Softwaredokumentation zum Programm „PhysX KugelLineal“

Auftraggeber: Ministerium für Kultus, Jugend und Sport des Landes Baden-Württemberg

Entwickler: Team Golden Girls

Stand: 29.11.2015

Inhaltsverzeichnis

1 Einleitung

Dies ist die Softwaredokumentation zum Programm „PhysX KugelLineal“, welches im Auftrag des Ministeriums für Kultus, Jugend und Sport des Landes Baden-Württemberg von der Firma „Team Golden Girls“ entwickelt wurde. Das Programm soll das Schießen mit einem Radierer-Lineal-Katapult simulieren.

Anforderungen waren mehrere Level und Schwierigkeitsgrade, verschiedene Hindernisse sowie Geräusche bei Kollision mit Hindernissen. Außerdem sollte das Verhalten der geschossenen Kugel physikalischen Gesetzen folgen.

Dieses Dokument soll das Verstehen der Funktionsweise des Programms erleichtern, um die Wartung oder Erweiterung zu vereinfachen. Hierzu werden alle Aspekte im groben in ihrer Funktionsweise beleuchtet und schwierigere Teile anhand des Quellcodes erklärt.

Zusätzlich zu diesem Dokument ist der Quellcode mit Kommentaren versehen, um das Einarbeiten zu erleichtern. Da das Programm in Java geschrieben ist und JavaFX nutzt, sind Kenntnisse in diesen Bereichen empfehlenswert, um das Programm zu verstehen.

