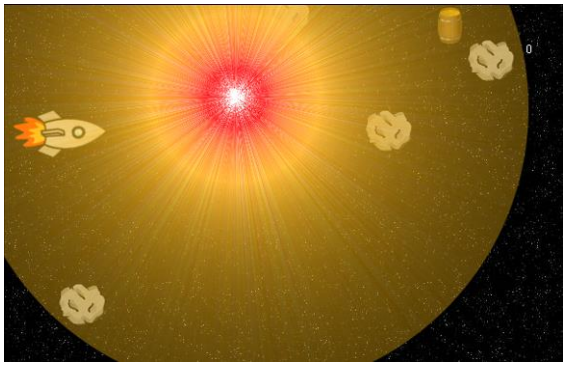
# OOP: Eine Rakete Steuern

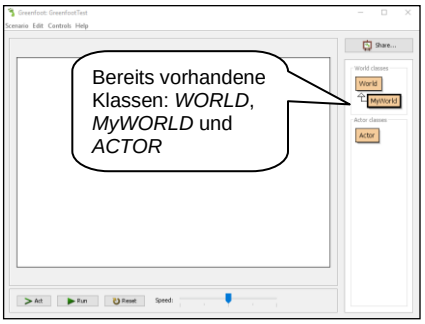
Unser Ziel ist es, ein Spiel zu erstellen, das so ablaufen soll: Wir steuern eine Rakete, die durch einen Meteoriten-Schwarm fliegt. Die Meteoriten bewegen sich von rechts nach links (also auf uns zu). Wenn die Rakete mit einem Meteoriten kollidiert, wird sie natürlich vernichtet. Wir können also ausweichen oder die Meteoriten mit unserer Kanone vernichten. Allerdings haben wir nur wenig Munition. Indem wir Munitions-Fässer aufsammeln die durchs All gleiten, erhalten wir neue Munition.



*Zu Erstellung des Spiels verwenden wir das Programm Greenfoot (www.greenfoot.org). Greenfoot ist eine JAVA-Entwicklungsumgebung, die hauptsächlich dafür vewendet wird, 2D-Animationen und Spiele zu erstellen.*

## Ein Projekt erstellen

- Greenfoot starten
- Scenario / New…
- Dateiname vergeben und speichern



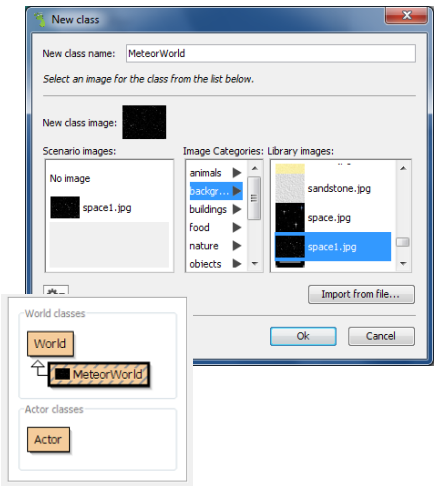

Im rechten Teil des Fensters sehen Sie, dass es bereits drei Klassen gibt: WORLD, MyWORLD und ACTOR.

*Wir erinnern uns: Eine KLASSE ist eine Art Schablone, die beschreibt, was alle Objekte „wissen“ und „können“. In der Klasse WORLD ist z.B. festgelegt, was alle Welten gemeinsam haben (z.B. eine Größe, ein bestimmtes Aussehen, etc.).*

## Ein Welt erstellen

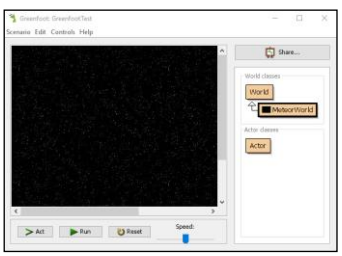
Greenfoot hat für uns bereits eine Welt mit dem Namen MyWORLD erstellt. Weil wir eine völlig neue Welt erschaffen wollen, löschen wir die Klasse MyWORLD:

- RMT (=Rechte Maustaste) auf WORLD
- Delete



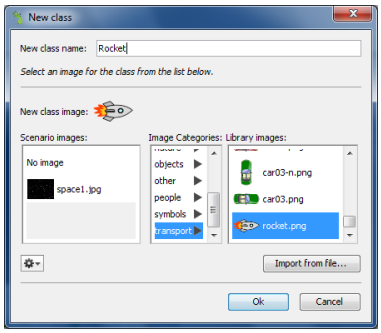

Jetzt können wir unsere eigene Welt erstellen:

- RMT (=Rechte Maustaste) auf WORLD
- New Subclass…
- Image: space1 aus backgrounds,
  New Class Name: MeteorWorld



Im Hauptfenster sehen Sie nun, dass eine neue Klasse erschienen ist (*MeteorWorld*).

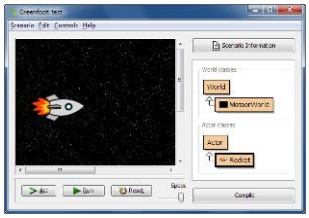
## Eine ACTOR-Klasse erstellen




ACTORs sind die Klassen, die in der Welt agieren, also etwas tun können (z.B. Rakete und Meteoriten). Wir erstellen zuerst eine Rakete:

- RMT auf ACTOR
- New Subclass…
- Image: rocket.png aus transport, Name: Rocket
- Vergessen sie das kompilieren nicht!

