstand zum Ausgangsobjekt berechnet. Dabei wird der Abstand vorher durch das Quadrat seiner Länge geteilt. Die Differenz aus berechnetem Mittelwert und der Position des im Verhalten gespeicherten Objektes liefert die Separation.

```
1  public Vektor2D getSeparation(double abstand) {
2      Vektor2D result = new Vektor2D();
3      for (int i = 0; i < om.getObjectCount(); i++) {
4          if (obj.id != i) {
5              Vektor2D dif = (Vektor2D) LineareAlgebra.sub(obj.pos, om.
                   getObject(i).pos);
6              if ((obj.abstand[i] < abstand) && (obj.abstand[i] > 0)) {
7                  result.add(dif.div(obj.abstand[i] * obj.abstand[i]));
8              }
9          }
10     }
11     return result;
12 }
```

3.2.3 Alignment

Für die Berechnung des Alignments wird die Durchschnittsgeschwindigkeit aller beweglichen Objekte, die sich in oben genanntem Umfeld (siehe 3.2.1) des Ausgangsobjektes befinden, berechnet.

```
1  public Vektor2D getAlignment(double abstand) {
2      Vektor2D average = new Vektor2D();
3      int count = 0;
4      for (int i = 0; i < om.getObjectCount(); i++) {
5          if (obj.abstand[i] < abstand) {
6              average.add(om.getObject(i).speed);
7              count++;
8          }
9      }
10     return count == 0 ? average : (Vektor2D) average.div(count);
11 }
```

3.2.4 Hindernisse-Separation

Die Hindernisse-Separation funktioniert wie die Separation. Allerdings werden zur Berechnung nur die statischen, nicht aber die beweglichen Objekte mit einbezogen.

```
1  public Vektor2D getObstacleSeparation(double abstand) {
2      Vektor2D result = new Vektor2D();
3      double diflength;
4      for (int i = 0; i < om.getObstacleCount(); i++) {
5          Vektor2D dif = (Vektor2D) LineareAlgebra.sub(obj.pos, om.getObstacle(
               i).pos);
6          if (((diflength = dif.lengthsquare()) < abstand * abstand) && (
               diflength > 0)) {
7              result.add(dif.div(diflength));
8          }
9      }
10     return result;
11 }
```

3.2.5 Alpha-Kohäsion

Die Alpha-Kohäsion funktioniert wie die Kohäsion. Allerdings werden zur Berechnung nur `AlphaObjekte` mit einbezogen.

```
1 public Vektor2D getAlphaCohesion(double abstand) {
2     Vektor2D result = new Vektor2D();
3     for (int i = 0; i < om.getAlphaCount(); i++) {
4         if (LineareAlgebra.manhattanDistance(obj.pos, om.getAlpha(i).pos) <
              abstand) {
5             result.add(LineareAlgebra.sub(om.getAlpha(i).pos, obj.pos));
6         }
7     }
8     return result;
9 }
```

3.3 SchwarmVerhalten

Die Klasse `Schwarmverhalten` besitzt die Funktion `getForce`, die die Kräfte von `getCohesion`, `getSeparation`, `getAlignment`, `getObstacleSeparation` und `getAlphaCohesion` aufaddiert und zurückgibt. Damit berechnet sie die Gesamtkraft die auf das Objekt wirken soll. Die Funktion `update` schreibt zuerst die Abstände zwischen dem Ausgangsobjekt und allen anderen beweglichen Objekten im `ObjektManager` in das Array `abstand`. Anschließend wird das explizite Euler-Verfahren auf die berechnete Gesamtkraft ausgefüht.

3.4 AlphaVerhalten

Die Klasse `Alphaverhalten` verhält sich ähnlich wie das `Schwarmverhalten`. Allerdings addiert `getForce` hier lediglich die Kräfte von `getSeparation` und `getObstacleSeparation`.

4 Anzeige

Die Anzeige ist verantwortlich für die Darstellung des Schwarmverhaltens.

4.1 BasisFenster

Die Klasse `BasisFenster` ist `abstract` und bildet die Grundlage für die grafische Ausgabe der Objekte. Sie besitzt eine Breite, eine Höhe und den Titel des Fensters als Attribute. Als Methoden beinhaltet sie die Initialisierung eines LWJGL Displays (siehe Abbildung 2) und das Starten des Renderloops.

