Beleg CGII

Maxim Aisel, Paul Wollf, Dennis Hinterwimmer

27. Juni 2016

Inhaltsverzeichnis

1 Einleitung **3**

2 Aufbau **3**

3 Objekte **3**

4 Schwarmverhalten **4**
4.1 alignment 4
4.2 cohesion 5
4.3 seperation 5
4.4 follow 5

5 Renderloop und Fragmentshader **6**

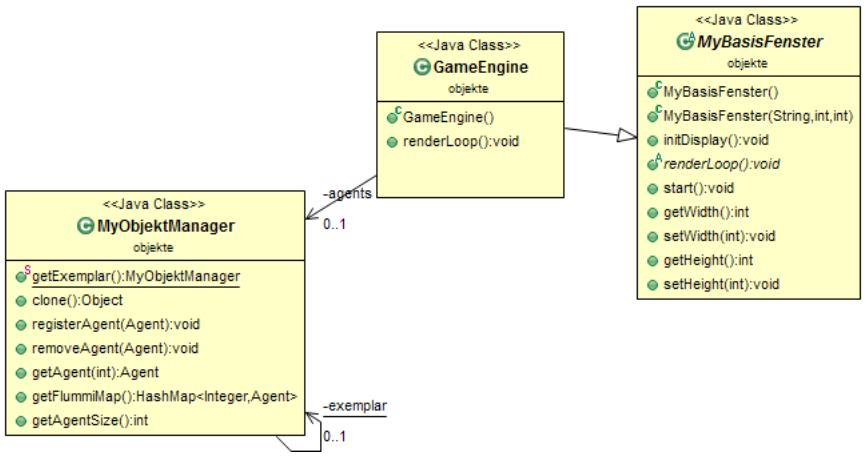
1 Einleitung

Diese Arbeit beschäftigt sich mit der Integration von Schwarmverhalten und Shader-Visualisierung. Die Integration von Schwarmverhalten ermöglicht es, einer Vielzahl von Objekten ähnnliche Verhaltensweisen zuzuweisen, so lassen sich beispielsweise Fischschwärme simulieren.

2 Aufbau

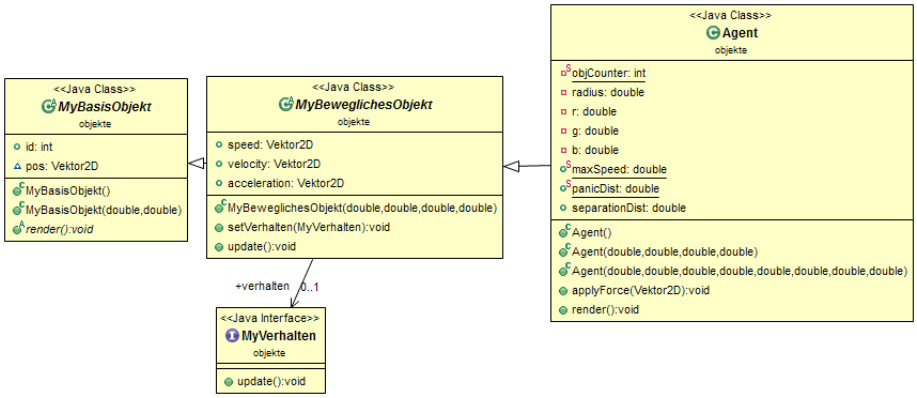
Der Aufbau von GameEngine gestaltet sich folgendermaßen:

- Die Gameengine läd alle notwendigen Objekte, darunter befindet sich die Klasse MyObjektManager, welche neu erzeugte Objekte der Klasse Agents (der zukünftige Schwarm) aufnimmt und verwaltet.
- Die Klasse MyBasisFenster erzeugt den notwendigen Rahmen zur Darstellung und wird von der Klasse GameEngine erweitert.



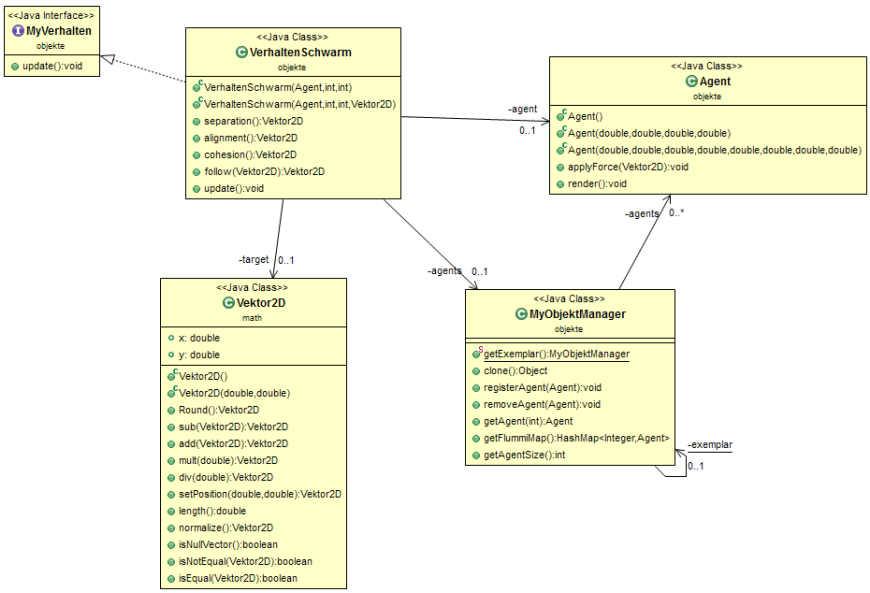
3 Objekte

Die Klasse MyBasisobjekt bildet die Grundlage für alle weiteren Objekte, in der die Position und ID bezogen werden kann.MyBasisobjekt wird von der Klasse MyBeweglichesObjekt erweitert. MyBeweglichesObjekt beinhaltet die Grundfunktionalitäten für Bewegungen, außerdem wird hier das jeweilige Verhalten via Interface übergeben. Diese Klasse wird wiederum von der spezialisierte Agent-Klasse erweitert. Diese weisst dem Objekt Charakteristika, wie z.B. den Radius zu.



