Greenfoot
An Introduction to OOP

Presented by
Adrienne Decker
Stephanie Hoeppner
Fran Trees

Greenfoot

- a combination between a framework for creating two-dimensional grid assignments in Java and an IDE.
- a tool for teaching programming with the Java language.
  - Intended for ages 14 and older.
  - Download from: http://www.greenfoot.org/
  - It has interface translations into languages including French, German and Italian.

Session Description

- In this presentation, we will introduce Greenfoot.
- This presentation is aimed at teachers of introductory Java programming courses (high schools and universities) who have never worked with the Greenfoot environment before.

Presentation Description

- This presentation is intended to give educators an introduction to the Greenfoot environment, a demonstration of how it can be used to introduce object-oriented programming to students, and a guided approach to developing Greenfoot experiences that can be integrated into existing curricula.
- The session is practically oriented and allows participants to use Greenfoot in their classroom immediately.

Session Goals

- To introduce the Greenfoot environment
  - A source code editor
  - A class browser
  - Compilation control
  - Execution control
- While discussing the teaching of
  - Java through examples

Agenda

- Building a Greenfoot program (to learn about Greenfoot) CRABS, WORMS, and LOBSTERS
  - Start with an existing scenario
  - Introduce objects and classes
  - Work with interacting classes
  - Look at movement in the world
  - Include random behavior and sound

Agenda

- Moving beyond the first scenario: Demonstration of some advanced features of Greenfoot:
  - Image control
  - Animation
  - Collision Detection

Agenda

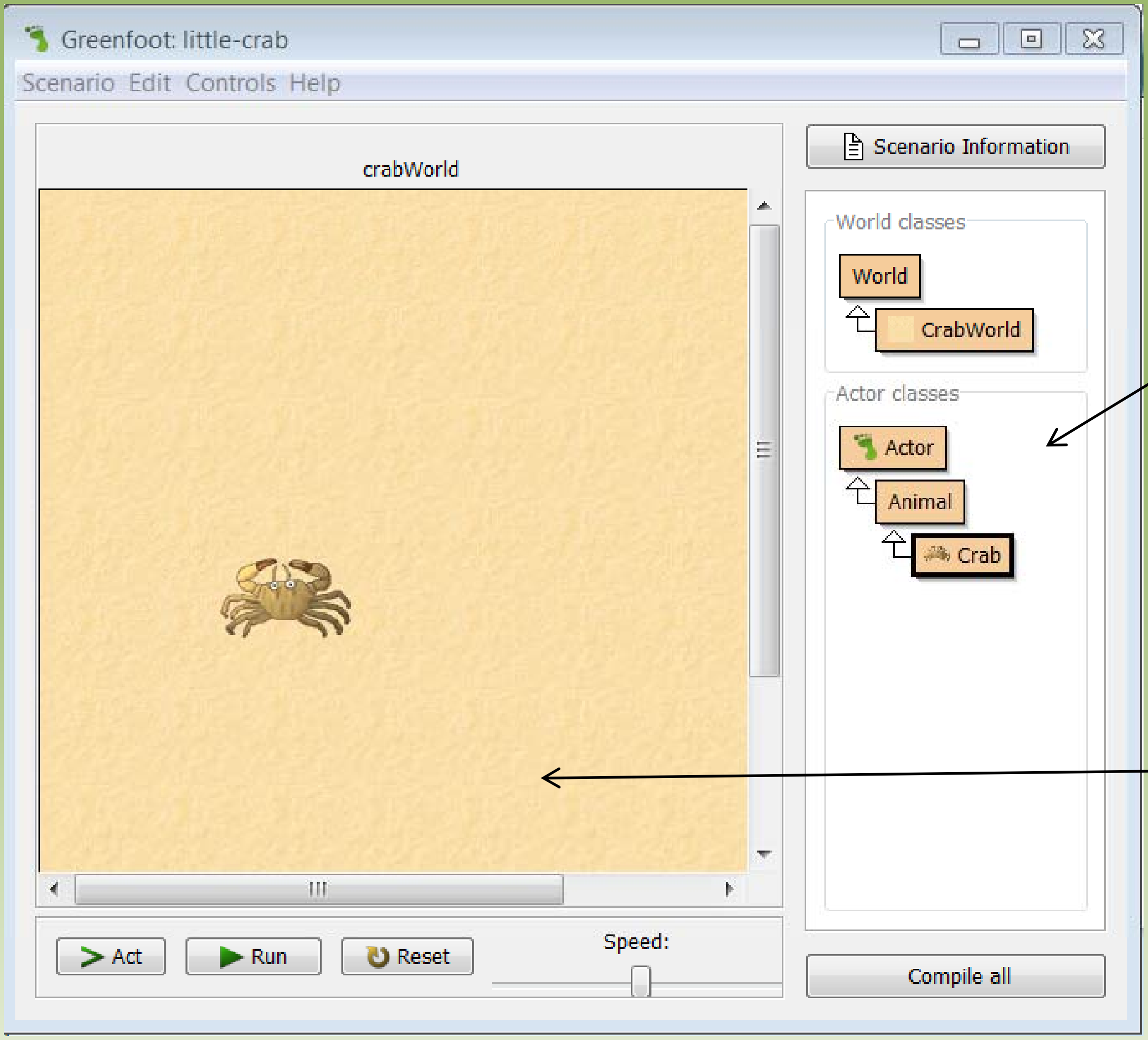
- Some examples using Greenfoot to teach or review CS topics
- Greenfoot.org
  - Publishing Scenarios
  - Available Resources
- Questions?????

Greenfoot (Little Crab Scenario)

- Like BlueJ, Greenfoot teaches object-oriented programming in a visual manner. Each actor is an object that moves around in a world (also an object).

- This allows teaching of object-oriented principles (method invocation, object state) before even beginning to look at or write code.

Greenfoot Environment



Class Diagram

World

Greenfoot classes

(Help Option)

