Übungsblatt 8

Abgabe: 16.12.2012

Aufgabe 1 Alles super hier (20%)

Führt Superklassen in eurem Spiel ein, mindestens für alle Klassen, deren Instanzen entweder Hindernisse darstellen oder aufgehoben werden können. Ändert euren restlichen Code so ab, dass diese Klassen verwendet werden, wo es sinnvoll ist. Beachtet, dass in Aufgabe 2 eine gemeinsame Superklasse für alle Klassen eingeführt wird, die auf Kollisionen testen.

Aufgabe 2 Massenkarambolage (30%)

Führt eine Superklasse für alle Klassen ein, die auf Kollisionen testen. Diese soll drei öffentliche Methoden haben:

1. Eine Methode, die das Objekt einer bestimmten Klasse zurückliefert, mit dem gerade eine Kollision besteht, bzw. `null`, wenn keine Kollision besteht. Vorerst ist diese Methode einfach nur ein Synonym für `getOneIntersectingObject`.
2. Eine Komfortmethode, die zurückliefert, *ob* gerade eine Kollision mit einem Objekt einer bestimmten Klasse besteht. Sie soll sich auf die erste Methode abstützen.
3. Eine Methode, die zurückliefert, ob gerade eine neue Kollision mit einer Instanz einer bestimmten Klassen begonnen hat. Da diese Methode für verschiedene Klassen aufgerufen werden kann, muss sie darüber Buch führen, ob und wenn ja, mit welcher Instanz einer Klasse gerade eine Kollision besteht. So kann sie auch zurückliefern, wenn plötzlich mit einer anderen Instanz derselben Klasse eine Kollision begonnen hat. Auch diese Methode soll sich auf die erste Methode abstützen.

Baut alle Kollisionstests in eurem Code so um, dass sie nur noch diese drei Methoden benutzen.

Aufgabe 3 Richtig kollidieren (50%)

Bisher hat es unsere HeldIn unnötig schwierig, weil `getOneIntersectingObject` auf dem Schneiden der die Objekte umgebenden, rotierten Rechtecke basiert. Dabei wird ignoriert, wenn eines oder beide Objekte an der Schnittstelle transparent sind und somit optisch gar keine Kollision zu sehen ist. Da es unsere HeldIn auch so schon schwer genug hat, wollen wir das nun verbessern.

1. Erweitert die erste Methode aus Aufgabe 2 so, dass sie die Liste aller Schnittobjekte durchläuft, d.h. die Rückgabe von `getIntersectingObjects` verarbeitet. Erst, wenn für eines der Objekte bestätigt wird, dass wirklich eine Kollision besteht, wird dieses als Ergebnis zurückgeliefert. Besteht zu keinem Objekt eine Kollision, ist die Rückgabe weiterhin `null`.

2. Um festzustellen, ob tatsächlich eine Kollision besteht, muss für jedes nicht-transparente Pixel des einen Objekts (z.B. jenem, das den Kollisionstest durchführt) überprüft werden, ob sich an derselben Stelle (in Weltkoordinaten) in dem anderen Objekt auch ein nicht-transparentes Pixel befindet. Sobald dies der Fall ist, gibt es tatsächlich eine Kollision. Um dies herauszubekommen, müssen die Pixelkoordinaten des Bildes des ersten Objekts in Weltkoordinaten umgerechnet werden und von dort weiter in Pixelkoordinaten des Bildes des zweiten Objekts. Die Pixelkoordinaten lassen sich wie folgt in Weltkoordinaten umrechnen (siehe Abb. 1):

$$\begin{aligned} x_{welt} &= x_{objekt} + \Delta x \cos\theta - \Delta y \sin\theta \\ y_{welt} &= y_{objekt} + \Delta x \sin\theta + \Delta y \cos\theta \\ &\text{wobei} \quad \Delta x = x_{pixel} - \tfrac{w}{2} \\ &\qquad\quad \Delta y = y_{pixel} - \tfrac{h}{2} \end{aligned}$$

Dabei sind x/y_{pixel} die Koordinaten eines Pixels im Objekt-Bild, x/y_{Objekt} die Weltkoodinaten des Objekts, θ seine Rotation, w die Breite des Objektbildes und h seine Höhe. x/y_{welt} sind dann die Weltkoordinaten des Pixels. Die Transformation ins Koordinatensystem des zweiten Objekts (gleiche Bezeichner, aber mit Apostroph, siehe Abb. 1) geht so:

$$\begin{aligned} x'_{pixel} &= \tfrac{w'}{2} + \Delta x \cos\theta' + \Delta y \sin\theta' \\ y'_{pixel} &= \tfrac{h'}{2} - \Delta x \sin\theta' + \Delta y \cos\theta' \\ &\text{wobei} \quad \Delta x = x_{welt} - x'_{objekt} \\ &\qquad\quad \Delta y = y_{welt} - y'_{objekt} \end{aligned}$$

3. Da man bei den Transformationen viel falsch machen kann, baut ihr einen Testmodus ein. Ist dieser aktiv, werden überhaupt keine Kollisionen mehr gemeldet. Stattdessen werden im Bild des Actors, der auf Kollisionen testet, alle Pixel farblich markiert, die mit einem anderen Objekt kollidieren, z.B. in `Color.RED`. Dadurch könnt ihr überprüfen, ob eure Umrechnung funktioniert. Beachtet, dass wegen der Ganzzahlarithmetik Pixel um ± 1 verschoben sein können, mehr sollte es aber nicht sein.

Tipps. Die Winkelfunktionen sind Klassenmethoden der Klasse `Math`. Beachtet, dass diese auf Winkeln in Bogenmaß arbeiten. Die Methode `Math.toRadians` hilft bei der Umrechnung.

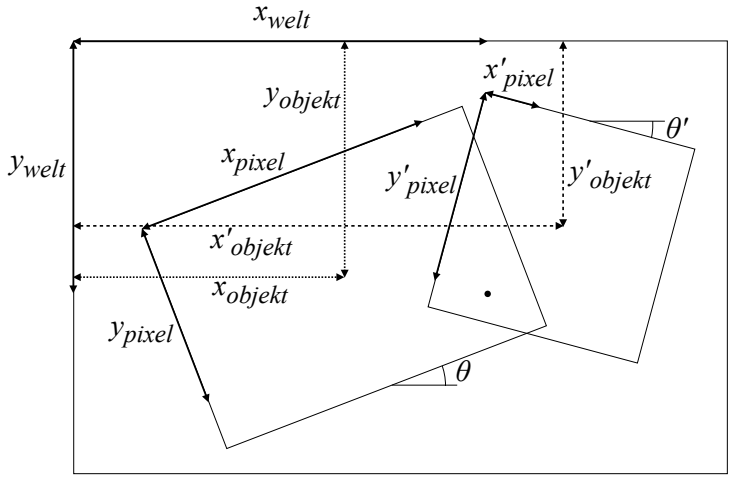

Abbildung 1: Die Längen und Winkel bei der Abbildung der Koordinaten eines Pixels (schwarzer Punkt) aus einem Objekt auf die Pixelkoordinaten in einem anderen Objekt.