2 Überblick

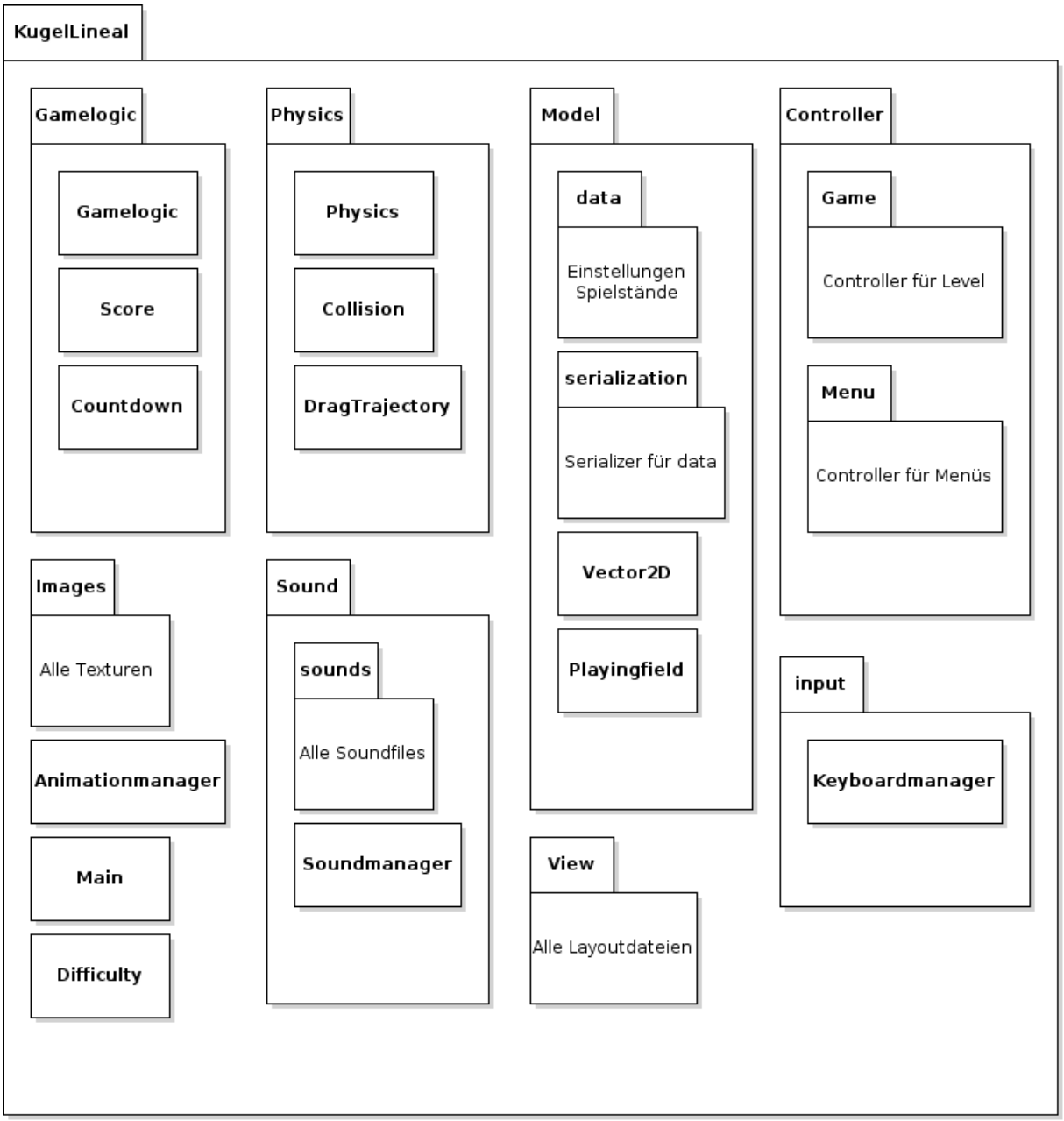

Abbildung 1 Grundlegender Aufbau

Das Programm unterteilt sich in mehrere Pakete:

1. Gamelogic, zuständig für die Rahmenbedingungen eines Levels
2. Physics, zuständig für das Bewegen der Kugel und Kollisionen
3. Model, zuständig für den Zugriff auf Daten sowie das Speichern
4. Input, zuständig für die Erkennung von Benutzereingaben
5. Controller, zuständig für die Buttons etc. auf jedem Screen
6. View, hier werden die Layouts der einzelnen Screens/Szenen definiert
7. Sound, zuständig für das Abspielen von Tönen
8. Images, hier werden die Texturen gespeichert

Für das grundlegende Spielen an sich sind die Pakete Physics und Gamelogic von zentraler Bedeutung. In diesen Paketen werden die meisten Berechnungen während des tatsächlichen Spielens vorgenommen.