Actor

GreenfootSound

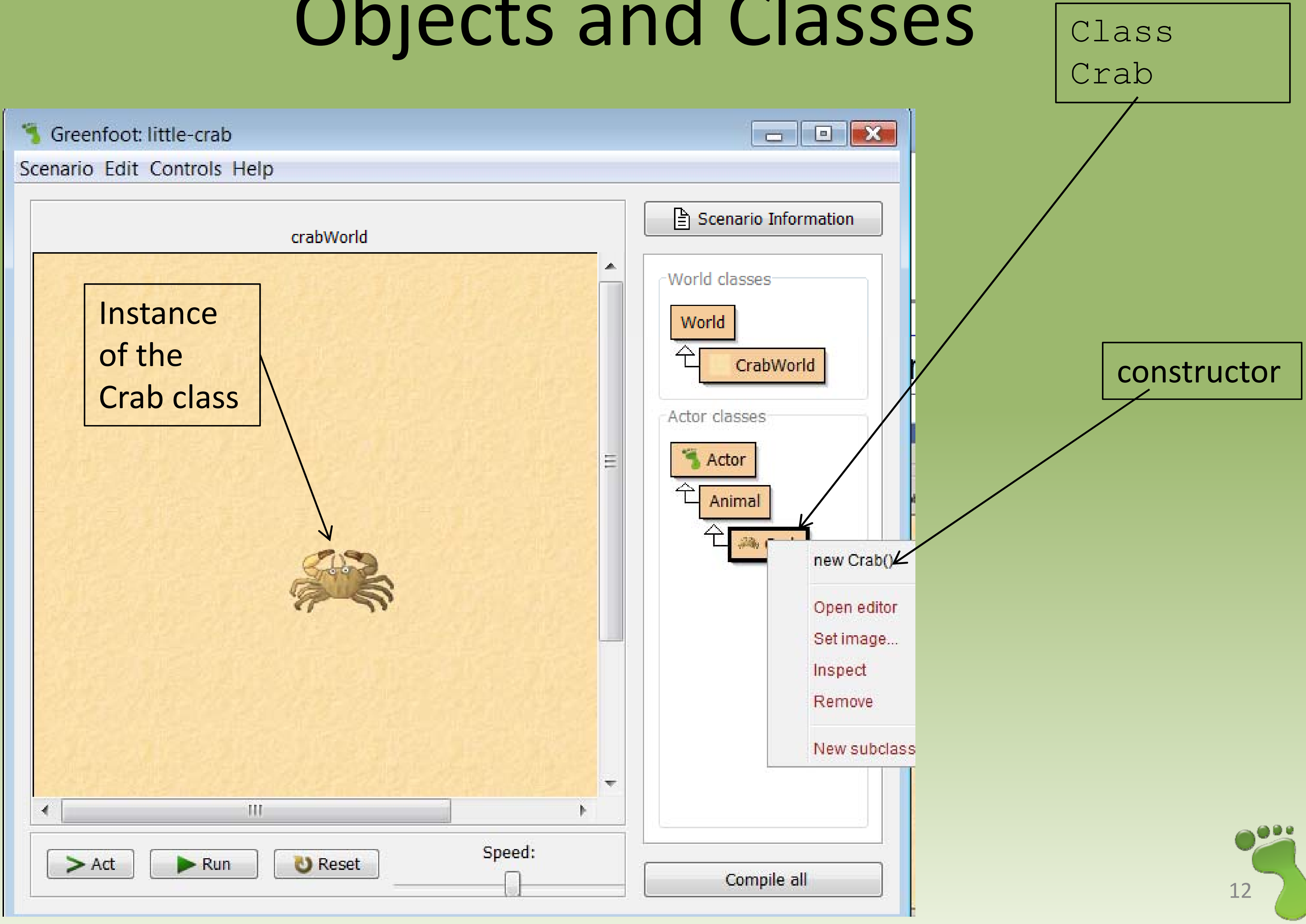
Greenfoot

MouseInfo

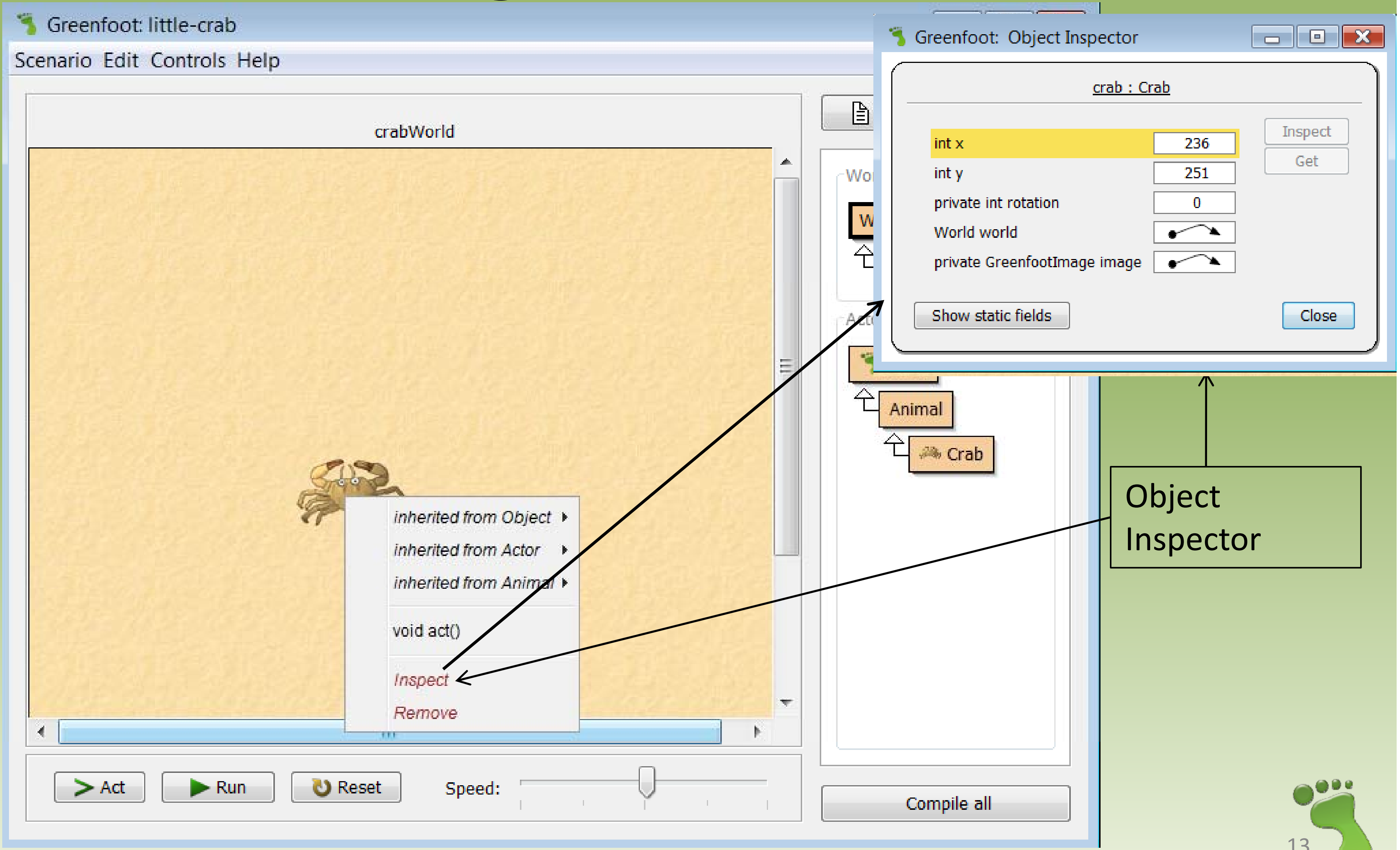
GreenfootImage

World

Objects and Classes



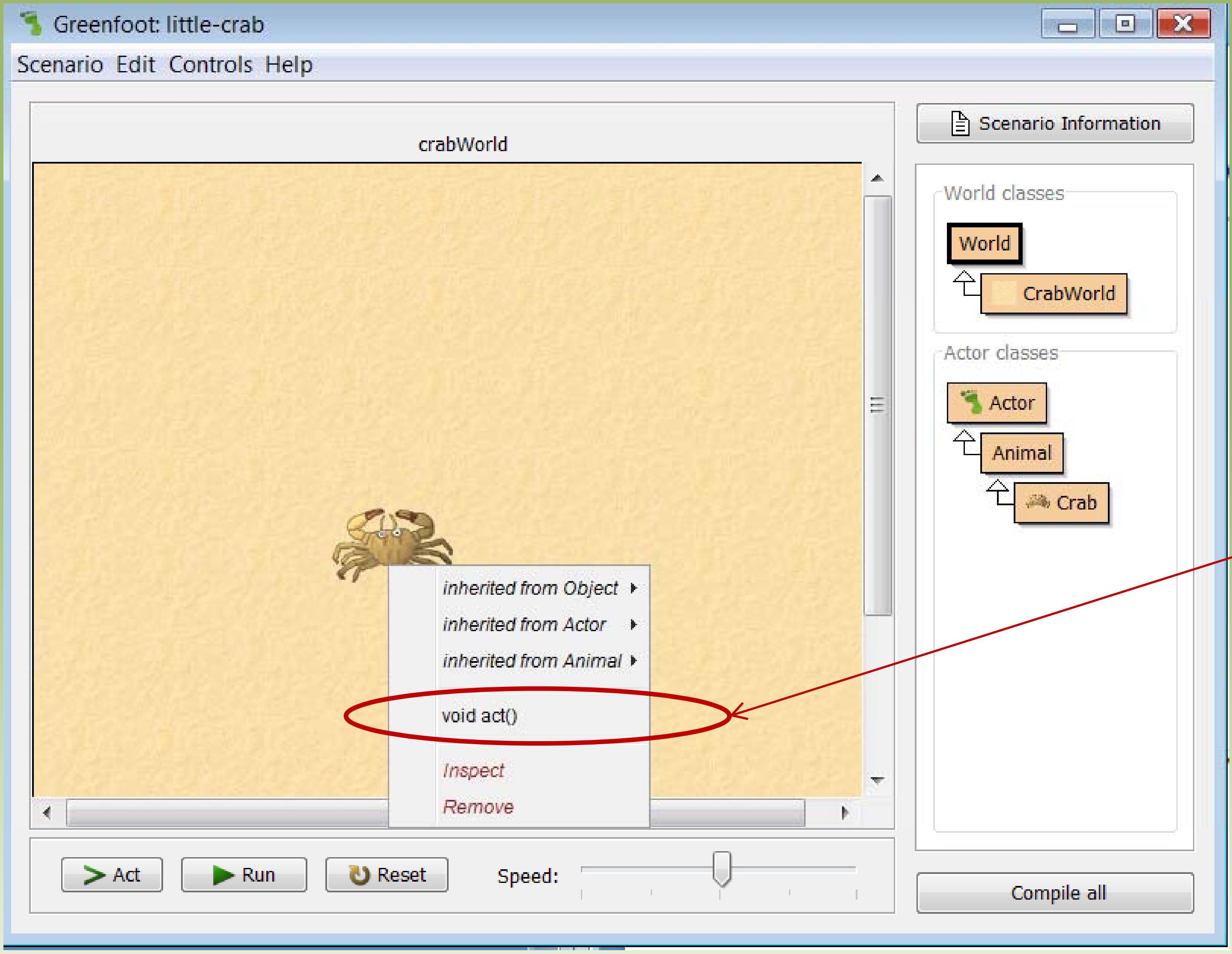
Defining state (attributes)



int x	236
int y	251
private int rotation	0
World world	
private GreenfootImage image	

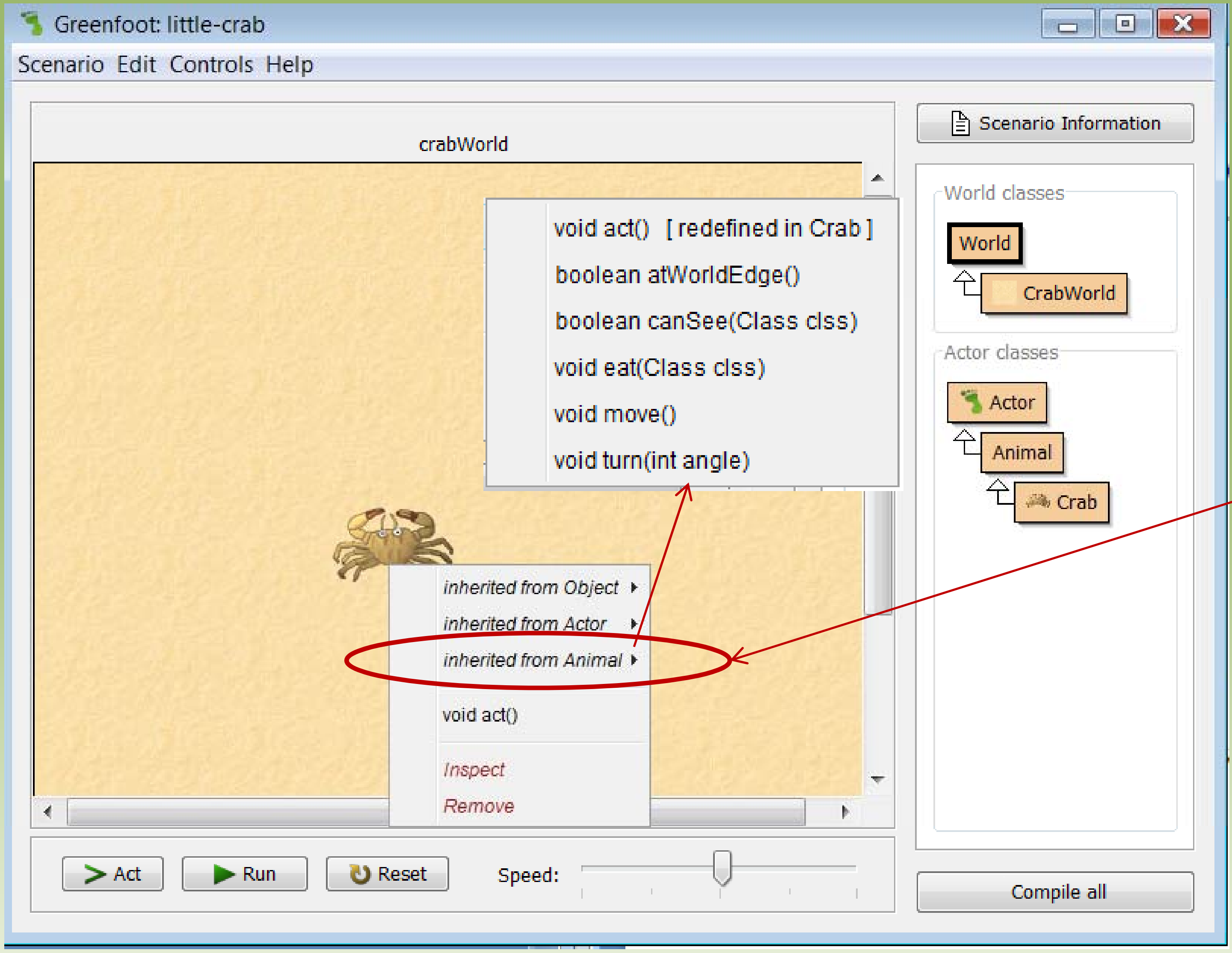
Object Inspector

Defining behavior (methods)



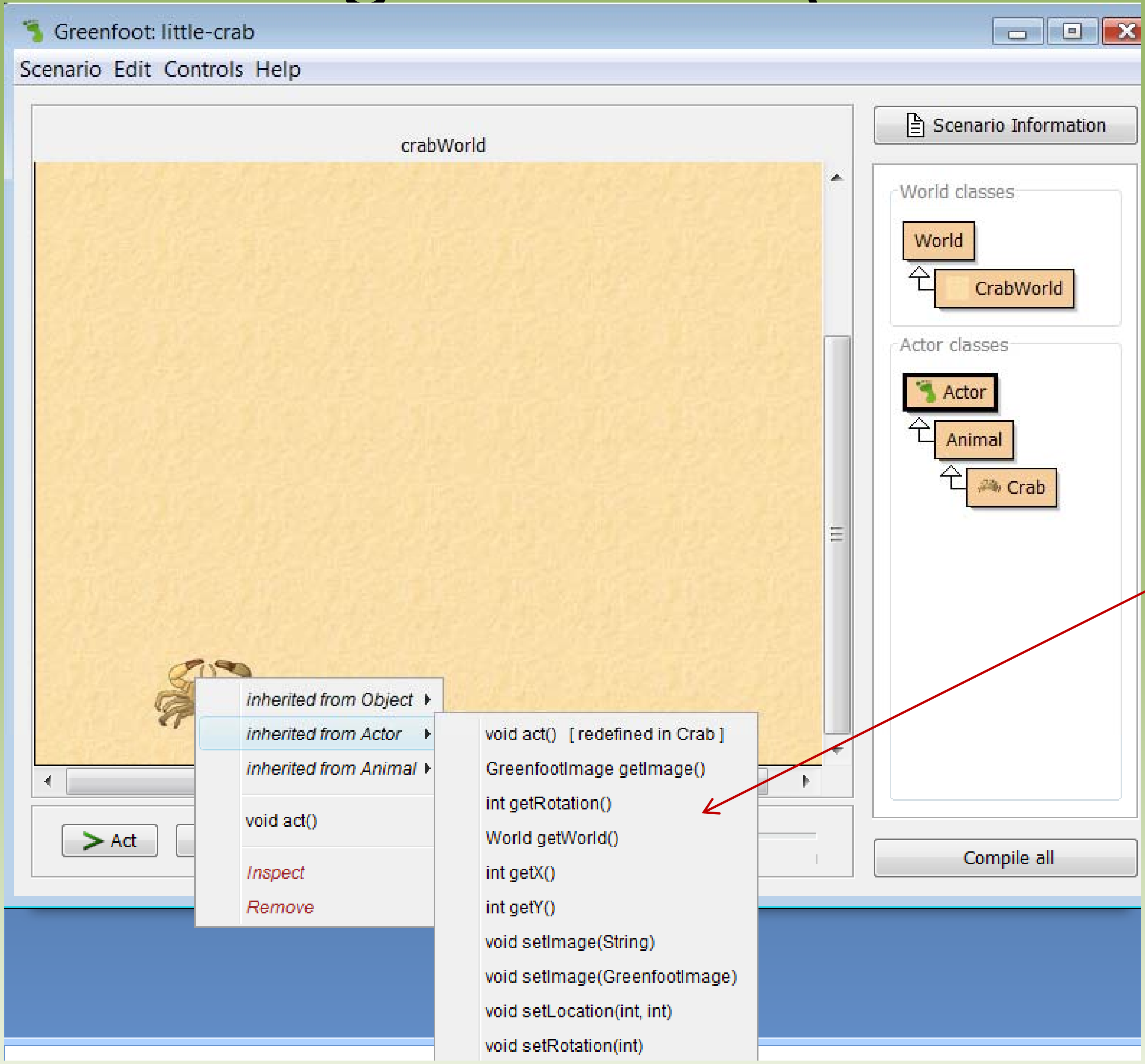
Crab methods

Defining behavior (methods)



Methods inherited from `Animal`

Defining behavior (methods)



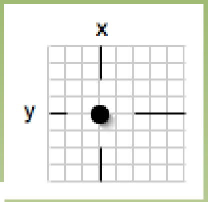
void act() [redefined in Crab]
GreenfootImage getImage()
int getRotation()
World getWorld()
int getX()
int getY()
void setImage(String)
void setImage(GreenfootImage)
void setLocation(int, int)
void setRotation(int)

