Schriftliche Dokumentation zum Softwareentwurf des Projekts „Kugel-Lineal“

Datum: 20.10.15

Version 2.0

Inhalt

1 Ziel des Projektes

Ziel des Projektes ist es, ein Programm zu schreiben, um das Schießen einer Papierkugel mit einem Lineal zu simulieren. Das Programm soll an Schulen zum Einsatz kommen, um die Verschmutzung in den Klassenzimmern zu reduzieren und Aggressionen bei Schülern abzubauen. Genauere Anforderungen können dem Pflichtenheft entnommen werden.

2 Grundlegende Komponenten

01_Grundkomponenten.class.violet.html

Eine Software nicht in Komponenten zu unterteilen kann zu verschlechterter Wartbarkeit, Übersichtlichkeit sowie Wiederverwendbarkeit führen.

Deshalb orientiert sich unsere Grundarchitektur an dem MVC-Prinzip, d. h.:

1. Es gibt eine Komponente für alles, was dem Benutzer angezeigt wird (View)

2. Es gibt eine Komponente für die Logik des Programmes (Controller)

3. Es gibt eine Komponente zum Halten, Speichern und Laden der Daten (Model)

In unserem Fall unterteilt sich der Controller nochmals in 2 Subkomponenten, die **3** Engine und das **5** Spielfeld bzw. die **6** Spielfeldelemente.

Die **3** Engine kümmert sich um grundlegende Aufgaben wie z. B. **3.5** Eventmanager oder **3.4** Processmanager.

Die **6** Spielfeldelemente sind die logische Repräsentation der grafischen Objekte wie z. B. der **6.4** Ball oder des **6.8** Catapult.

3 Engine

Die Klasse **Engine** bietet Funktionen zum Starten, Laufen und Beenden der Engine. Sie ist, genau wie alle weiteren Komponenten, als Singleton implementiert. Sie enthält deren Instanzen und bietet Methoden zum Erhalten von Referenzen auf diese.

3.1 Graphics

Die Klasse **Graphics** verwaltet alle **Objekte** die von **Graphicsobject** erben. Sie zeichnet alle Objekte. Es können Objekte hinzugefügt oder gelöscht werden.

3.2 Physics

Die Klasse **Physics** verwaltet alle Objekte, die von **Physicsobject** erben. Hier wird das nächste Frame simuliert und auf Kollisionen geprüft. **Physicsobjects** können hinzugefügt oder gelöscht werden.

3.3 Timer

Die Klasse **Timer** dient zum frequentiellen Ablauf des Programms.

3.4 Processmanager

Die Klasse **Processmanager** verwaltet alle Prozesse. Durch den Prozessmanager können Prozesse hinzugefügt oder entfernt werden. Ein **Processowner** besitzt eine Callback-Methode und der **Process** eine **Tickrate** und einen **Processowner**. Alle **Processe** werden über **runProcesses** durchlaufen.

3.5 Eventmanager

Die Klasse **Eventmanager** verwaltet alle Objekte, die von **Eventlistener** erben. Solche können hinzugefügt oder entfernt werden. **Events** werden über **queueEvent** eingereiht und abgearbeitet durch Aufrufen der Callback-Methoden der **Eventlistener** über einen Vektor.

3.6 Controller

Die Klasse **Controller** verwaltet **Controls**. Diese enthalten einen Vektor, der Tasten IDs auf Event IDs mappt. Der **Controller** prüft in **run** auf Tastenveränderungen und sendet bei solchen ein Event mit entsprechender ID.

4 Menüs


02_Menunavigation.html


03_Menu.class.violet.html

4.1 Erstellung der Menüoberfläche:

Es können eine ganze Reihe an Packages importiert werden, die alle Standartmäßig in Java 8 vorhanden sind. Für die Erstellung des Menüs werden folgende benötigt:

1. javafx.application.*
2. javafx.stage.*
3. javafx.scene.*
4. javafx.scene.layout.*
5. javafx.scene.control.*

4.2 Main

Die Klasse Main erbt von der Klasse „Application“ in dieser sind einige Methoden bereits implementiert. Als erstes wird die Methode **launch(args)** aus „Application“ aufgerufen, diese bildet die Grundlage für jedes JavaFX Programm. Die Methode **void start( )** wird aufgerufen und alle in dieser anzuzeigenden Buttons und Szenen (unterschiedliche Menüfester, z.B. Hauptmenü, Settings…) werden darin implementiert.

Das Interface **EventHandler** gibt die Methode **handler( )** vor, mit der alle Buttons mit Funktionen versehen werden.