4 Schwarmverhalten

Die spezialisierte Klasse für Verhalten, in diesem Fall VerhaltenSchwarm, verfügt über die Klassen MyObjektManager, Vektor2D und Agent. Sie verfügt über verschiedene Methoden, welche als Verhaltensmuster kombiniert werden können, darunter follow, alignment, cohesion und separation. Für Berechnungen ist die Klasse Vektor2D zuständig.



4.1 alignment

Die Methode alignment passt die Bewegungsrichtung des Agents an die des Schwarms an.

```
public Vektor2D alignment() {
    Vektor2D steeringForce = new Vektor2D(0, 0);
    int count = 0;
    for (int i = 0; i < agents.getAgentSize(); i++) {
        if (agent.id == i)
            continue;
    MyBasisObjekt bObj = agents.getAgent(i);
        if (bObj instanceof Agent) {
            Agent bObjF = (Agent) bObj;
            if (LineareAlgebra.euklDistance(agent.pos, bObjF.pos) < (agent.separationDist)) {
                steeringForce.add(bObjF.speed);
                count++;
                }
            }
        }
    if (count > 0) {
        steeringForce.mult(1.0 / count);
        steeringForce.sub(agent.speed);
    }
    return steeringForce;
}
```

4.2 cohesion

Die Methode cohesion beeinflusst die Kohäsion der einzelnen Mitglieder.

```
public Vektor2D cohesion() {
    Vektor2D steeringForce = new Vektor2D(0, 0);
    int count = 0;
    for (int i = 0; i < agents.getAgentSize(); i++) {
        if (agent.id == i)
            continue;

        MyBasisObjekt bObj = agents.getAgent(i);
        if (bObj instanceof Agent) {
            MyBeweglichesObjekt bObjF = (MyBeweglichesObjekt) bObj;
            steeringForce.add(bObjF.pos);
            count++;
        }
    }
    if (count > 0) {
        steeringForce.mult(1. / count);
        steeringForce.sub(agent.pos);
    }
    return steeringForce;
}
```

4.3 seperation

Die Methode seperation kümmert sich um den Abstand der Agents von einander.

```
public Vektor2D separation() {
    Vektor2D steeringForce = new Vektor2D(0, 0);
    for (int i = 0; i < agents.getAgentSize(); i++) {
        if (agent.id == i)
            continue;
        MyBasisObjekt bObj = agents.getAgent(i);
        if (bObj instanceof Agent) {
            Agent bObjF = (Agent) bObj;
        if (LineareAlgebra.euklDistance(agent.pos, bObjF.pos) < (agent.separationDist)) {
            Vektor2D help = new Vektor2D();
            help = LineareAlgebra.sub(agent.pos, bObjF.pos);
            double length = help.length();
            help.normalize();
            help.div(length);
            steeringForce.add(help);
            }
        }
    }
    return steeringForce;
}
```

4.4 follow

Die Methode follow sorgt dafür, dass der Schwarm sich auf einen bestimmten Punkt hinbewegt.

```
public Vektor2D follow(Vektor2D target) {
    Vektor2D help = new Vektor2D(0, 0);
    double dist;
    double speed;
    if (target.x > 0) {
        help = LineareAlgebra.sub(target, agent.pos);
        dist = help.length();
        speed = agent.maxSpeed * (dist / 2);
        speed = Math.min(speed, agent.maxSpeed);
        help.mult(speed / dist);
        help.sub(agent.speed);
    }
    return help;
}
```

5 Renderloop und Fragmentshader

Mit dem Fragmentshader wurden alle Fragmenten auf gelb gesetzt. Renderloop ist eine Schleife, die solange läuft bis das Fenster geschlossen wird. Dabei werden Agenten gerendert und deren Position aktualisiert.

```
public void renderLoop() {
    String fragShader = "" + "void main() { " + "gl_FragColor = vec4(1, 1, 0, 1);" + "}";
    int shaderObjectF = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(shaderObjectF, fragShader);
    glCompileShader(shaderObjectF);
    int programObject = glCreateProgram();
    glAttachShader(programObject, shaderObjectF);
    glLinkProgram(programObject);
    glUseProgram(programObject);

    while (!Display.isCloseRequested()) {
        glClearColor(0.1f, 0.2f, 0.3f, 1);
        glClear(GL_COLOR_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0, this.getWidth(), this.getHeight(), 0, 0, 1);
        glMatrixMode(GL_MODELVIEW);
        glDisable(GL_DEPTH_TEST);

        for (int i = 1; i <= agents.getAgentSize(); i++) {
            Agent aktAgent = agents.getAgent(i);
            aktAgent.setVerhalten(new VerhaltenSchwarm( aktAgent, this.getWidth(), this.getHeight(),
                                                        this.sep, this.ali, this.coh));
            aktAgent.render();
            aktAgent.update();
        }
        Display.update();
    }
}
```