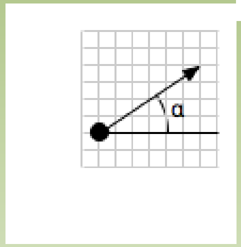
Methods inherited from `Actor`

Actors

'Actors' have predefined state:

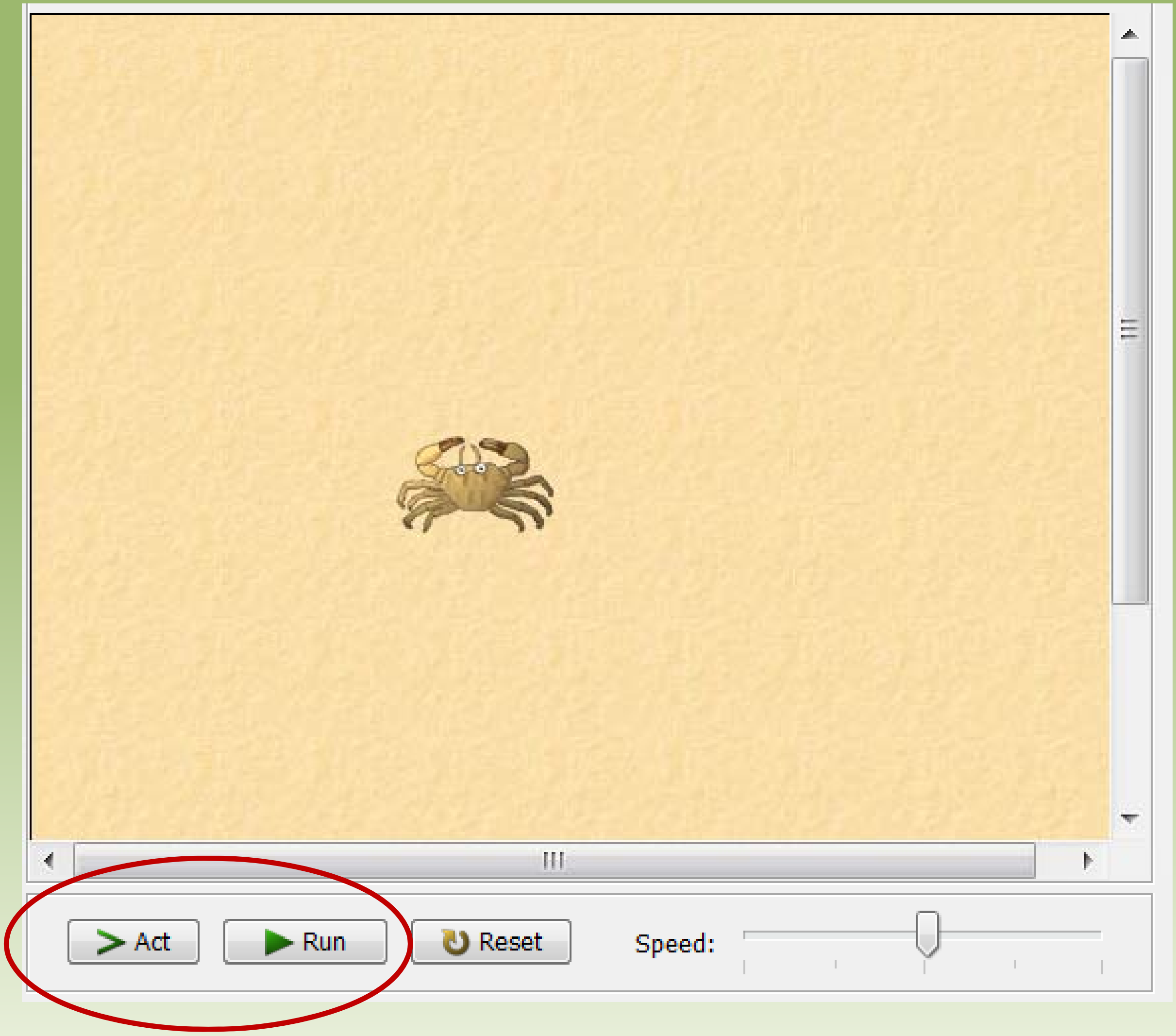
- image
- location (in the world)
- rotation





Create a new Crab and press " >Act "

Click "►Run" to call act continuously



Nothing happens!

Let’s investigate the Java code for `act`!

```
import greenfoot.;  // (World, Actor,
  GreenfootImage, and Greenfoot)

/
  This class defines a crab. Crabs live on the
  beach.
 /
public class Crab extends Animal
{
    public void act()
    {
    }
}
```

The crab does nothing when it acts!

Defining the Crab's behavior for act

```
public class Crab extends Animal
{
    public void act()
    {
        move();
     }
}
```

- Compile
- ▶Run
- What happens?

Movement-Key Point

- You can choose the level of abstraction/complexity to expose to your students by preparing the scenario.

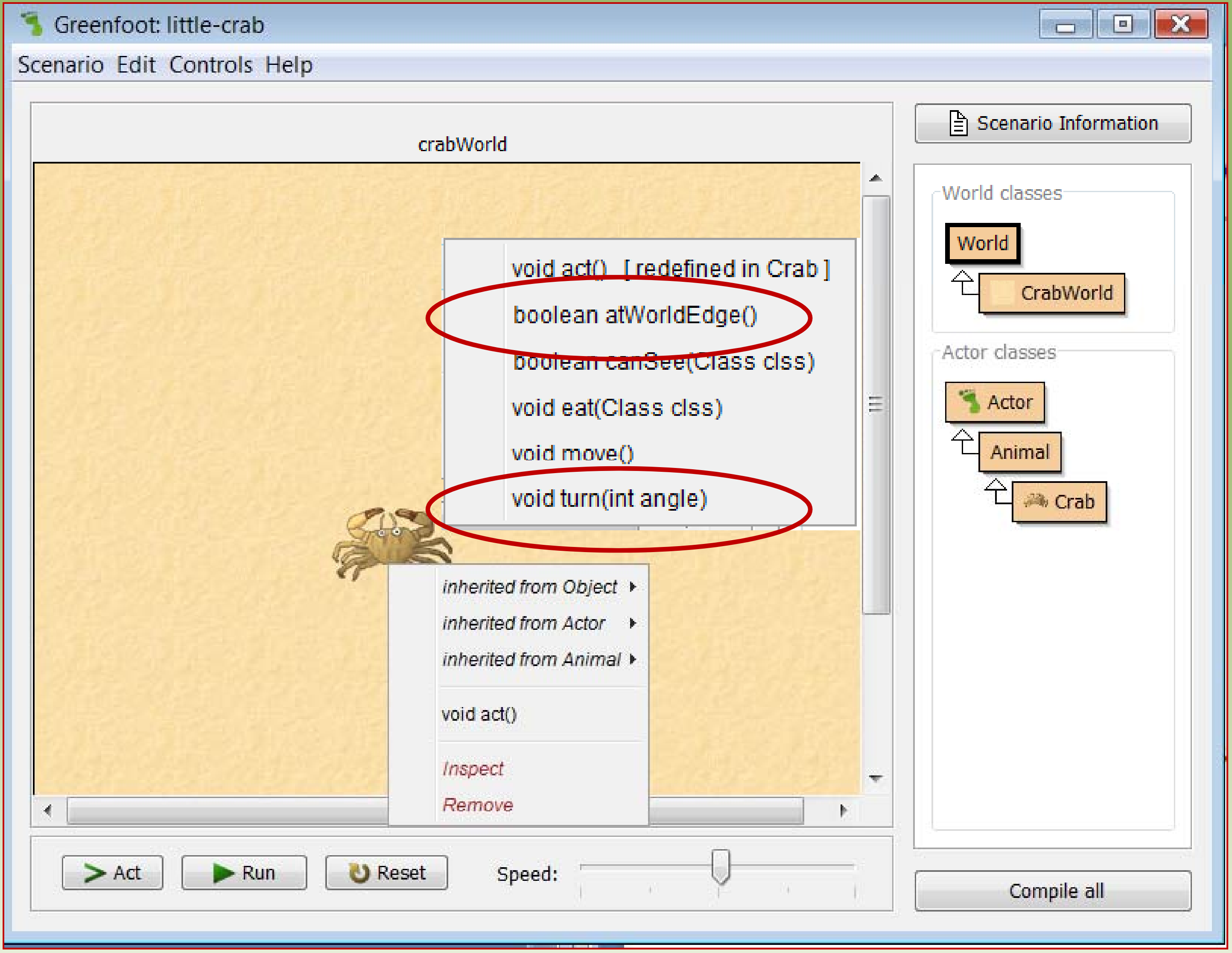
- `move();`
- `setLocation( getX()+1, getY() );`

Defining the Crab's behavior for `act`

- *If crab is at the edge, turn around and go the other way.*

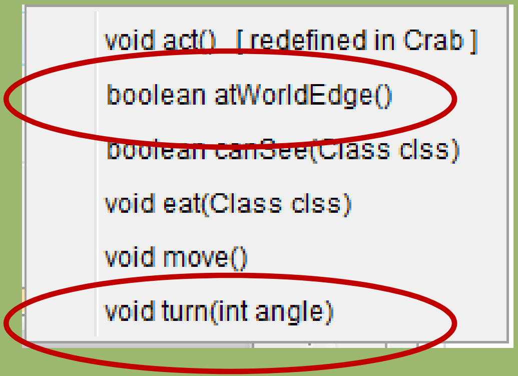
```
public class Crab extends Animal
{
    public void act()
    {
        move();
       // more code here

     }
}
```

- *If crab is at the edge, turn around and go the other way.*



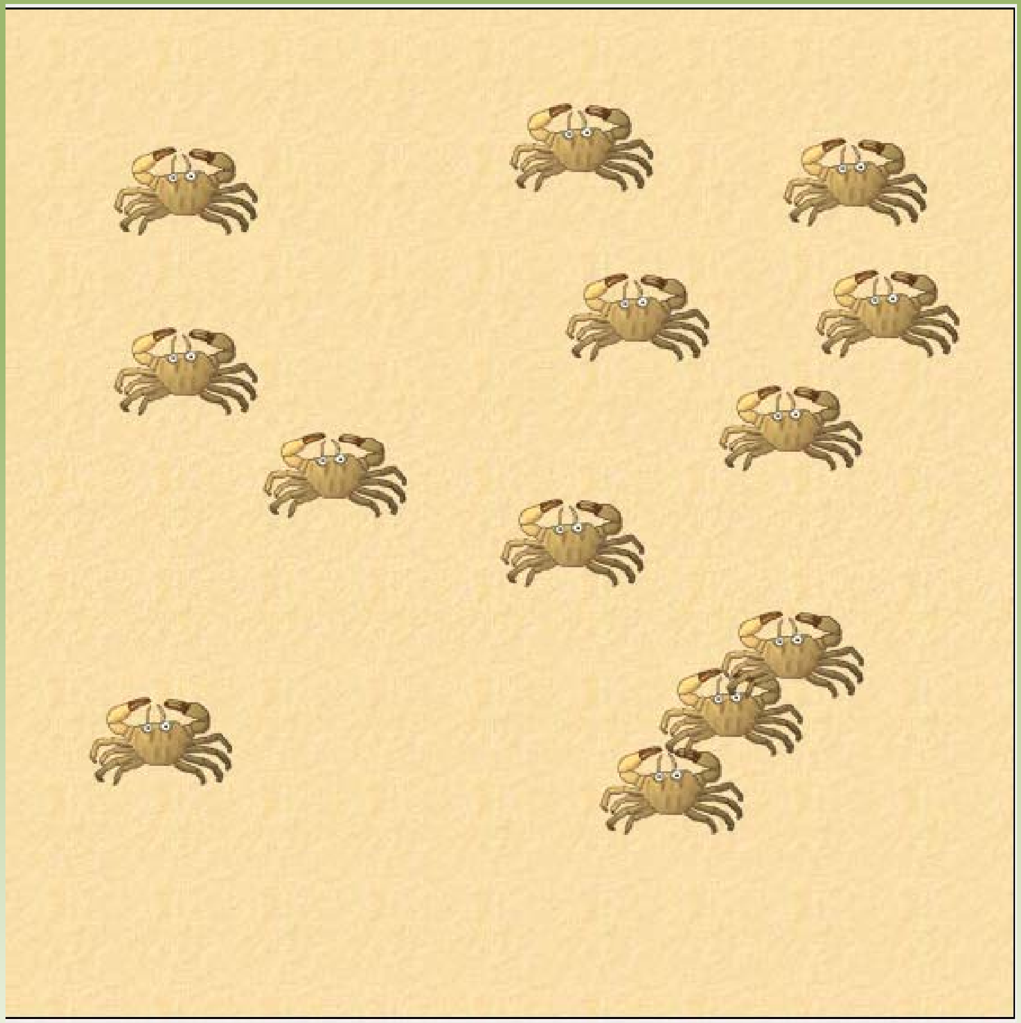
```
void act()  [ redefined in Crab ]
boolean atWorldEdge()
boolean canSee(Class clss)
void eat(Class clss)
void move()
void turn(int angle)
```

- *If crab is at the edge, turn smoothly and go the other way!*


void act() [redefined in Crab]
boolean atWorldEdge()
boolean canSee(Class clss)
void eat(Class clss)
void move()
void turn(int angle)

```
public void act()
    {
        move();
        if( atWorldEdge() )
        {
            turn(15);
        }
     }
```

Objects and Classes
Populate your world with crabs



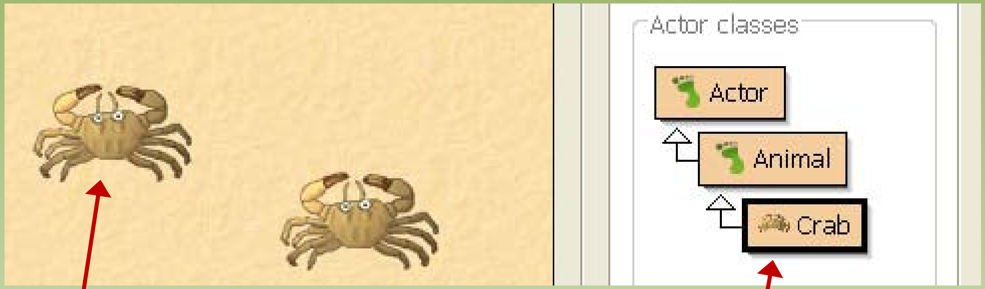
Each instance of a `Crab` has its own attributes (state).