3 Start des Programms und die Klasse Main

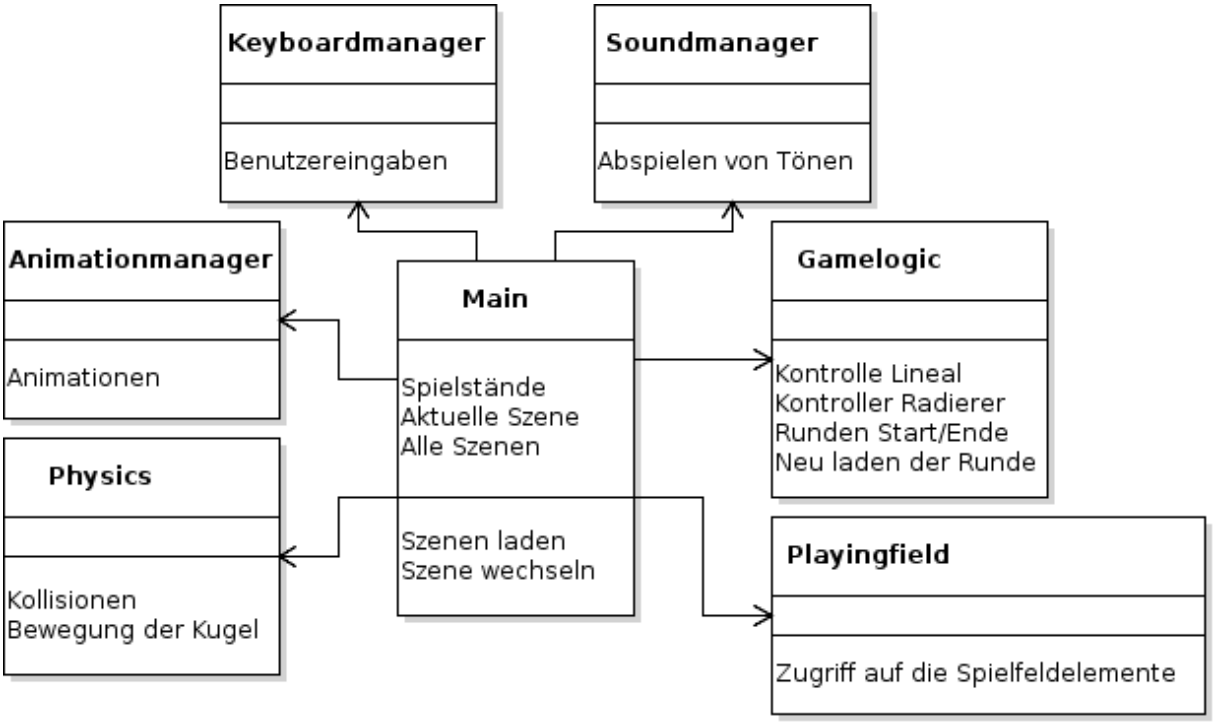

Abbildung 2 Die Klasse Main und ihre grundlegenden Funktionen

Beim Start des Programmes wird die Methode start(Stage primaryStage) der Klasse Main aufgerufen. Die Klasse Main dient als zentrale Anlaufstelle des Programms. Sie bietet Zugriff auf die wichtigsten Klassen (siehe Abb. 2). Außerdem dient die Klasse Main als „Stage“. Auf einer Stage können Scenes angezeigt werden. Die Stage ist im Grunde der Fernseher und die Scenes sind die verschiedenen Sender. Die Klasse Main lädt alle verfügbaren Scenes und startet mit der Scene „MainMenu“, welche das Hauptmenü darstellt. Außerdem werden alle verfügbaren Spielstände geladen, damit diese in „LoadScreen“, also in der „Spiel-Laden“-Scene, zur Verfügung stehen. Alle Scenes sind im Ordner „View“ zu finden.

Von zentraler Bedeutung in der Klasse Main ist die Methode fillScenesMap(List<String> files), welche alle Scenes aus dem Ordner View lädt. Falls eine „Level“-Scene geladen wird, wird zusätzlich noch eine weitere „BaseGame“-Scene hinzugefügt. Dadurch gibt es zu jedem Level „x“ auch ein „BaseGamex“. In der „BaseGame“-Scene sind UI-Elemente, die bei jedem Level gleich sind, zum Beispiel die Position des Lineal-Radierer-Katapults oder die Punkteanzeige. Zu jedem Level gibt es im Ordner /Controller/Game/ außerdem noch einen Controller, welcher die in „BaseGame“ spezifizierten Objekte lädt.