4.2 WeltDesSchwarms

Die Klasse `WeltDesSchwarms` wird von `BasisFenster` abgeleitet und ist Startklasse des Projektes. Bei der Ausführung werden vorab zunächst sämtliche für den Programmablauf relevanten Objekte erstellt. Im Anschluss wird das Display erzeugt und der `renderLoop` gestartet. Die Funktion `renderLoop` löscht erst den Inhalt des Displays, erzeugt dann ein Shaderprogram in der Klasse Shader (siehe section 4.3), führt anschließend die Funktionen `update` und `render` des `ObjektManagers` aus und löscht schließlich den erzeugten Shader. Dieser Ablauf wird wiederholt, solange keine Schließanfrage für das Display aufgetreten ist.

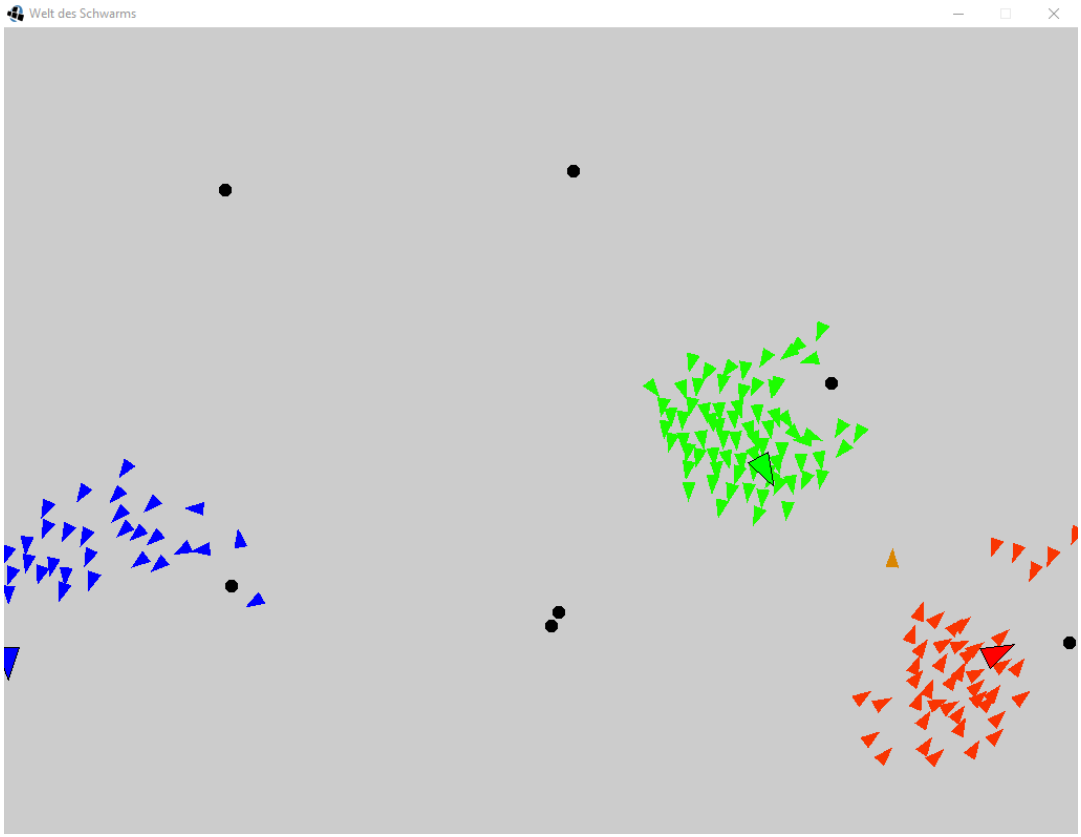

Abbildung 2: Display

4.3 Shader

Die Klasse `Shader` 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 wir aus einer externen Datei geladen, um anschließend die beiden Shader mit dem Shaderprogramm zu linken.

```
1  public void createShaderProgram() {
2      shaderProgramm = glCreateProgram();
3      vertexShader = glCreateShader(GL_VERTEX_SHADER);
4      fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
5      attachShader("src/verhalten/anzeige.vshader", vertexShader);
6      attachShader("src/verhalten/anzeige.fshader", fragmentShader);
7      glLinkProgram(shaderProgramm);
8      glValidateProgram(shaderProgramm);
9      glUseProgram(shaderProgramm);
10 }
```