Objects and Classes

The description of what comprises an ***object*** of a particular type is a ***class***

A ***class*** defines the ***behavior*** and ***attributes (characteristics)*** of objects (what an object knows about itself).

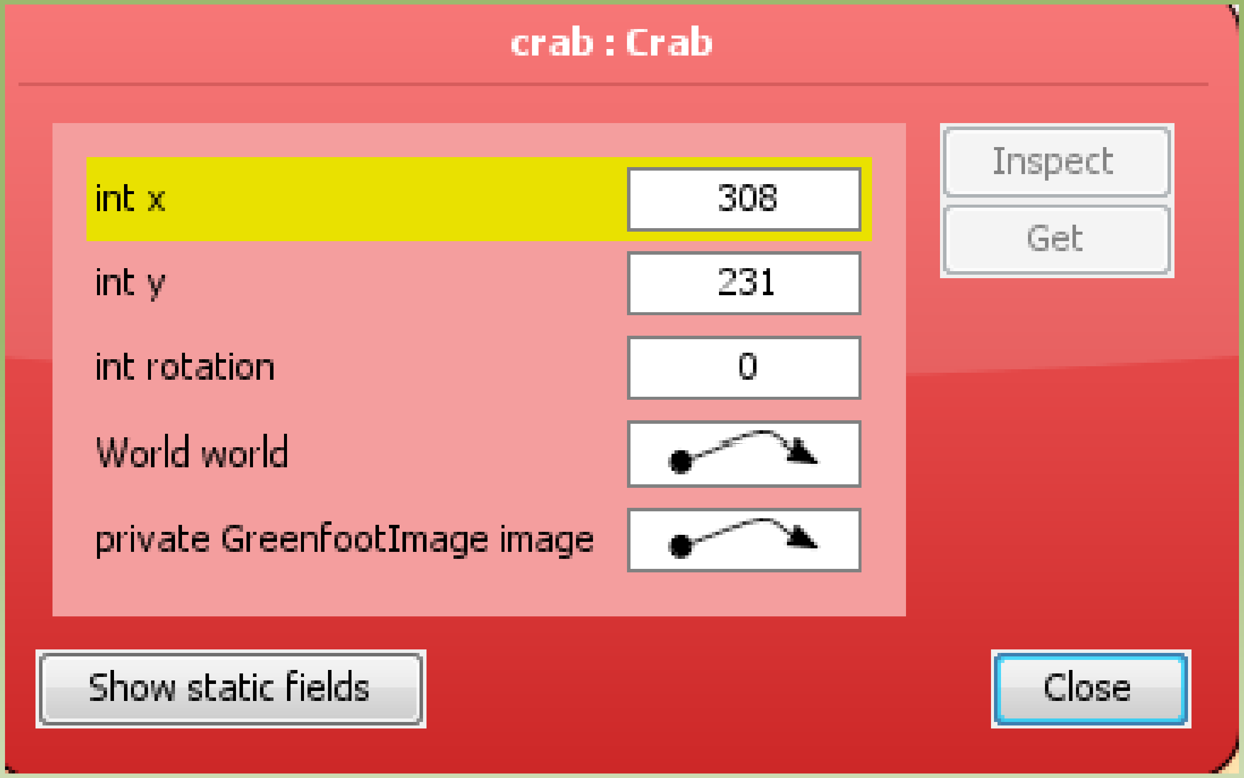
Objects are ***instances*** of a class.



`Crab` ***objects***

`Crab` ***class***: blueprint for crabs

What a crab knows about itself:


crab : Crab

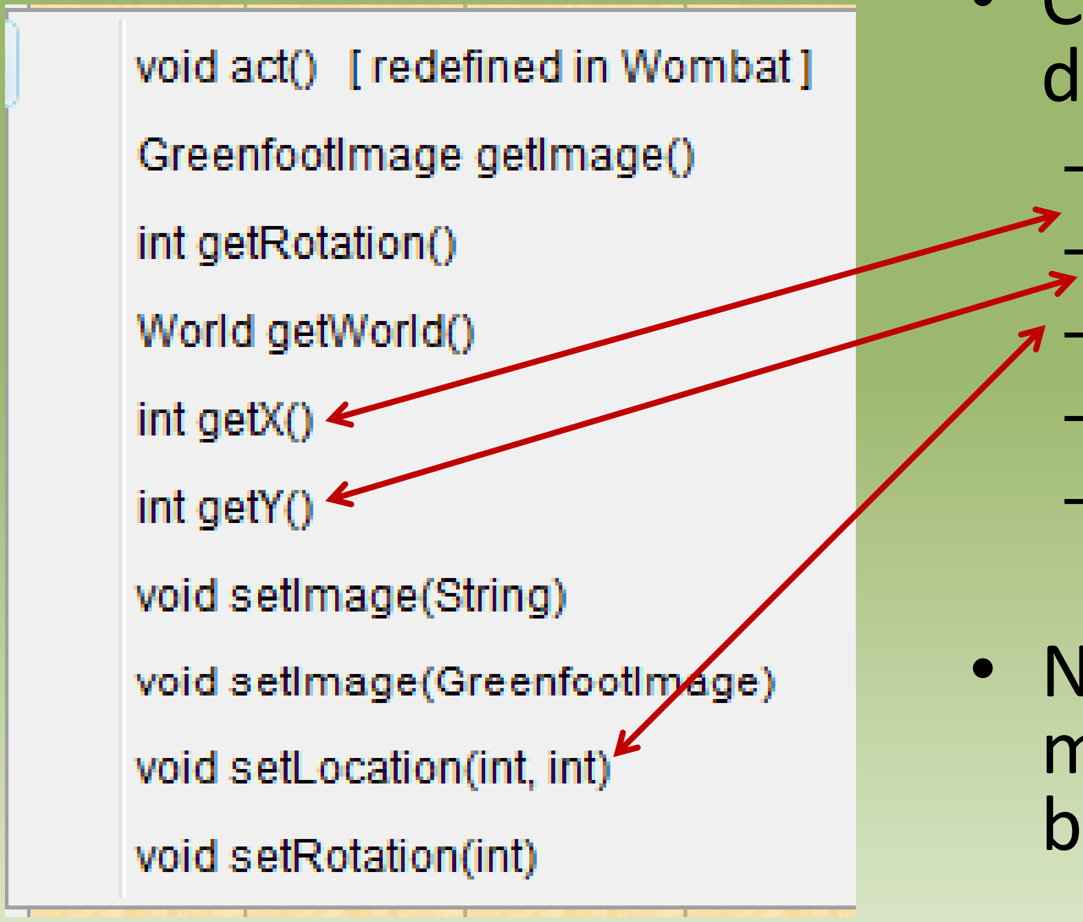
int x	308
int y	231
int rotation	0
World world	
private GreenfootImage image	

Crabs can not directly access:

- x
- y
- Rotation
- World
- image

Need *accessors* and *mutators* provided by `Actor` to do that.

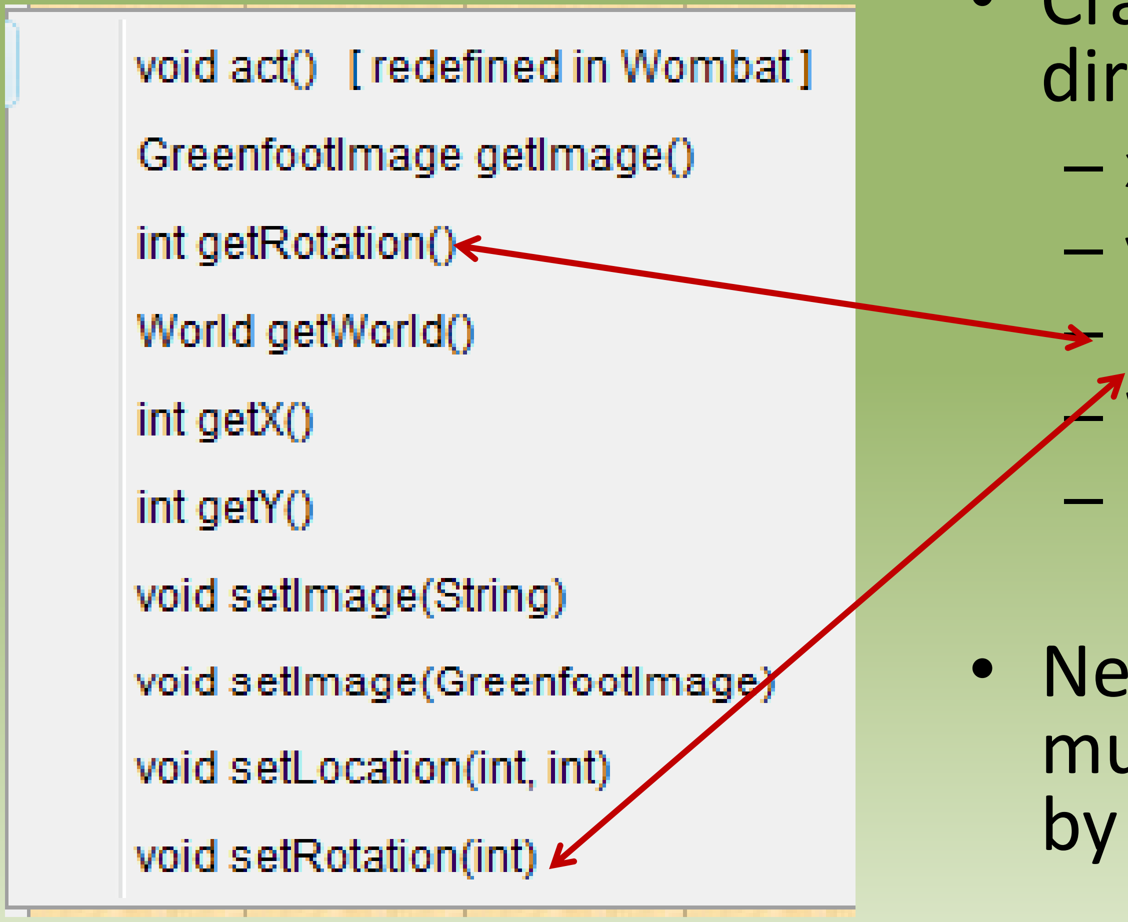
`Crab` inherits from `Actor`



```
void act()   [ redefined in Wombat ]
GreenfootImage getImage()
int getRotation()
World getWorld()
int getX()
int getY()
void setImage(String)
void setImage(GreenfootImage)
void setLocation(int, int)
void setRotation(int)
```

- Crabs can not directly access:
  - x
  - y
  - Rotation
  - World
  - image
- Need accessors and mutators provided by Actor to do that.