4 Das Paket Physics

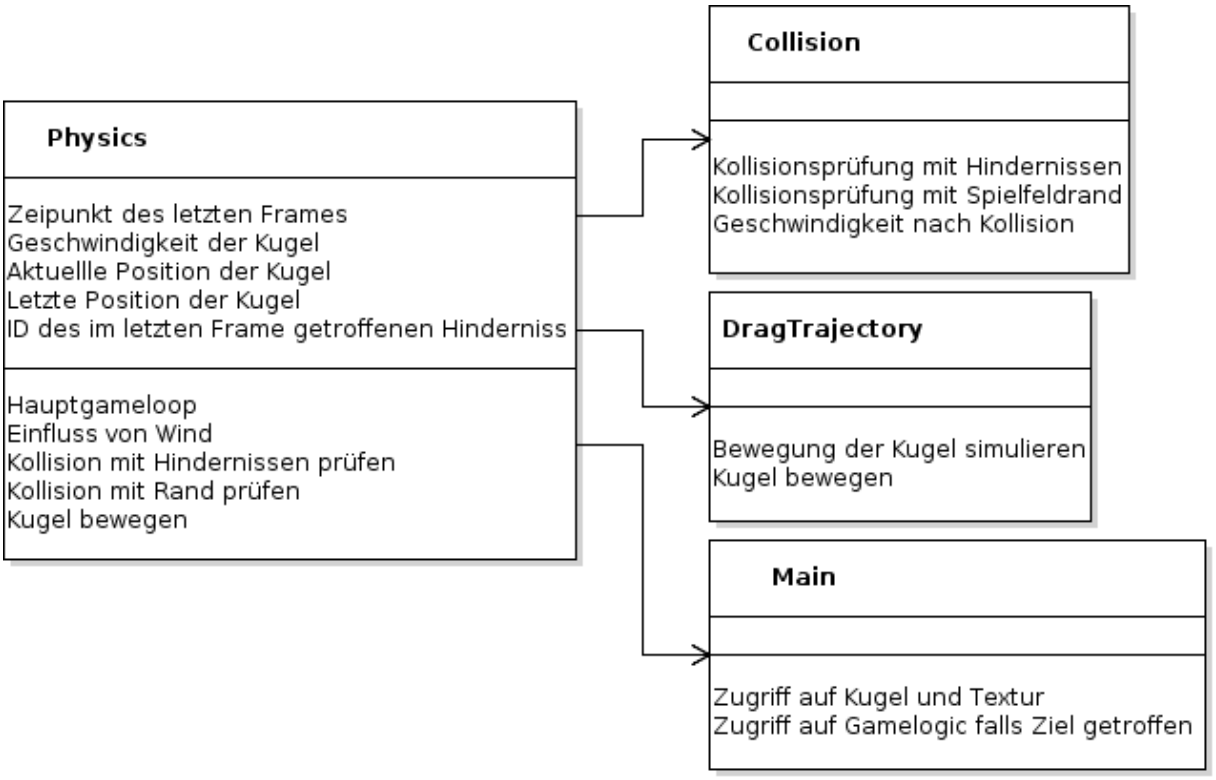

Abbildung 3 Die in "Physics" enthaltenen Klassen und ihre grundlegenden Funktionen

Die in „Physics“ enthaltenen Klassen befassen sich mit dem realitätsnahen Bewegen der Kugel. Die Klasse Physics erbt von der JavaFX-Klasse AnimationTimer und enthält den „Hauptgameloop“ in der Methode handle( ), welche jedes Frame aufgerufen wird und folgendes erledigt:

1. Berechnet die Zeit seit dem letzten Frame
2. Simuliert das Bewegen der Kugel
3. Simuliert den Einfluss von Wind auf die Kugel
4. Prüft auf Kollisionen mit Hindernissen, dem Ziel oder dem Lineal
5. Prüft auf Kollision mit dem Spielfeldrand

Die Klasse DragTrajectory ist für die Berechnung der Flugbahn zuständig und das Bewegen der Kugel entlang dieser. Die Berechnung der neuen Position erfolgt inkrementell bei jedem Frame.

Die Klasse Collision wird für die Kollisionserkennung genutzt. Hierzu wird die Methode isColliding() genutzt, welche prüft, ob 2 JavaFX Shapes sich überschneiden. Außerdem wird über die Methode getPostCollisionVelocity() die Geschwindigkeit der Kugel nach einer Kollision berechnet. Hierzu wird die Geschwindigkeit der Kugel, der Dämpfungsfaktor des Hindernisses und der Normalen-Vektor an das Hindernis benötigt.

```
class Collision {
    //checks if 2 JavaFX Shapes are colliding
    public boolean isColliding(Shape obj1, Shape obj2) {
        return Shape.intersect(obj1, obj2).getBoundsInLocal().getWidth() != -1;
    }
    //checks if the point y is above or below the window
    public boolean isOutsideVerticalBounds(double y, double height) {
        return ((y < 0) || (y > height));
    }
    //checks if the point x is left or right of the window
    public boolean isOutsideHorizontalBounds(double x, double width) {
        return ((x < 0) || (x > width));
    }
    //calculates the velocity after a collision
    public void getPostCollisionVelocity(Vector2D velocity, double dampening, Vector2D normalUnitVector) {
        final double factor = (1 + dampening) * velocity.scalarProduct(normalUnitVector);
        final Vector2D vec = normalUnitVector.scalarMultiplication(factor);
        velocity.subtract2(vec);
    }
}
```

Abbildung 4 Die Klasse Collision

4.1 Die Methode checkShapeCollisions()

Die Methode checkShapeCollisions() prüft bei jedem Element, ob es zu einer Kollision mit der Kugel kommt. Hierbei gibt es mehrere Fälle zu betrachten.

