Lernaktivität #6 – **Little Crab**

Name: Jasmin

In dieser Lernaktivität wirst du:

1. Lernen, wie man neue Klassen in Greenfoot kreiert.
2. Lernen, wie man die Dokumentation für Greenfoot-Klassen aufruft.
3. Lernen, wie man Kollisionen zwischen zwei `Actors` detektiert.
4. Lernen, wie man neue Methoden definiert

Neue `Actor`-Klassen in Greenfoot kreieren

Öffne das Szenario *little-crab-stride*.

Neue `Actors` müssen eine *Subclass* -- zu Deutsch „Unterklasse“ -- einer bestehenden `Actor`-Klasse sein. In diesem Fall wird unsere neue Klasse, `Worm`, eine dirkete Unterklasse vom `Actor` sein, genau wie die `Crab`-Klasse. „Rechtsklicke“[1] auf die `Actor`-Klasse in der Klassenübersicht und selektiere „*New Subclass“.* Es erscheint einen neuen Dialog. Benenne die neue Klasse „`Worm`“ und wähle das entsprechende Bild. Bemerke, dass nach Konvention Klassen in Java immer großgeschrieben werden, also „`Worm`“ und nicht „`worm`“.

Würmer fressen

Klassendokumentation lesen

Die Krebse sollen die Würmer fressen, wenn sie ihnen begegnen. Wie kann ein `Crab` wissen, dass er einem Wurm begegnet ist, und diesen dann fressen? Wir gucken in die

[1] *Bemerkung: die grammatikalisch korrekte Ausdrücksweise „Klicke mit der rechten Maustaste…“ scheint mir auf Dauer zu umständlich zu sein, weshalb ich hier vereinbare, dass wir, angelehnt am Englischen „right click on ..“ein neues Verb „rechtsklicken “ prägen und zukünftig so verwenden. Ebenso mit „linksklicken“ und „doppelklicken“.*

Dokumentation für die `Actor`-Klasse, um zu sehen, welche Methoden uns hierfür zur Verfügung stehen, die unsere `Crab`-Klasse geerbt hat. Hierzu einfach doppelklicke auf die `Actor`-Klasse in der Klassenübersicht.

Kollisionen zwischen Actors detektieren

Finde in der Dokumentation zu der Klasse `Actor` die beiden Methoden

```
boolean isTouching(java.lang.Class cls)
```

und

```
void removeTouching(java.lang.Class cls)
```

Diese Methoden erwarten eine Java-Klasse als Parameter, also zum Beispiel Worm.class

Verwende diese beiden Methoden, um Deinen Krebs einen Wurm fressen zu lassen, falls er einen trifft. Verteile einige Würmer in die Welt und teste Deine Implementierung.

Beschreibe Deine Implementierung:

Nachdem wir einige Würmer hinzugefügt haben und in der Welt gespeichert haben haben wir programmiert:

```
move(5)
if(isAtEdge())
    turn(Greenfoot.getRandomNumber(360))
if (isTouching(Worm.class))
    removeTouching(Worm.class)
```

Neue Methoden definieren

Um die `Act`-Methode übersichtlich zu halten, kann es sinnvoll sein, teile des Programms in Methoden abzulegen.

Bewege den Frame-Cursor nach unten bis er außerhalb des hellgrünen `act`-Methoden-Blocks ist. Die Hotkey „m“ fügt eine neue Methode ein. Unsere Methode wird ein Befehl sein, also hat sie den Rückgabentyp `void`. Nach Konvention werden Methoden in Java am Anfang kleingeschrieben, und ansonsten mit sogenanntem *CamelCase,* d.h. Wörter zusammengeschrieben, und jeder neue Wortanfang großgeschrieben. So sollte unsere neue Methode wie folgt deklariert werden:

```
public void lookForWormAndEat()
```

Verschiebe den Quelltext von oben aus der `act`-Methode in Deine neue Methode und rufe die neue Methode in der `act`-Methode auf. Teste Deine Implementierung.

Räume Deine `act`-Methode auf, um diese übersichtlicher zu machen, indem Du auch den Quelltext, der den Krebs am Weltrand drehen und willkürliche Bewegungen durchführen lässt, in eigene Methoden verschiebst und entsprechende Methoden-Aufrufe in der `act`-Methode einfügt. Dieser Prozess nennt man *Refactoring.*