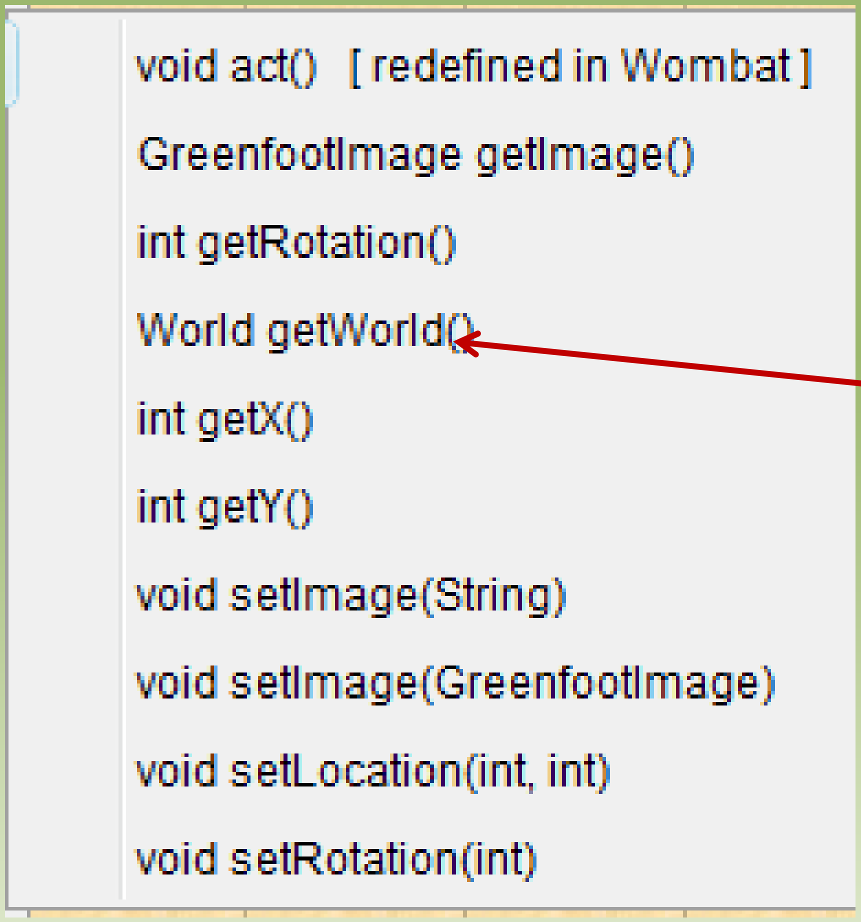
Crab inherits from Actor



```
void act()   [ redefined in Wombat ]
GreenfootImage getImage()
int getRotation()
World getWorld()
int getX()
int getY()
void setImage(String)
void setImage(GreenfootImage)
void setLocation(int, int)
void setRotation(int)
```

- Crabs can not directly access:
  - x
  - y
  - Rotation
  - World
  - image
- Need accessors and mutators provided by `Actor` to do that.

`Crab` inherits from `Actor`

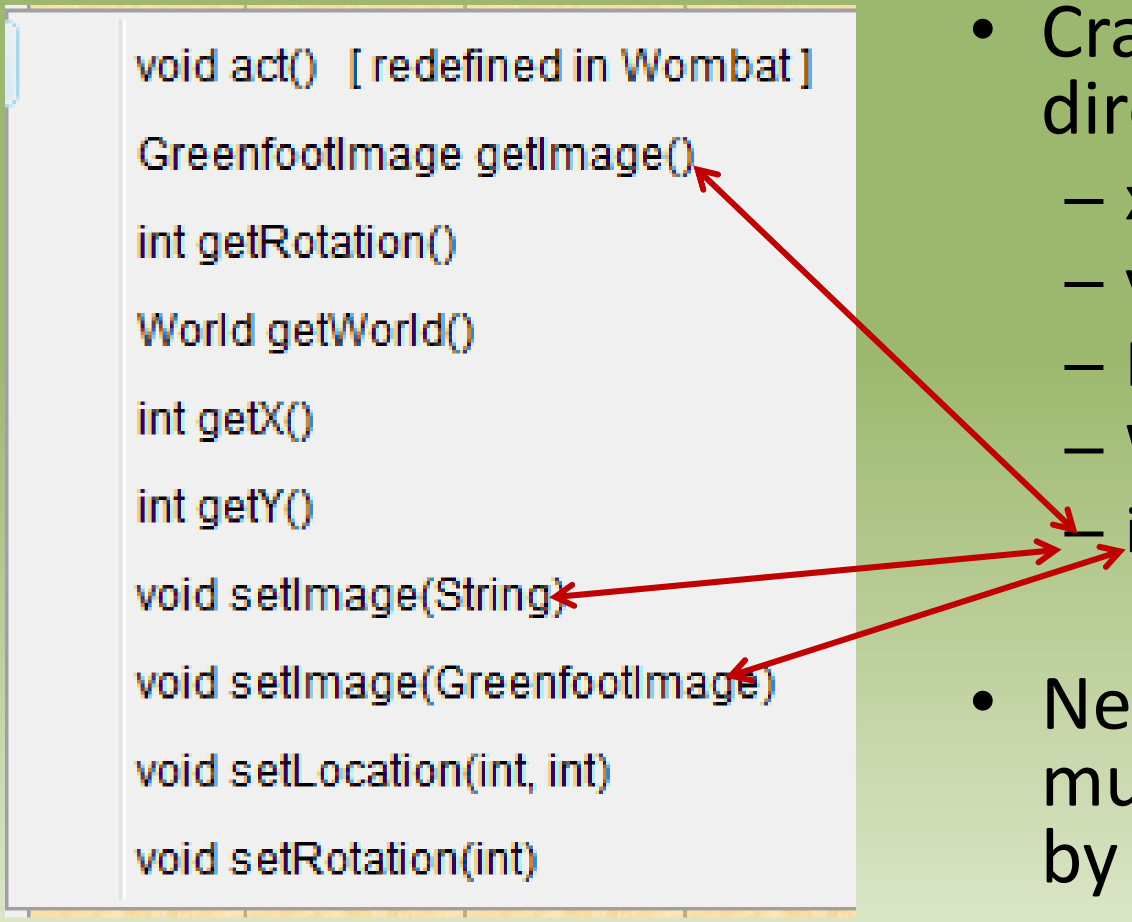


```
void act()   [ redefined in Wombat ]
GreenfootImage getImage()
int getRotation()
World getWorld()
int getX()
int getY()
void setImage(String)
void setImage(GreenfootImage)
void setLocation(int, int)
void setRotation(int)
```

- Crabs can not directly access:
  - x
  - y
  - Rotation
  - World
  - image

- Need accessors and mutators provided by `Actor` to do that.

Crab inherits from Actor


```
void act()   [ redefined in Wombat ]
GreenfootImage getImage()
int getRotation()
World getWorld()
int getX()
int getY()
void setImage(String)
void setImage(GreenfootImage)
void setLocation(int, int)
void setRotation(int)
```

- Crabs can not directly access:
  - x
  - y
  - Rotation
  - World
  - image
- Need accessors and mutators provided by `Actor` to do that.

Objects and Classes

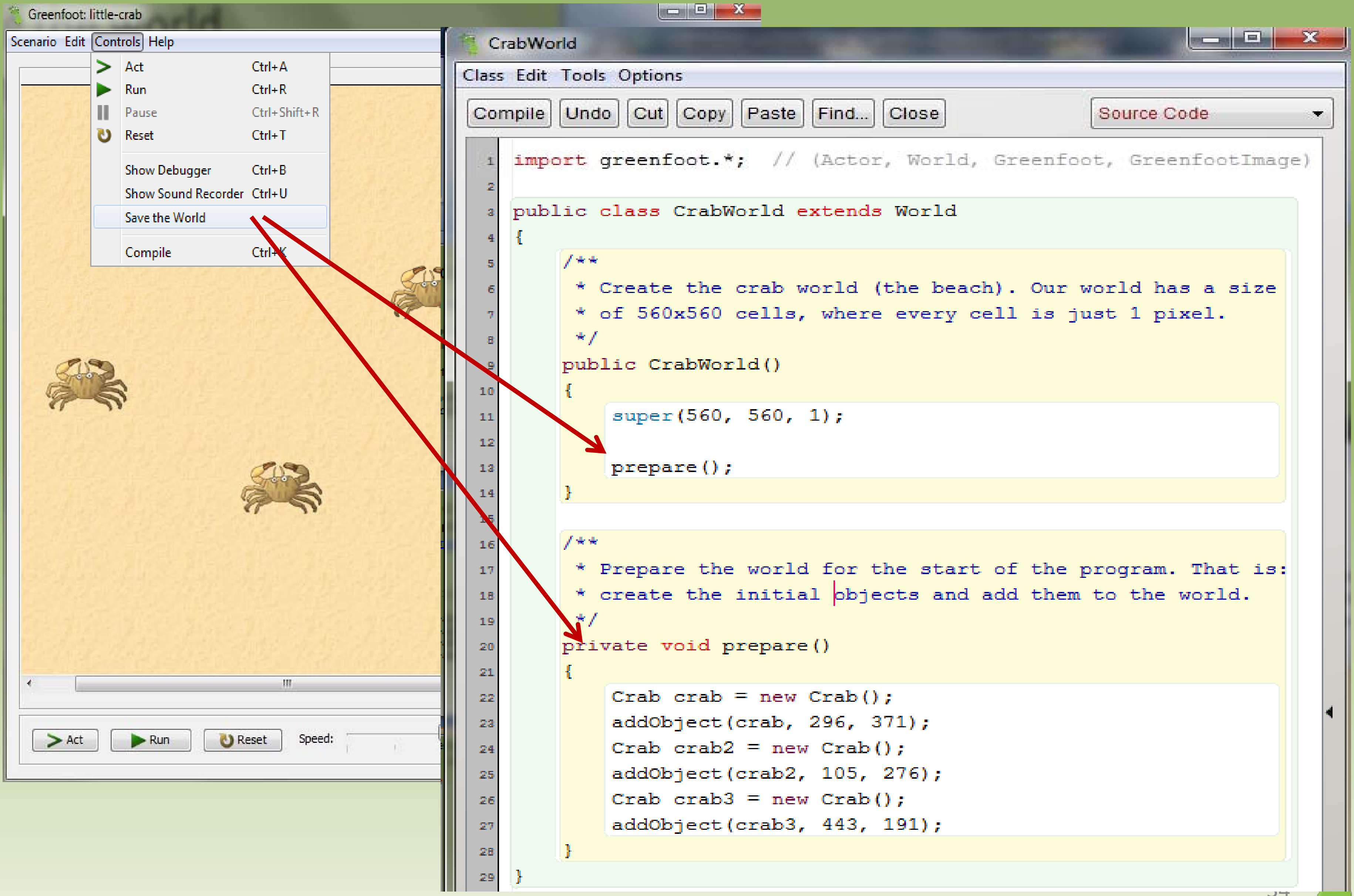
- Populate your world with a few instances of the `Crab` class.
- Can there be more than one object in a position?
- Can a `Crab` move to a position that another `Crab` occupies?
- Can a `Crab` move outside the boundaries of the world?
- How is invoking the `act` method of a `Crab` different from clicking the *act* execution control button.

Populating the world (Using the Greenfoot API)

http://www.greenfoot.org/doc/javadoc/

```
public CrabWorld()
{
        super(560, 560, 1);

        addObject(new Crab(), 100,200);
        addObject(new Crab(), 150,290);
        addObject(new Crab(), 300,400);
        addObject(new Crab(), 500,50);

}
```

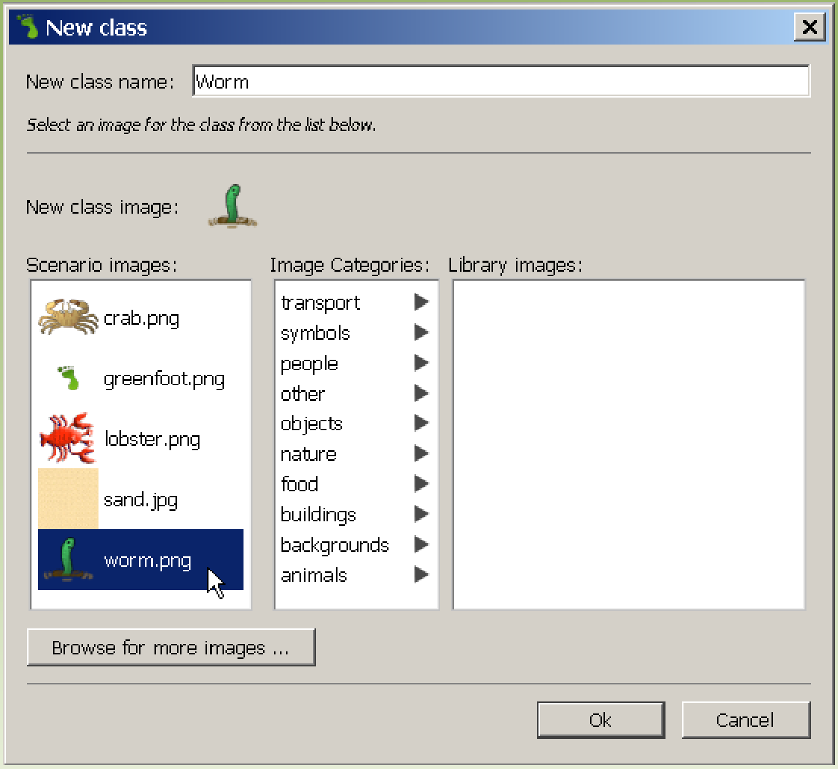
Save the World!



```
import greenfoot.*;  // (Actor, World, Greenfoot, GreenfootImage)

public class CrabWorld extends World
{
    /**
     * Create the crab world (the beach). Our world has a size
     * of 560x560 cells, where every cell is just 1 pixel.
     */
    public CrabWorld()
    {
        super(560, 560, 1);

        prepare();
    }

    /**
     * Prepare the world for the start of the program. That is:
     * create the initial objects and add them to the world.
     */
    private void prepare()
    {
        Crab crab = new Crab();
        addObject(crab, 296, 371);
        Crab crab2 = new Crab();
        addObject(crab2, 105, 276);
        Crab crab3 = new Crab();
        addObject(crab3, 443, 191);
    }
}
```

Interacting classes
The crabs are hungry!

