Praktische Informatik I | WS 2012/13 | Vitalij Kochno
TutorIn: Jan Winkler | Gruppe: 9 | Yorick Netzer
Christophe Stilmant

# Übungsblatt 8

Aufgabenlösung
Abgabe: 16.12.2012

---

## Aufgabe 1 Alles super hier

Im unserem Projekt benutzen wir 4 Superklassen: ScrollableWorld, Scrollable, Item und Obstacle. Zu beginn erst mal die Idee zu unserem Spiel. Wir haben eine sichtbare Welt diese wird durch eine scrollbare Welt erweitert. Dies erreichen wir mit der ScrollableWorld-Klasse. Dort werden die x- und y-Koordinaten aller Klassen umgerechnet in die x- und y-Koordinaten der ScrollablenWorld. Desweiteren wird die Rocket als Mittelpunkt gesetzt bis sie an das echte Ende der Welt kommt. Scrollable verwaltet das neue move, da das move vom Actor bei uns durch die neue berchnung der x- und y-Koordinaten nicht mehr funktioniert. Item ist die Superklasse der sammelbare Items. Bis jetzt ist der nur unser Shield drinne, aber da noch weitere Items folgen werden haben wir uns schon mal vorbereitet. Zum Schluss Objects, dass ist die Superklasse die die Hindernisse verwaltet, hier wird bei einer Kollision von SpaceLemon oder BlakHole Pose ausgeführt, was zu ein weiterfleigen verhindert.

**Tests:** Nachdem wir unsere ScrollableWorld erstellt hatten und auch in Scrollable und den Klassen alle eigenschaften verteilt haben, merkten wir das die Methode move aus Actor nicht mehr richtig funktioniert, deswegen haben wir eine neue geschrieben. Anschliesend hatten wir Probleme mit dem Portal. In der neuen Welt wurde die Rocket nicht mehr als Mittelpunkt betrachtet, anfangs hatten wir ja eine Array und das erste Objekt war die Rocket und dieses wurde dann zentrisiert. Durch wechseln der Welt wird das Array neuerstellt und die Rocket wird ja nachträglich übergeben, was dazu führt das sie nicht mehr an der ersten stelle steht. Dies haben wir dadurch behoben indem wir die Rocket sofort als Mittelpunkt gesetzt haben und nicht mehr durch den ersten Array Platz.

## Aufgabe 2 Massenkarambolage

Da wir eine Superklassen besitzen die über fast alle Objekte steht und es eigentlich auch diese sind die für eine Kollision wichtig sind, haben wir das Kollidieren dort programmiert.

```
350     /* Hier die Methoden für Aufgabe 2*/
351
352     /**
353      * Diese Methode überprüft, ob der Objekt ObjClass
354      * eine Kollision mit irgend ein anderen Objekt von
355      * die Welt hat.
356      *//*
357     public Actor  getCollidingObject(Class ObjClass)
358     {
359         return getOneIntersectingObject(ObjClass);
360     }
361     */
362
363     /**
364      * Testet auf Kollision mit einer best KLasse
```

```
365      *//*
366     public boolean isColliding(Class objClass)
367     {
368     return getOneIntersectingObject(objClass) != null ;
369     }*/
370
371     /**
372      * Testet auf neue Kollision( mit einer best. Klasse)
373      *//*
374     public boolean neueKollsionmit(Class objClass)
375     Actor a = getCollidingObject(objClass);
376     if(a!=null)
377     {
378         Actor a2 = collisions.get(objClass);
379         collisions.put(objClass,a);
380         if(a.equals(a2))
381         {
382             return false;
383         }
384         else
385         {
386             return true;
387         }
388     }
389     else
390     {
391         return false;
392     }
393     */
394 }
```

**Tests:** Für die ersten beiden Methoden haben wir uns mit System.out.println(methode1/2) das Ergebnis anzeigen lassen. Für die ersten Methode haben wir ganze null bekommen bis wir eine Klasse berührt haben, dann haben wir die Klasse ganze Zeit geliefert bekommen solange die Kollision besteht.
Bei der zweiten Methode wurde und ganze Zeit false geliefert bis eine Kollision entstand, dann wurde true returnt.
Die dritte Methode haben wir ebenfalls in System.out.println() getestet, dies war aber etwas komplizierter, da wir ganze Zeit false geliefert bekommen, dann ganz kurz ein true und dann wieder nur false bis wir ein neues Objekt berühren. Da aber die Abfrage recht schnell geht, hat man das true oft übersehen. Deswegen haben wir den Debugger verwendet. Wir hatten einen Stop auf die Zeile 358 gelegt, dort wo das true return wird und haben und dann weiter getestet ob er an der gleiche Stelle noch mehr trues ausgeben wird, hat er nicht weil es ja keine neue Kollision ist.