4.1.1 Das Hindernis ist vom Typ „Goal“

Falls es zu einer Kollision mit einem Hindernis vom Typ „Goal“ kommt, so ist der Level erfolgreich abgeschlossen.

4.1.2 Das Hindernis ist vom Typ „Line“

```
//supported are collisions with lines only
if (child instanceof javafx.scene.shape.Line) {
    javafx.scene.shape.Line line = (javafx.scene.shape.Line) child;
    //use colliding isColliding of Collision-Class that uses the intersects method of javafx-shapes
    if (m_Collision.isColliding(m_Main.getPlayingfield().getBall(), line)) {
        Vector2D normal = new Vector2D();
        //get the angle of the line
        double yx = line.getLocalToSceneTransform().getMyx();
        double yy = line.getLocalToSceneTransform().getMyy();
        double angle = Math.atan2(yx, yy);
        //rotate our normal vector so that it resembles the normal of the line
        normal.rotate(angle);
        //in case we hit the same object as last time (this case occurs if the ball gets stuck)
        if (line.getId() != null) {
            if (m_hitlastframe.equals(line.getId())) {
                //we move the ball away from the line
                Vector2D n2 = normal.scalarMultiplication(-10);
                m_Position.add2(n2);
                m_Collision.getPostCollisionVelocity(m_Velocity, 1, normal);
            }
        }
        // new line is hit
        else {
            // post-collision velocity is calculated
            m_Collision.getPostCollisionVelocity(m_Velocity, m_Dampening, normal);
```

Abbildung 5 Kollision mit einer Linie

„Line“ ist eine JavaFX Klasse, die um die Texturen gelegt wird. Falls es zu einer Kollision mit einer „Line“ kommt, so kann auf 3 verschiedene Arten reagiert werden

1. Falls die Line im letzten Frame bereits getroffen wurde, wird die Kugel ein Stück vom Kontaktpunkt entfernt, bevor die neue Geschwindigkeit berechnet wird. Dadurch wird verhindert, dass die Kugel stecken bleibt.

2. Falls eine Line getroffen wird, die nicht im letzten Frame bereits getroffen wurde, so wird direkt die neue Geschwindigkeit berechnet.
3. Falls die getroffene Line das Lineal ist, so wird geprüft, ob der Ball gerade geschossen werden soll, oder einfach nur im Verlauf des Levels wieder an das Lineal prallt. Falls die Kugel gerade geschossen werden soll, so wird ihr noch zusätzlich die Kraft, die das Lineal auf sie ausübt, angerechnet. Falls die Kugel nicht gerade geschossen werden soll wird die Kollision wie eine normale Kollision mit einer Line gehandhabt.

```
if (line.getId() != null) {
    if (line.getId().equals(m_Main.getPlayingfield().getLineal().getId()) && m_Main.getAnimationmanager().islineallaunched()) {
        //call the game logic
        Main.getGamelogic().onLinealHit();
        //the next part is for calculating the velocity that is applied to the ball according to the distance, the ball is away from the center of "Line
        //first we get the start of the line
        Vector2D linestart = new Vector2D(line.getStartX() + line.getLayoutX(), line.getStartY() + line.getLayoutY());
        //its rotation
        Point3D rotaxis = line.getRotationAxis();
        //we calculate the center of the line by subtracting the radius
        Vector2D linecenter = new Vector2D(rotaxis.getX() + line.getLayoutX(), rotaxis.getY() + line.getLayoutY());
        double lineradius = linestart.subtract(linecenter).length();
        //we calculate the distance between the line center and ball
        Vector2D centertocircle = m_Position.subtract(linecenter);

        //the next part might seem redundant
        //we calculate the angle to the ball two times
        //this is necessary since the angle calculated only goes from 0-180 but we have to know on which side of the line we are

        // the angle between the normal and the center of the ball is calculated
        double deltaangle = Math.toDegrees(normal.getAngleTo(centertocircle));
        double distance = centertocircle.length();
        //this flips the normal of the line by 180°
        Vector2D linevector = new Vector2D();
        linevector.rotate((Math.PI / 2.0) + angle);
        //again the angle to the ballcenter is calculated
        double deltaangle2 = Math.toDegrees(linevector.getAngleTo(centertocircle));

        boolean kickball = false;
        //the next part is for checking if the ball is actually kicked or just bounces
        //the angle-checking checks if the ball to be in front of "Lineal" and also in its direction of rotation
        if (deltaangle > 90 && deltaangle2 < 90) {
            kickball = true;
        }
        if (deltaangle < 90 && deltaangle2 > 90) {
            kickball = true;
        }
        // if the ball is about to be kicked, we take the relation of lineradius and distance into the calculation of the new velocity of the ball
        if (kickball) {
            normal.scalarMultiplication2(-1 * Main.getGamelogic().getLinealPower() * (distance / lineradius));
            m_Velocity.add2(normal);
        }
    }
}
```