- Crabs eat worms.
  1. Right click Animal; choose "**`New Subclass...`**"
  2. Name it "Worm".
  3. Choose an image.



New class

New class name: Worm

Select an image for the class from the list below.

New class image:

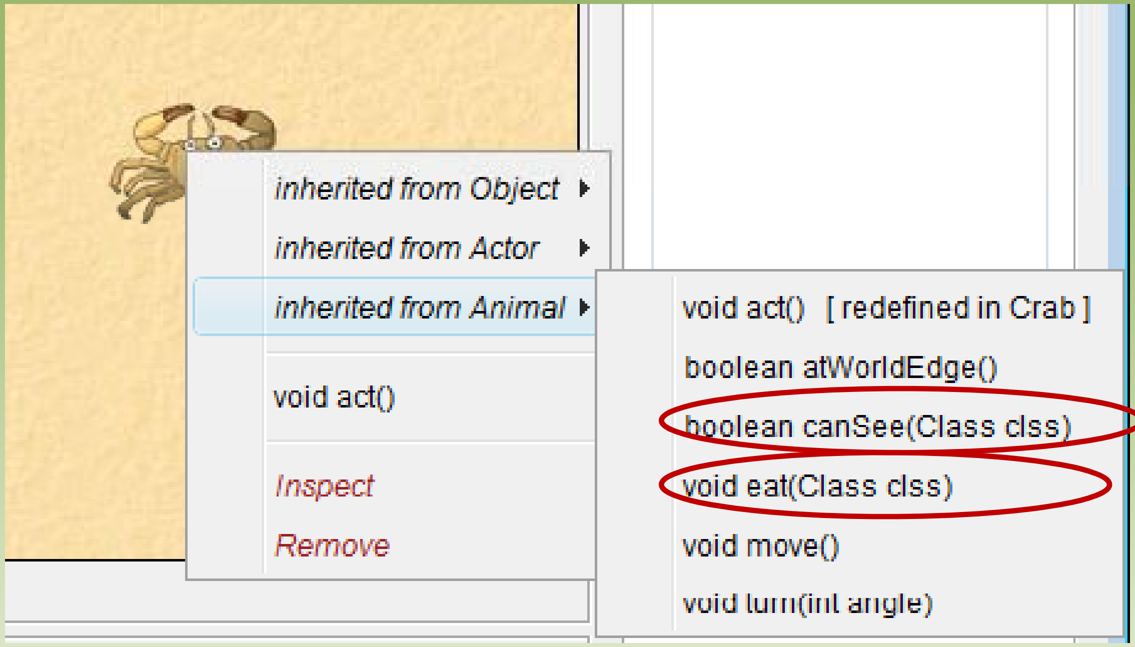
Add worms to your world.

►Run

What happens?

Modify the Crab class to teach the crab to eat worms.

- *If the crab sees a worm* → **Worm.class**
  - *Eat the worm*



Modify the Crab class to teach the crab to eat worms

- *If the crab sees a worm*
  - *Eat the worm*

```
 public void act()
 {
       move();
       if(canSee(Worm.class))
       {
            eat(Worm.class);
       }
       else if( atWorldEdge() )
       {
            turn(15);
       }
}
```

Greenfoot sounds

```
if(canSee(Worm.class))
 {
     eat(Worm.class);
      Greenfoot.playSound("slurp.wav");
 }
```

- You can choose the level of abstraction/complexity to expose to your students by preparing the scenario.
- *Animal* defines '*canSee*' and '*eat*'...
- ...but it does not have to.

Summary: so far…..

- The difference between objects and classes
- Method signature and definition
- Parameters
- Boolean expressions
- `int`, `boolean`, and `void` returns
- `private` vs `public`
- `if; if-else`

More interesting behavior

- Random Numbers
  - In Greenfoot:
    - `int r = Greenfoot.getRandomNumber(100);`
    - *Returns a random int between 0 (inclusive) and 100(exclusive)*

    - `int r = Greenfoot.getRandomNumber(100)- 50;`
    - *Returns a random int between -50 (inclusive) and 50(exclusive)*

A more interesting Crab

```
public class Crab extends Animal
{
    public void act()
    {
        if ( atWorldEdge( ) )
        {
            turn(15);
        }
        move();
        if (canSee(Worm.class))
        {
            eat(Worm.class);
        }

        // Generate a random int between 0 and 100

        // If the number is less than 10, turn a random number of degrees
        // between 0 and 45 (right or left).
    }
}
```

The improved Crab

```
public class Crab extends Animal
{
    public void act()
    {
        turnAtEdge()
        randomTurn();
        move();
        lookForWorm();

   }
}
```

An alternate `CrabWorld`

```
public CrabWorld()
{
   super(560, 560, 1);
   populateWithCrabs();
   populateWithWorms();
}
public void populateWithCrabs()
{
         final int NUM_CRABS =
         //add NUM_CRABS crabs in random locations
}
public void populateWithWorms()
{
         final int NUM_WORMS =
         //add NUM_WORMS worms in random locations
}
```

Game is over when there are no worms!

- Whose responsibility is it to keep track of the worms?
- World Methods?
- How do we "stop" the GREENFOOT Game?

Boring…

- Nothing EXCITING happens!
- Introduce…

- The Lobster

Interactive programs
The Lobster meets the crab

- Crabs eat worms.
- Lobsters eat crabs!
- You get to control the single crab!

The Lobster

- The Lobster behaves like the crab did but eats crabs instead of worms.

```
public void act()
{
    turnAtEdge();
    randomTurn();
    move();
    lookForCrab();
}
```

- The crab's behavior will change.

The interactive Crab

- The crab is controlled by the keyboard.
  - Left arrow turn the crab -4 degrees
  - Right arrow turns the crab 4 degrees

```
public void act()
{
    checkKeypress();
    move();
    lookForWorm();
}
```

Keypress

```
public void checkKeypress()
{
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-4);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(4);
    }
}
```

Animated creatures
Animate the crab!

- The image files used for your animation should live in the images folder for the project.
  - crab.png
  - crab2.png

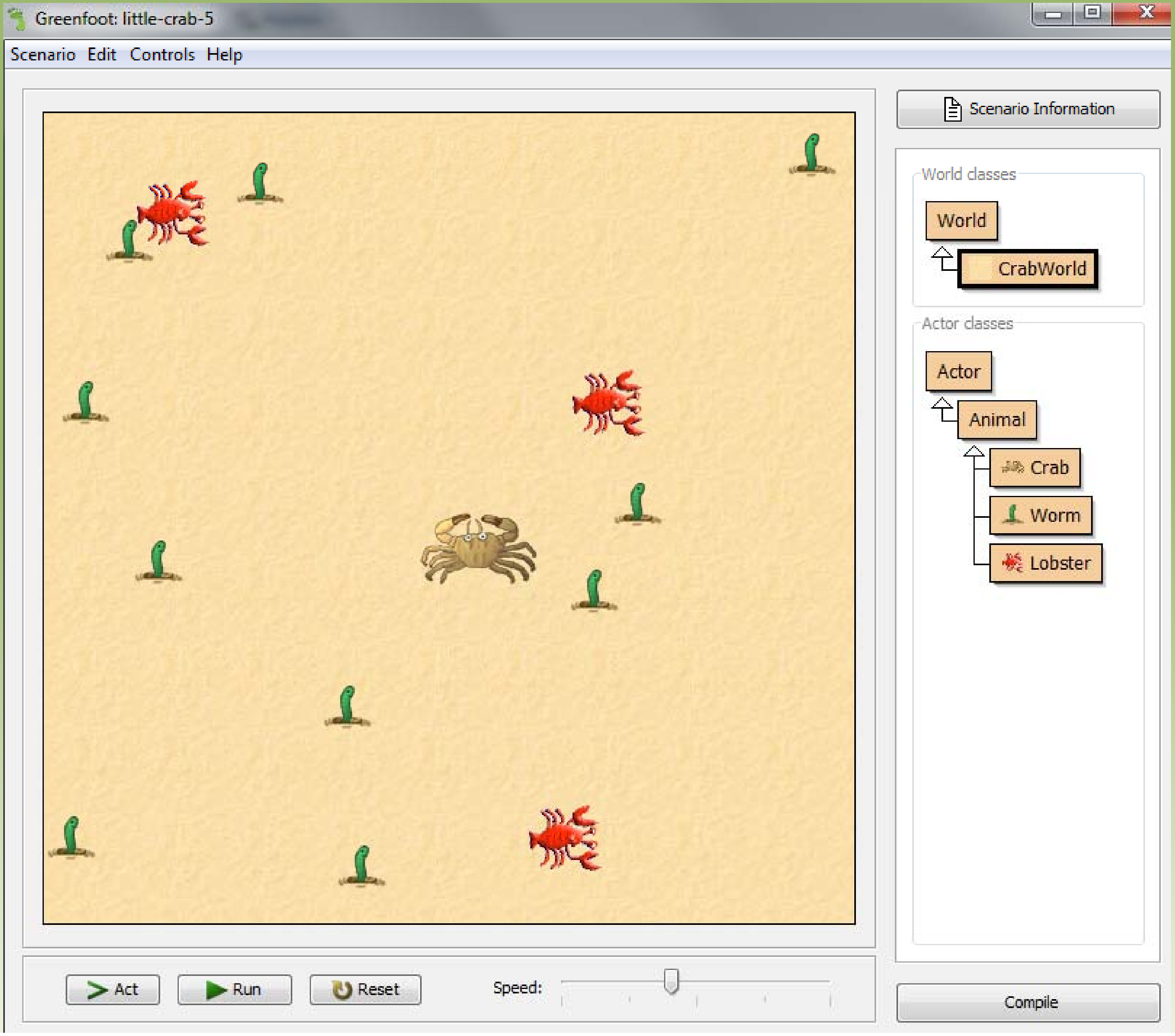
Animate the crab!

```
private GreenfootImage image1;
private GreenfootImage image2;

public Crab()
{
    image1 = new GreenfootImage("crab.png");
    image2 = new GreenfootImage("crab2.png");
    setImage(image1);
}
```

Animate the crab!

```
public void act()
  {
        checkKeypress();
        move();
        lookForWorm();
        switchImage();
  }
```

Our new World



Game is over when there are no worms OR when a lobster eats the crab

- Distinguish between WIN and LOSS

(Some) Advanced Features

Collision Detection & Animations

Key Point (Reiterated)

- You can control the level of abstraction for the students with regards to movement and animation.

Moving Actors

```
 public void move(double distance) {
  double angle = Math.toRadians( getRotation() );

  int x=(int)Math.round(getX()+Math.cos(angle) distance);
  int y=(int)Math.round(getY()+Math.sin(angle) distance);

  setLocation(x, y);
}
```

Motion Using Vectors

- `SmoothMover` class uses a `Vector` to hold direction and “speed” of motion.
- Can create subclasses of `SmoothMover` to use this style of motion.
- Can illustrate the separation of model and view.

Collision Detection

- Even if you don’t have the students control motion/turning/etc. you can still have them write more sophisticated collision detection behaviors using the built in methods in `Actor`.