Eine Rakete können Sie jetzt so in die Welt einfügen:

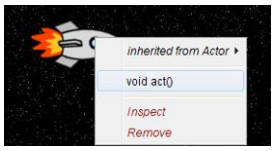


- RMT auf Rocket
- new Rocket()
- Klick auf die Welt: Dort wird eine Rakete eingefügt

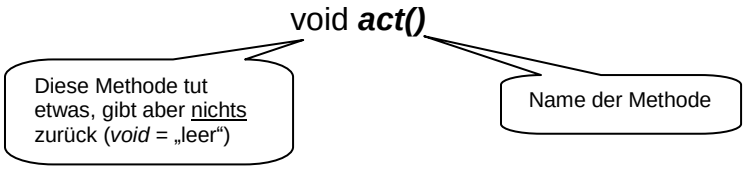
*Machen Sie sich noch mal klar, was gerade eben passiert ist: Sie haben ein* **Objekt** *der* **Klasse Rocket** *erzeugt.*

*Genauer gesagt: Mit* new Rocket() *wird der sogenannte* **Konstruktor** *der Klasse Rocket aufgerufen. Der Konstruktor ist eine besondere Methode, die dafür sorgt, dass man Objekte erzeugen kann.*

## Methoden erstellen: Die Rakete bewegen




Wenn Sie mit der rechten Maustaste auf die Rakete klicken, sehen Sie, dass die Rakete bereits über eine Methode verfügt, nämlich:



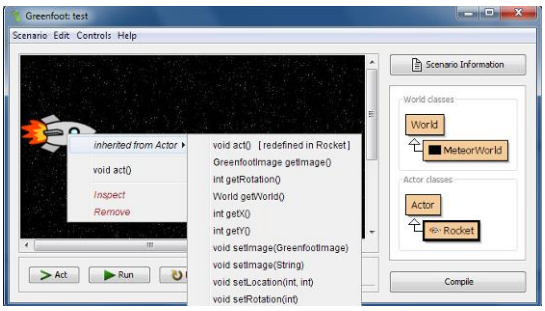

Ein Doppelklick auf die Klasse Rocket bringt uns in die Programmier-Umgebung. Hier können wir festlegen, was passieren soll, wenn jemand die act()-Methode aufruft.

```
public void act()
   {
      // Add your action code here.
   }
```

Die Rakete soll sich nach oben, unten, links und rechts bewegen können. Dies kann man über die *setLocation()*-Methode erreichen.

*Woher kennt unsere Rakete eigentlich die* setLocation*-Methode? Eben haben wir doch noch behauptet, dass es nur die act()-Methode gibt!*

*Des Rätsels Lösung ist relativ einfach: Alle ACTORs kennen eine bestimmte Anzahl an Methoden. Welche Methoden das sind, kann man sich ganz einfach anzeigen lassen, indem man auf der Rakete auf* inherited from Actor *klickt.*

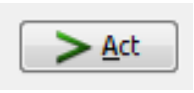



Inherited from Actor *heißt übrigens „geerbt vom Actor“. Hiermit haben Sie eines der wichtigsten Konzepte der objektorientierten Programmmierung kennen gelernt: Eine Subclass (die Rakete) erbt alles von der Super Class (Actor).*

Wir experimentieren mit der setLocation-Methode, indem wir diese so ändern:

```
 public void act()
{
   this.setLocation(50, 100);
}
```

Klicken Sie anschließend auf die Schaltfläche **Act**.
→ Die Rakete bewegt sich auf die die Koordinaten: x = 50, y = 100
(Achtung: Der Ursprung (0 | 0) liegt beim Computer immer links oben, weil das in einem Fenster der einzige fixe Punkt ist. Das Koordinatensystem ist also gekippt. Nach unten werden die y-Koordinaten größer!)




## Eine konstante Bewegung

In der obigen Lösung bewegt sich die Rakete zum angegebenen Punkt. Ab dann verändert sie ihre Position nicht mehr. Damit die Rakete jedes Mal die Position ändert, wenn die act()-Methode aufgerufen wird, verändern wir diese so:

```
public void act()
{
   this.setLocation(this.getX(),this.getY() + 2);
}
```

Wenn Sie immer wieder auf die Act-Schalftläche klicken, bewegt sich die Rakete immer weiter nach unten. Alternativ können Sie auch auf die Run-Schaltfläche klicken, die nichts weiter tut, als die act()-Methode immer wieder aufzurufen.

## Tastatur-Steuerung

In der obigen Lösung bewegt sich die Rakete zum angegebenen Punkt. Ab dann verändert sie ihre Position nicht mehr. Damit die Rakete jedes Mal die Position ändert, wenn die act()-Methode aufgerufen wird, verändern wir diese so:

```
public void act()
 {
    if(Greenfoot.isKeyDown("down"))
    {
       this.moveDown();
    }
 }

 public void moveDown()
 {
   this.setLocation(getX(), getY() + 2);
 }
```

## Arbeitsauftrag

Erweitern Sie die Methode ***act()*** so, dass die Rakete über die Pfeiltasten auch nach oben, links und rechts gesteuert werden kann!