## Aufgabe 3 Richtig kollidieren

```
234      * @return Collidierendes Object bzw null
235      */
236     public Actor getCollidingObject(Class objClass)
237     {
238         List<Actor> actors = getIntersectingObjects(objClass);
239
240         for(Actor actor : actors)
241         {
242             Scrollable scrble = (Scrollable) actor ;
243             if(isRealCollision(scrble))
244             {
245                 return actor;
246             }
247         }
```

```
248         return null;
249     }
```

Hier erstellen wir eine Array mit den Klassen und überprüfen anschliesend ob eine Kollision da ist, dann return wir das Objekt oder null. Hier haben wir schon die nächste Methoden mit einbezogen und aus der normalen Kollision eine echte Kollision erstellt.

```
307     /**
308      * Diese Methode überprüft, ob die Kollision visuel ist; dh dass die Kollision
           zwichen unsichtbare Pixel nicht als
309      * Kollision erkannt wird. Die Kollision wird als "real" erkannt wenn sie Visuel ist.
310      *
311      * @return true wenn die Kollision Visuel ist
312      * @return false wenn die Kollision nicht Visuel ist.
313      */
314     public boolean isRealCollision(Scrollable scrble)
315     {
316         boolean colliding = false;
317         for(int i = 0; i < getImage().getWidth(); i++)
318         {
319             for(int j = 0; j < getImage().getHeight(); j++)
320             {
321                 if(getImage().getColorAt(i, j).getAlpha() != 0)
322                 {
323                     double x1_welt = getRealX() + (i - getImage().getWidth() / 2)*Math.
                          cos(Math.toRadians(getRotation())) - (j - getImage().getHeight()/2)
                          *Math.sin(Math.toRadians(getRotation()));
324
325                     double y1_welt = getRealY() + (i - getImage().getWidth() / 2)*Math.
                          sin(Math.toRadians(getRotation())) + (j - getImage().getHeight()/2)
                          *Math.cos(Math.toRadians(getRotation()));
326
327                     int x2_px = (int) (scrble.getImage().getWidth() /2  + (x1_welt -
                          scrble.getRealX())* Math.cos(Math.toRadians(scrble.getRotation()))
                          + (y1_welt - scrble.getRealY())*Math.sin(Math.toRadians(scrble.
                          getRotation())));
328                     int y2_px = (int) (scrble.getImage().getHeight() /2 - (x1_welt -
                          scrble.getRealX())* Math.sin(Math.toRadians(scrble.getRotation()))
                          + (y1_welt - scrble.getRealY())*Math.cos(Math.toRadians(scrble.
                          getRotation())));
329                     if(0 <= x2_px && 0 <= y2_px && x2_px < scrble.getImage().getWidth()
                          && y2_px < scrble.getImage().getHeight()){
330
331                       if(scrble.getImage().getColorAt(x2_px, y2_px).getAlpha() != 0)
332                       {
333                           if(getScrWorld().isTestmodus())
334                           {
335                               getImage().setColorAt(i, j, Color.RED);
336                               colliding = true;
337                           }
338                           else
339                           {
340                               return true; // für Geschwindigkeit, im Normalfall kann
                                      man hier abbrechen
341                           }
342                       }
343                     }
344                 }
345             }
346         }
347         return colliding;
348     }
```

Hier berechnen wir die Pixel um und testen die auf Transparent, anschliesend wird hier noch auf eine Kollisien der sichtbaren Pixel überprüft.

Um in den Testmodus zu gelangen muss man rechtsklick-¿geerbt von ScrollableWorld -¿ settestmodus oder man fügt getScrWorld().getTestmodus an die passende stelle ein.

**Tests:** Wir haben auch hier mit System.out.printIn() gearbeitet, aber das hat uns nicht viel weitergeholfen. Deswegen haben wir unsere transparenten Pixel erstmal eine Farbe gegeben, also überprüft ob sie transparent sind und dann Color.BLUE benutzt. So haben wir die echten Umrisse des Images gesehen. Desweiteren hat uns der Testmodus, nachdem wir den durch ausprobieren hinbekommen haben, geszeigt das bei einer Kollision wirklich nur die "echten" Pixel angemalt werden, also dort bei einer Kollision true return wird.