Collision Detection Methods

java.util.List getIntersectingObjects(java.lang.Class cls)

java.util.List getNeighbours(int distance, boolean diagonal, java.lang.Class cls)

java.util.List getObjectsAtOffset(int dx, int dy, java.lang.Class cls)

java.util.List getObjectsInRange(int radius, java.lang.Class cls)

Actor getOneIntersectingObject(java.lang.Class cls)

Actor getOneObjectAtOffset(int dx, int dy, java.lang.Class cls)

`getOneIntersectingObject(java.lang.Class cls)`

- Pass in the class of the object to look for a particular type of collision
- Pass in **`null`** to look for any intersecting object

- Returns one `Actor` object that matches the criteria.
- **`null`** is returned if no intersections are detected

`getOneObjectAtOffset (int dx, int dy, java.lang.Class cls)`

- Look elsewhere for an object (pass in a dx and dy)
- The Class parameter works the same as previous.

- Returns same as previous: an `Actor` object that is `null` if no intersections are detected.

`getIntersectingObjects(java.lang.Class cls)`

- Returns a list (`java.util.List`) of all intersecting objects of a particular class type `cls`.
  - Passing in `null` returns intersections of all types.

- Can be used to discuss collections, generics, for-each loop.

`getObjectsAtOffset (int dx, int dy, java.lang.Class cls)`

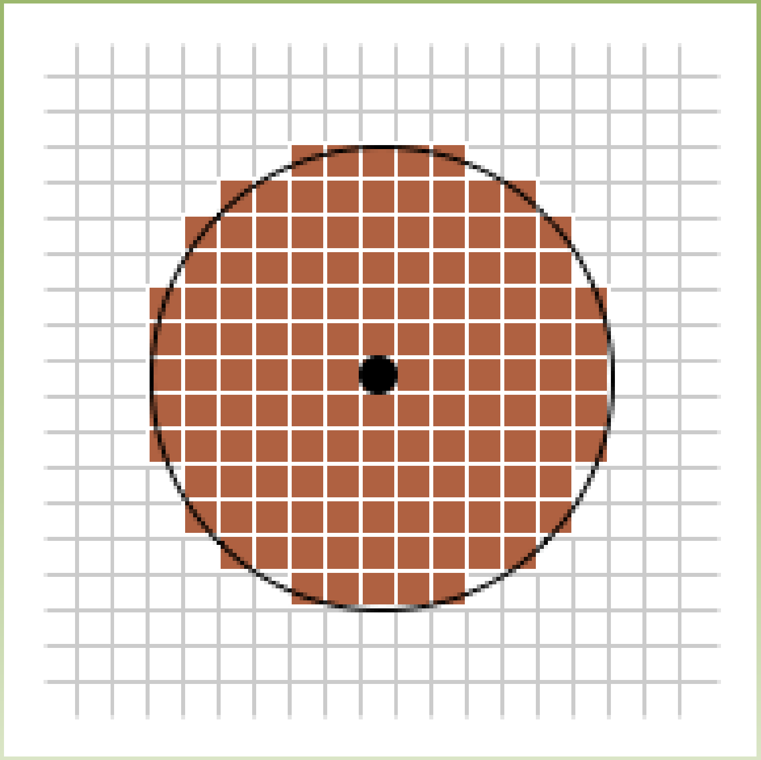
- Analogous to `getObjectAtOffset` except that it returns all of the objects that are intersecting.

getObjectsInRange

```
List getObjectsInRange(int r, java.lang.Class cls)
```

Return all objects within range 'r' around this object.

cell: 10 pixels



r = 6 (cells)

An object is "in range" if its center point is inside the circle.

`getNeighbours`

- Most useful in a world where actors are contained in a cell (like GridWorld case study for AP CS Exam).
- Returns the list of neighbors in the four cardinal directions `distance` cells away from the actor’s current location.
- Passing `true` to the boolean parameter includes the diagonals.

GreenfootImage

- This is the class that represents the images of the actors and the world in the scenarios.
- You can programmatically manipulate these images or draw your own images programmatically.

Drawing Methods

clear()
drawImage(GreenfootImage image, int x, int y)
drawLine(int x1, int y1, int x2, int y2)
drawOval(int x, int y, int width, int height)
drawPolygon(int[] xPoints, int[] yPoints, int nPoints)
drawRect(int x, int y, int width, int height)
drawShape(java.awt.Shape shape)
fill()
fillOval(int x, int y, int width, int height)
fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
fillRect(int x, int y, int width, int height)
fillShape(java.awt.Shape shape)

Outline Shapes

Filled-in Shapes

Sets the current drawing color. Needs to be done before drawing or filling.

Sidenote: Drawing Text

- **drawString**(java.lang.String string, int x, int y)
  - Draw the text given by the specified string, using the current font and color.
- java.awt.Font **getFont**()
  - Get the current font.
- **setColor**(java.awt.Color color)
  - Set the current drawing color.
- **setFont**(java.awt.Font f)
  - Set the current font.

Messages to the screen during a game.

Growing, Shrinking, Transparency

Growing & Shrinking	int getHeight() Return the height of the image.
	int getWidth() Return the width of the image.
	scale(int width, int height) Scales this image to a new size.
Fading Away	int getTransparency() Return the current transparency of the image.
	setTransparency(int t) Set the transparency of the image.
Other animated effects	mirrorHorizontally() Mirrors the image horizontally (flip around the x-axis).
	mirrorVertically() Mirrors the image vertically (flip around the y-axis).
	rotate(int degrees) Rotates this image around the center.

Fun with Images

- Useful if interested in using some of the media computation stuff
- See **getColorAt(x,y)** and **setColorAt(x,y)** methods
  - Grayscale an image
  - Filters on images
  - Create negative
  - Chromakey
- Can be used to reinforce loops

Greenfoot

Simple Assignments
with Dice and Cards

Boolean Expressions: Craps

The game consists of rolling 2 6-sided dice. The shooter makes a "come-out roll" with the intention of establishing a point.

If the shooter's come-out roll is a 2, 3 or 12, it is called "craps" (the shooter is said to "crap out") and the round ends with players losing.

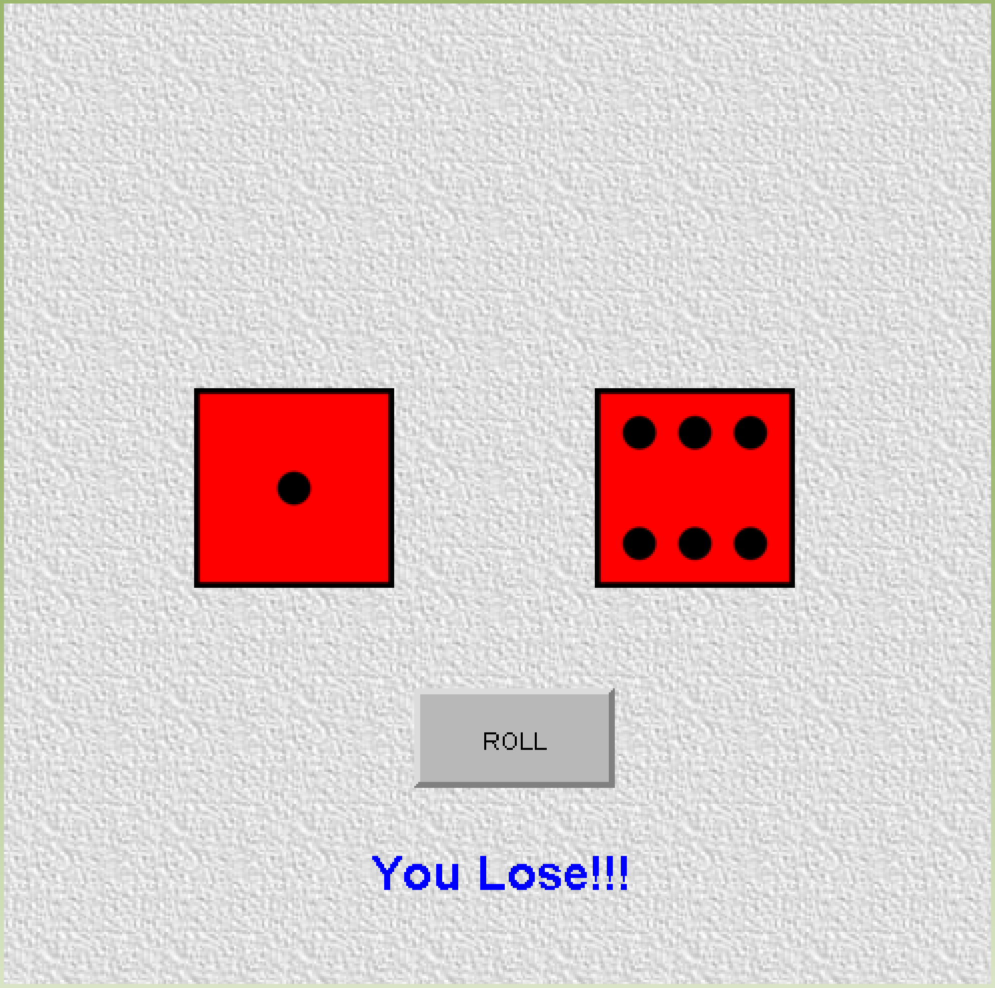
A come-out roll of 7 or 11 is called a "natural," resulting in a win.

If the shooter's come-out roll is a 4, 5, 6, 8, 9, or 10, this number becomes the "point". Once the "point" is established, the shooter will now continue rolling for either the point number or a seven.

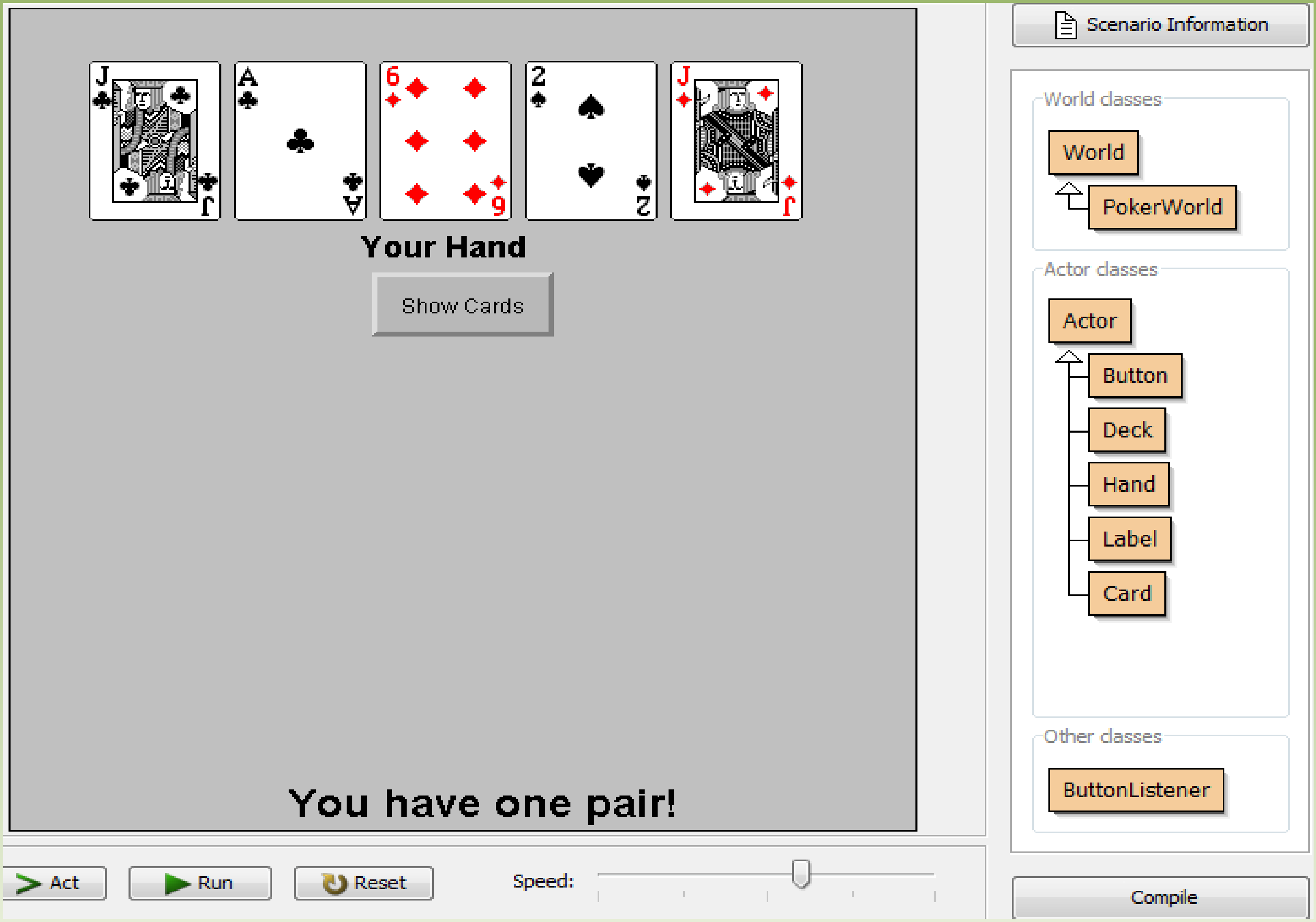
If the shooter is successful in rolling the point number, the result is a win.