5 Spielfeld


06_Spielfeld.class.violet.html

5.1 Playingfield

Die Klasse **Playingfield** fungiert als eine Art Container, in dem alle in einem Level vorkommenden Objekte gesammelt werden. Außerdem kann über das **Playingfield** auch auf die Objekte zugegriffen werden. Die **ArrayList obstacles** enthält alle Hindernisse, die auf dem **Playingfield** existieren. Da alle Hindernisse von der Klasse **Entity** erben, können sowohl runde, als auch eckige Hindernisse in der Liste gespeichert werden. Außerdem kann über das **Playingfield** noch auf folgende Elemente zugegriffen werden:

1. Die Kugel (**ball**)
2. Das Ziel (**target**)

3. Der Balken, der die Schusskraft repräsentiert (**bar**)
4. Die Punkteanzeige (**score**)
5. Den Windmesser (**wind)**
6. Das Katapult (**catapult**)
7. Den Luftwiderstand (**airDensity**)

5.2 PowerBar

Die Klasse **PowerBar** repräsentiert die Schusskraftanzeige am Rande des Bildschirms. Über die Methode **launch( )** wird die Anzeige gestoppt und die aktuelle Kraft wird zurückgegeben. Die anderen Methoden dienem dem Zugriff auf die beiden Attribute **power** (Kraft mit der auf das Lineal gedrückt wird) und **speed** (Geschwindigkeit, mit der die Anzeige fluktuiert).

5.3 ScoreBoard

Die Klasse **ScoreBoard** repräsentiert die Punkteanzeige über der Schusskraftanzeige am Bildschirmrand. Über die Methoden **add( )** sowie **reduce( )** kann der Punktestand (**score**) bearbeitet werden, ohne direkt auf ihn zugreifen zu müssen.

5.4 WindMeter

Die Klasse **WindMeter** repräsentiert den Windmesser in der rechten oberen Bildschirmecke. Über den Windmesser kann auf die Windrichtung (**direction**) sowie die Windstärke (**strength**) zugegriffen werden.

6 Spielfeldelemente

07_Spielfeldelemente.class.violet.html

08_Catapult.class.violet.html

6.1 Vector

Die Klasse **Vector** dient dazu, im Zweidimensionalen möglichst einfach arbeiten zu können. Die Klasse **Vector** enthält zwei Attribute **x** und **y**. Durch diese Werte lassen sich sowohl Punkte, als auch Richtungen definieren. Durch die Methoden **addVector** sowie **multiplyVector** wird das Rechnen mit Vektoren vereinfacht.

6.2 Entity

Die Klasse **Entity** enthält grundlegende Elemente, die auch alle anderen Spielfeldelemente benötigen. In der **ArrayListe positions** wird gespeichert, wo das Element ist. Das Attribut **dampening** gibt an, wie hoch der Dämpfungsfaktor ist, falls es zu Kontakt mit einem anderen Element kommt.

6.3 Circle

Die Klasse **Circle** entspricht einem kreisförmigen Hindernis. Zusätzlich zu den in **Entity** bereits definierten Attributen gibt es hier noch das Attribut **radius**, welches den Radius des Kreises festlegt. Über die Methode **getBorders( )** werden näherungsweise alle Punkte zurückgeliefert, die auf dem Kreis liegen.

6.4 Ball

Die Klasse **Ball** entspricht der vom Katapult gefeuerten Kugel. Zusätzlich zu den in **Circle** definierten Attributen gibt es hier noch die Richtung der Kugel (**direction**), die Geschwindigkeit der Kugel (**speed**) und das Gewicht der Kugel (**weight**). Über die Methoden **speedUp( )**, sowie **speedDown( )**, kann die Geschwindigkeit der Kugel beeinflusst werden, ohne direkt auf den Wert zugreifen zu müssen.

6.5 TargetCircle

Die Klasse **TargetCircle** repräsentiert ein kreisförmiges Ziel. Sie enthält die gleichen Attribute wie die Klasse **Circle**.

6.6 Rectangle

Die Klasse **Rectangle** entspricht einem rechteckigen Hindernis. Über die Methode **getBorders( )** werden näherungsweise alle Punkte zurückgeliefert, die auf dem Hindernis liegen.

6.7 TargetRectangle

Die Klasse **TargetRectangle** repräsentiert ein rechteckiges Ziel. Sie enthält die gleichen Attribute wie die Klasse **Rectangle**.

6.8 Catapult

Das Attribut **rubber** ist ein Dreieck, **contactPoint** ist der Punkt, an dem sich Lineal und Dreieck berühren (der Drehpunkt).

- **fire( )** animiert das Lineal zu einer Drehbewegung
- **move*( )** ändert die Position des Lineals nach links oder rechts

7 DragTrajectory

09_DragTrajectory.class.violet.html

calculatePositions() berechnet die einzelnen Positionen der Flugbahn, die dann in **mPositions** gespeichert werden.

Alternativ kann man die Positionen in einem Path zusammenfassen, entlang dessen dann die Kugel fliegt, je nachdem, wie die Animation am Ende ausgeführt wird (über PathTransition?) In dem Fall kann die Klasse auch statisch werden.

Die Berechnung der Flugbahn mit Luftwiderstand beachtet folgende Faktoren:

- Fläche des Durchschnitts der Kugel
- Masse der Kugel
- Luftdichte (fest)
- Luftwiderstandskonstante ist abhängig von Form des Körpers