SadEngine – Handbuch

1. Lol Engine was das?

Wenn man beim Programmieren etwas grafisch darstellen möchte, muss man letztlich auf irgendeinem Weg die Farbe von den Pixeln auf dem Monitor ändern. Jetzt könnte man natürlich anfangen, Algorithmen (oder riesige Switch-Cases) zu schreiben, die genau das tun. Das dauert dann aber potentiell etwas länger.

Jetzt gibt es zum Glück Frameworks. Das sind Programme, die Aufgaben erledigen, um die man sich nicht selbst kümmern will. Zum Beispiel das Berechnen der Farbwerte von Pixeln. Wenn man nur „UIs“ oder anders gesagt, zweidimensionale Bedienoberflächen braucht, bietet Java selbst mit Swing ein sehr gutes Framework hierfür an. Will man jedoch dreidimensionale Welten anzeigen, am besten noch bewegte Welten mit denen man interagieren kann, wie etwa in einem PC-Spiel, dann wird das mit Swing schon schwieriger, nicht zuletzt weil es die Pixelfarben mit der CPU und nicht mit der Grafikkarte berechnet. Hier kommen Engines ins Spiel. Sie sind darauf ausgelegt, dreidimensionale Bilder in Echtzeit zu berechnen und sind dabei im Besten Fall nicht viel schwieriger zu verwenden wie Swing.

Warum sollte man jetzt aber ausgerechnet mit der SadEngine arbeiten? Gut, wenn man zu Depressionen neigt, sollte man sich wohl besser eine andere Engine suchen, aber für alle anderen ist die SadEngine ein sehr leicht zu benutzendes 3D-Welt-zu-Pixelfarbe-Umrechnungs-Framework.

1. Hello World

public class Main extends SadEngine {

private Main() {

start();

}

public static void main(String[] args) {

new Main();

}

public void setup(Sadness sadness) {

SadContent c = sadness.getContent();

SadWindow w = sadness.getWindow();

SadInput i = sadness.getInput();

//Code...

}

public void update(Sadness sadness) {

SadContent c = sadness.getContent();

SadWindow w = sadness.getWindow();

SadInput i = sadness.getInput();

//Code...

}

}

Der wichtigste Teil des Programms sind die setup- und die update-Methode. Die setup-Methode wird aufgerufen, bevor das erste Frame berechnet wird. Danach wird standartmäßig 60 mal pro Sekunde die update-Methode ausgeführt.

Das entspricht etwa diesem Programmablauf:

setup(sadness);

while (true) {

update(sadness);

}

1. Erste Welt erstellen

Um etwas anzuzeigen, muss es in eine Welt (SadLevel) gesetzt werden. Außerdem muss eine Kamera (SadCamera) diese Welt filmen. Das Bildmaterial der Kamera muss dann auf einem Bildschirm (normalerweise SadFrame, hier das SadWindow w) angezeigt werden.

Um irgendein Objekt zu erstellen, muss die entsprechende create-Methode in SadContent aufgerufen werden.

Der Code kommt in die setup-Methode:

SadCamera camera = c.createCamera(“TestCamera”);

SadLevel level = c.createLevel(“TestLevel”);

w.setCamera(camera.getName());

camera.setLevel(level.getName());

Die Welt kann man sich auch als dreidimensionales Koordinatensystem vorstellen. Wenn die Kamera nicht gedreht wurde, entspricht die X-Achse der Breite, die Y-Achse der Höhe und die Z-Achse der Tiefe.

1. Erstes Objekt erstellen

Alles, was irgendwie in die Welt soll, um angezeigt, -gehört, oder sonstwie verarbeitet zu werden, kommt in ein Entity (SadEntity). In diesem Fall soll ein Modell (SadModel) angezeigt werden. Das Modell braucht eine Form, die durch ein Gitter aus Dreiecken (SadMesh) bestimmt wird. Ein Gitter ist eine Resource und muss aus einer Datei geladen werden. Deshalb muss beim Erstellen der Parameter new File(“Pfad“) angegeben werden. Ein Modell kann außerdem eine Textur (SadTexture) besitzen, die auch wieder eine Resource ist und einen Pfad braucht.