Abbildung 6 Kollision mit dem Lineal

4.1.3 Nach der Kollisionsprüfung

Falls es zu einer Kollision kam, so wird der Ball auf seine Position aus dem vorherigen Frame zurückgesetzt und ein zufälliger Ton wird abgespielt. Bei einer Kollision mit einer Line wird zusätzlich die ID der getroffenen Line bis zum nächsten Frame gespeichert, außerdem wird collisionHappened auf „true“ gesetzt.

5 Das Paket Gamelogic

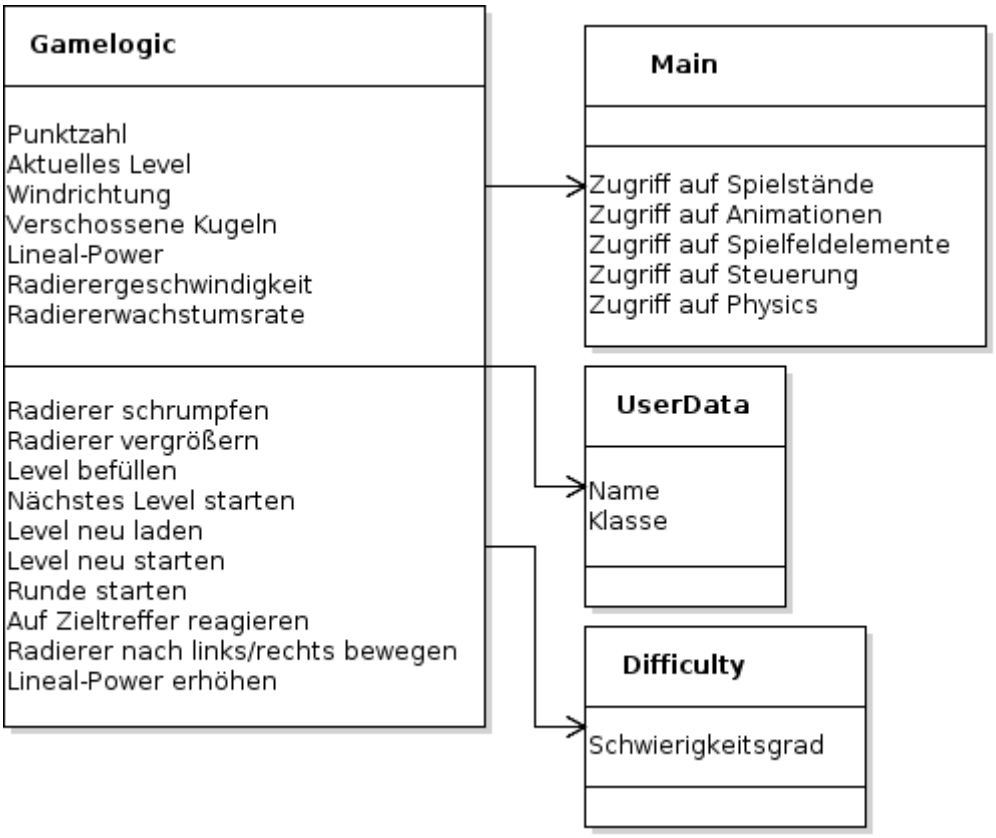

Abbildung 7 Die Klasse Gamelogic

Die Klassen im Paket Gamelogic beschäftigen sich mit den Rahmenbedingungen des Spiels. Hier werden Aufgaben wie der Start der Runde, der Countdown, das Laden der Hindernisse im Level, das „Spannen“ des Lineals etc. erfüllt. Neben der Klasse Physics ist diese Klasse notwendig, um das Spiel wirklich spielen zu können. Eine der wichtigsten Methoden dieser Klasse ist die Methode setLevel(). Diese Methode setzt die gewählte Scene auf die PrimaryStage und bringt die Kugel an die richtige Position. Außerdem wird das Level angezeigt sowie der Windmesser eingestellt. Des Weiteren werden die Nutzereingaben auf die aktuelle Scene gewechselt und die Animationen gestartet (z.B. das drehen der Kugel).

```
//Load all game elements and prepare everything to play
public void setLevel(String scenename, Difficulty d) {
    if (Main.setScene(scenename)) {
        m_difficulty = d;
        m_currentSceneName = scenename;
        m_level = Integer.parseInt(m_currentSceneName.substring(m_currentSceneName.lastIndexOf('l') + 1));
        m_Main.getPlayingfield().restore();
        m_Main.getPlayingfield().getBall().setLayoutX(0);
        m_Main.getPlayingfield().getBall().setLayoutY(0);
        m_Main.getPlayingfield().getBall_Image().setLayoutX(m_Main.getPlayingfield().getBall().getLayoutX());
        m_Main.getPlayingfield().getBall_Image().setLayoutY(m_Main.getPlayingfield().getBall().getLayoutY());
        Main.getKeyboardmanager().applyControlsToCurrentScene();
        m_Main.getAnimationmanager().start();
