
JSGLib Tutorial 1: Ladybug

As a very first example we let a ladybug eat cloverleaves.

Setting up the Program Class

The package (directory) is supposed to be named *ladybug*. It contains a class *Program* with the *main* method. Note, that while Eclipse adds the package line automatically, you have to add it yourself in Java-Editor (case matters, so best always name packages lower-case). With the import command we include everything from the *jsglib* package.

```
package ladybug;

import jsglib.*;

public class Program {
    public static void main(String[] args) {
    }
}
```

The following instructions are added to *main*.

Creating the Stage

The *Stage* class provides a window, input handling and simple counters.

Create the stage with

```
Stage s = new Stage("Ladybug",800,600,"sand.jpg");
```

The first parameter is the window caption, followed by window size and background image. You can change the background later with *Stage.loadBackground(String s)*. See *images* directory for all the images available (backgrounds are *jpgs*) . You can add own images, of course.

Creating the Ladybug

The *StageObject* class provides objects that can be moved, rotated, scaled, change their appearance, ...

Create the ladybug with

```
StageObject b = new StageObject(s,400,300,"ladybug.png",20);
```

There are different constructors for *StageObject* (see doc), this is the most common one. First, you specify the stage and position, then the image and optionally a scaling factor (100 is 100%). So we scale the image down to 20% of the original size.

Make the Ladybug follow the Pointer

Add the following while-loop:

```
while (true) {
```

JSGLib Tutorial 1: Ladybug

```
        b.lookAtPointer();
        b.move(6);
        Tools.wait(20);
    }
```

lookAtPointer instantly rotates the object so that it looks towards the mouse pointer. *move* moves an object forward in the current looking direction. The given steps are in pixels. Angles work as in math: at 0° an object looks eastwards. A positive angle means counterclockwise rotation.

The *Tools* class provides static helper methods, e.g., *wait* puts the thread to sleep for 20 ms. Note, that *Stage*'s repaint timer task is scheduled at 20 ms, so in terms of frame rate it does not make sense to go below this value unless you also reduce *Stage.repaintDelay*. Generally 20 ms is a good delay so you should work this this.

Feed the Bug

Now, add the instructions written in bold:

```
StageObject l = new StageObject(s, 600, 300, "cloverleaf.png", 20);

while (true) {
    b.lookAtPointer();
    b.move(6);
    if (b.touches(l)) {
        l.moveTo(Tools.rand(50, 750), Tools.rand(50, 550));
        b.scaleUp(2);
    }
    Tools.wait(20);
}
```

We create another object showing a cloverleaf. When bug and leaf touch each other (you can also use method *checkCollision* which does the same whatever name suites you better) the cloverleaf is moved to a random position (*Tools.rand(int min, int max)* returns a random integer number between and including *min* and *max*). *scaleUp* increases the object size by a percentage of the original image size.

Note, that the collision check right now is very basic and assumes circular objects. It works fine when width and height are almost the same but with stretched out objects you will get false hits.