Der Code hierfür kommt auch in die setup-Methode:

SadEntity entity = c.createEntity(“TestEntity”);

SadModel model = c.createModel(“TestModel”);

SadMesh mesh = c.createMesh(“TestMesh”, new File(“C:/Pfad/zum/Model.obj”));

SadTexture texture = c.createTexture(“TestTexture”, new File(“C:/Pfad/zur/Textur.png”));

model.setMesh(mesh.getName()).setTexture(texture.getName());

entity.setModel(model.getName());

level.addEntity(entity);

Da die Kamera und das Objekt beide am Nullpunkt sind, muss die Kamera noch nach hinten versetzt werden, um das Objekt filmen zu können:

camera.getPosition.z(-5);

1. Methoden-Verkettung

Die meisten Methoden, die etwas verändern und nichts abfragen, geben das Objekt zurück, in dem sie aufgerufen wurden.

Beim SadModel geben beispielsweise die setMesh und die setTexture Methode das SadModel zurück. Daher ist es möglich,

model.setMesh(mesh.getName()).setTexture(texture.getName());

zu schreiben.

1. Positionables

Viele Dinge, zum Beispiel Kameras, Entitys oder Modelle sind Positionables. Das bedeutet ganz einfach, das sie eine Position, eine Rotation und eine Skalierung besitzen.

Die Position gibt mit drei Koordinaten die Position eines Objekts im Koordinatensystem an.

Die Rotation gibt mit drei Zahlen in Grad an, wie weit ein Objekt um welche Achse gedreht ist. Beträgt sie beispielsweise 90/0/0, ist das Objekt nach hinten umgekippt.

Die Skalierung streckt das Objekt mit drei Faktoren in jede Richtung. Hat ein Würfel die Skalierung 1/2/1, ist er ein Quader, der doppelt so hoch wie breit oder tief ist.

Man kann diese drei Eigenschaften mit den Methoden getPosition, getRotation und getScale aufrufen. Alle drei geben einen SadVector zurück.

1. Vektoren

Die wichtigsten Methoden dieses SadVectors sind die x(), y() und z() Methoden.

Ruft man sie ohne Parameter auf, geben sie den aktuellen x, y oder z Wert zurück. Übergibt man einen float, wird dieser als neuer Wert übernommen.

Die get(int index, float value) und set(int index, float value) Methoden machen genau dasselbe, nur gibt man hier über einen Index an, ob man auf die x, y oder z Koordinate zugreifen will. Mithilfe der set(float x, float y, float z) Methode kann man alle drei Werte auf einmal ändern, mithilfe der set(float scalar) Methode werden alle drei Werte auf den einen neuen Wert gesetzt. Genauso kann man auch einen float[], also einen float-Array übergeben.

Außerdem ist es möglich die Werte eines Vektors auf die eines anderen zu übertragen. Dazu gibt es die set(SadVector vector) Methode.

Es werden immer nur die Werte des Vektors verändert, in dem eine Methode aufgerufen wird. Ein Vektor, der als Parameter übergeben wird, wird also nicht verändert.

Sollen die Werte eines Vektors nicht verändert werden, kann mit der clone Methode eine Kopie erzeugt werden.

Mit der add Methode kann man einen Wert zum aktuellen addieren (+Rechnen) (zum Subtrahieren den neuen Wert erst negieren).

Entsprechend kann man mit der mul Methode einen Wert mit dem aktuellen multiplizieren (zum Dividieren zuerst 1 / neuer Wert rechnen).

Die Parameter bei den add und mul Methoden funktionieren wie die der set Methoden:

|  |  |  |
| --- | --- | --- |
| x(float value) | addX(float value) | mulX(float value) |
| y(float value) | addY(float value) | mulY(float value) |
| z(float value) | addZ(float value) | mulZ(float value) |
| set(int index, float value) | add(int index, float value) | mul(int index, float value) |
| set(float x, float y, float z) | add(float x, float y, float z) | mul(float x, float y, float z) |
| set(float scalar) | add(float scalar) | mul(float scalar) |
| set(float[] values) | add(float[] values) | mul(float[] values) |
| set(SadVector vector) | add(SadVector vector) | mul(SadVector vector) |

Mit der negate oder invert Methode wird das Vorzeichen von allen Werten umgekehrt, also mul(-1) ausgeführt.

Die getLength Methode gibt die Länge des Vektorpfeils zurück.

Die setLength Methode setzt die Länge des Vektors auf einen anderen Wert, ohne die Richtung zu ändern.

Entsprechend gibt es auch die Methoden addLength und mulLength.

Die Methode crossProduct errechnet das Kreuzprodukt zweier Vektoren, die Methode dotProduct entsprechend das Skalarprodukt.