        Main.getPhysics().setWindfactor(((float) Difficulty.toInteger(d)) / ((float) Difficulty.toInteger(Difficulty.EXTREME)));
        windDirectionInDegrees = (int) (Math.random() * 360);
        m_Main.getPlayingfield().getWindmesser().setRotate(windDirectionInDegrees);
        retry();
    }
}
```

Abbildung 8 Die Methode setLevel

6 Möglichkeiten zur Erweiterung

In diesem Kapitel werden mögliche Erweiterungen besprochen und wie diese am einfachsten umgesetzt werden können.

6.1 Level

Um ein neues Level hinzuzufügen ist es am einfachsten, eine Kopie eines bereits vorhandenen Levels zu machen und diese zu überarbeiten. Wichtig ist, dass wenn die Grafiken verschoben werden, auch die Lines zu verschieben. Um das Ziel zu definieren muss ein Rectangle mit der fx:id=goal unter die entsprechende Textur gelegt werden. Zusätzlich zum Erstellen einer weiteren .fxml Datei muss auch ein neuer Controller erstellt werden. Hierzu ist es am einfachsten, eine Kopie eines bereits vorhandenen Level-Controllers zu erstellen und in diesem die Zahl hinter „Basegame“ entsprechend zu ändern. Wenn eine Level-FXML-Datei angelegt wird ist zu beachten, dass diese im Namen „Level“ und dann eine noch nicht benutzte Zahl enthält, ansonsten wird die Datei als normales Menü angesehen.

6.2 Töne

Um neue Töne einzufügen müssen diese als .mp3-Datei in den Ordner /sound/sounds eingefügt werden. Die Datei muss „sound“ + von 0 fortlaufende Zahl + „.mp3“ heißen. Falls also die Sounds von 1-10 schon existieren, so muss der neue Sound sound11.mp3 heißen. Zusätzlich zum Hinzufügen des neuen Sounds muss in der Klasse Soundmanager noch das Attribut number_of_sounds auf die neue Zahl an Sounds geändert werden. Soll der neue Sound auch bei der Kollision mit einem Hinderniss abgespielt werden, so muss der Aufruf von playRandSound in der Methode checkShapeCollisions() angepasst werden. Zur Wiedergabe der Sounds werden JavaFX AudioClips genutzt.

6.3 Grafiken

Falls neue Grafiken genutzt werden sollen, so müssen diese in den Ordner „images“ geschoben werden. Anschließend können die Texturen in den .fxml-Dateien genutzt werden.