If the shooter rolls a seven (called a "seven-out"), the result is a loss.

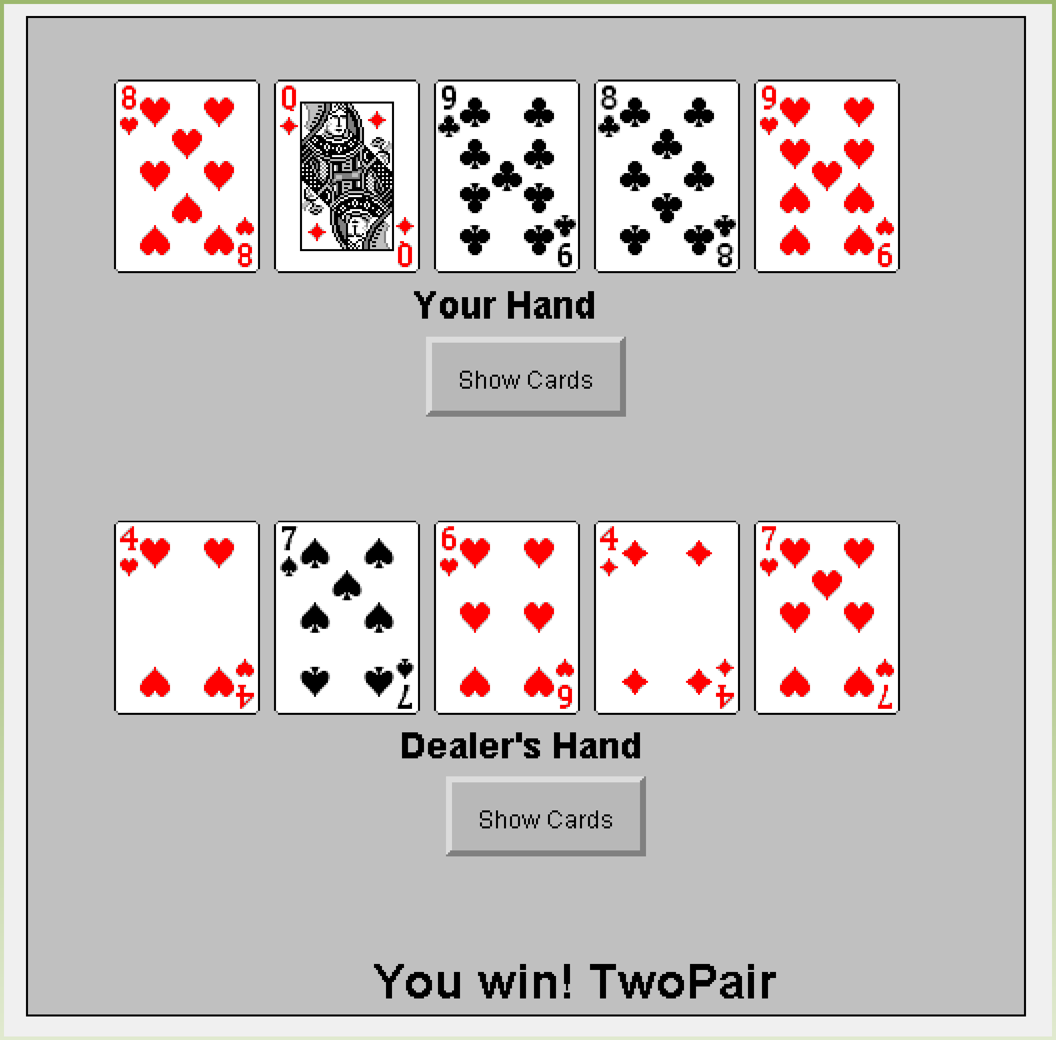
Boolean Expressions: Craps

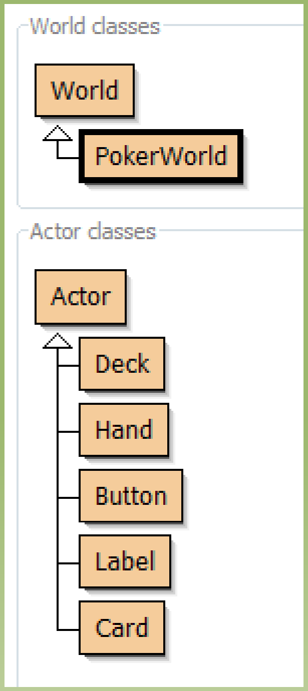


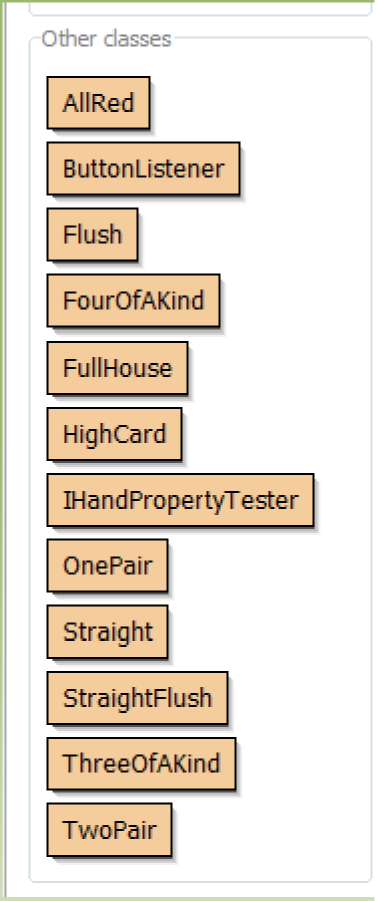
Arrays and ArrayLists: Poker (part 1)



Interfaces: Poker (Part2)



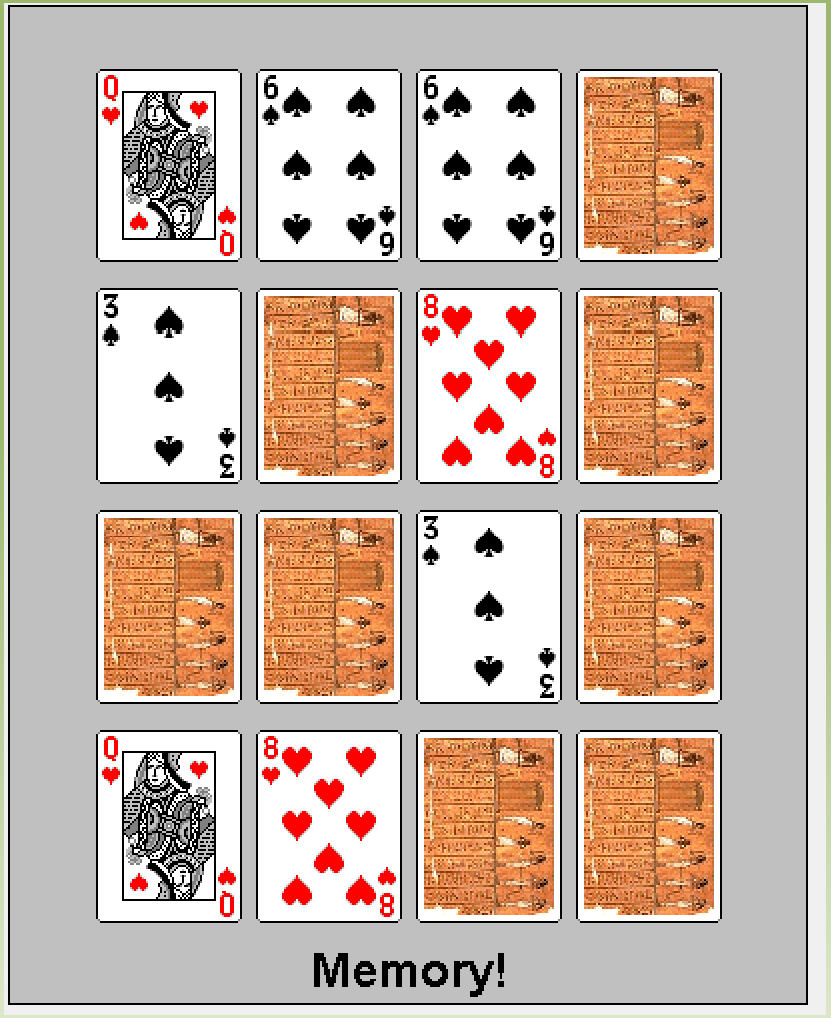




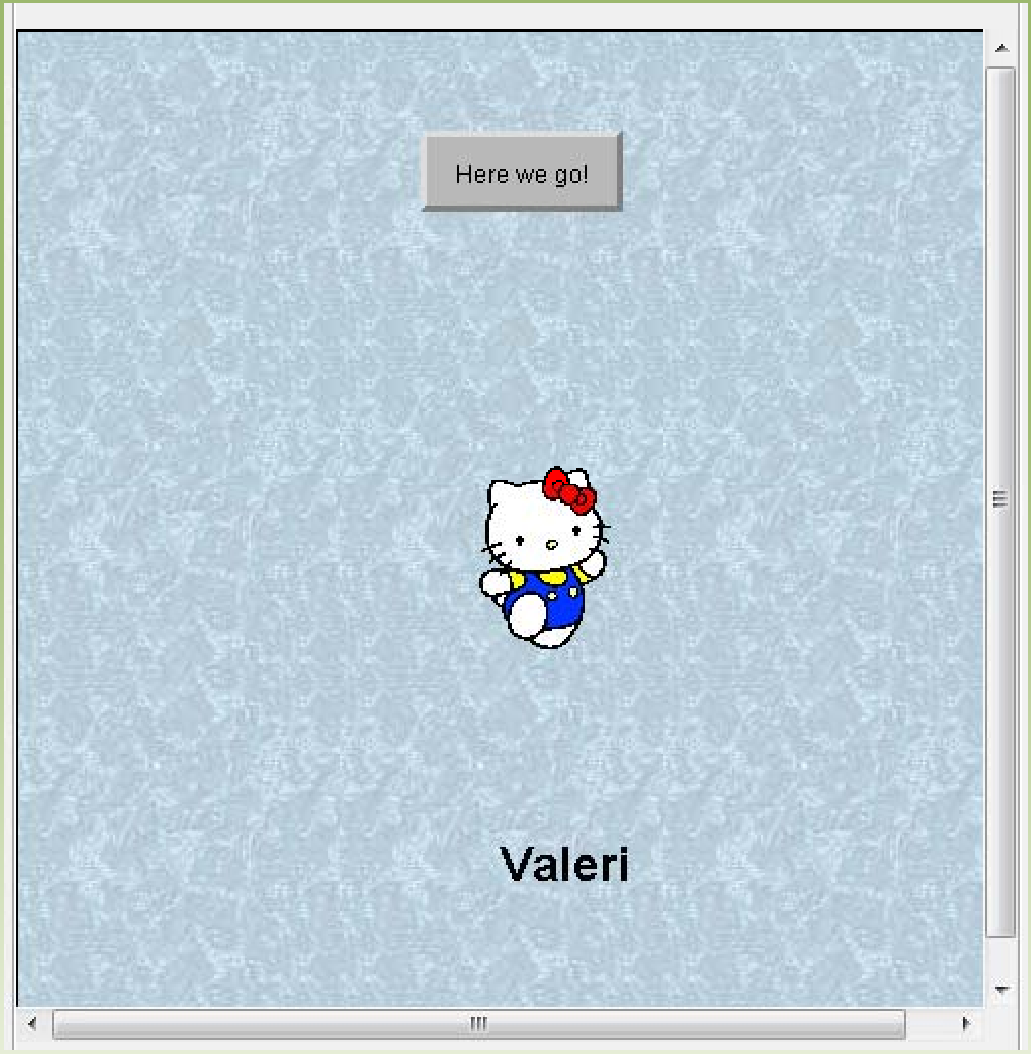
Interfaces: Poker (Part2)

```
public interface IHandPropertyTester
{
   /**
    * Returns true if and only if hand satisfies this property
    * @param hand is the hand tested
    * @return true if hand satisfies property being tested
    */
   boolean hasProperty(Hand hand);

   /**
    * Returns the value of this hand if hand satisfies the property
    * @param hand is the hand tested
    * @return value of the hand tested.
    */
   int getHighValue(Hand hand);
}
```

Arrays and 2-D arrays



Student Lottery
(Who gets this question or prize?)



Beyond Advanced (and onto really, really cool)

Game Pads, Kinect

Game Pads

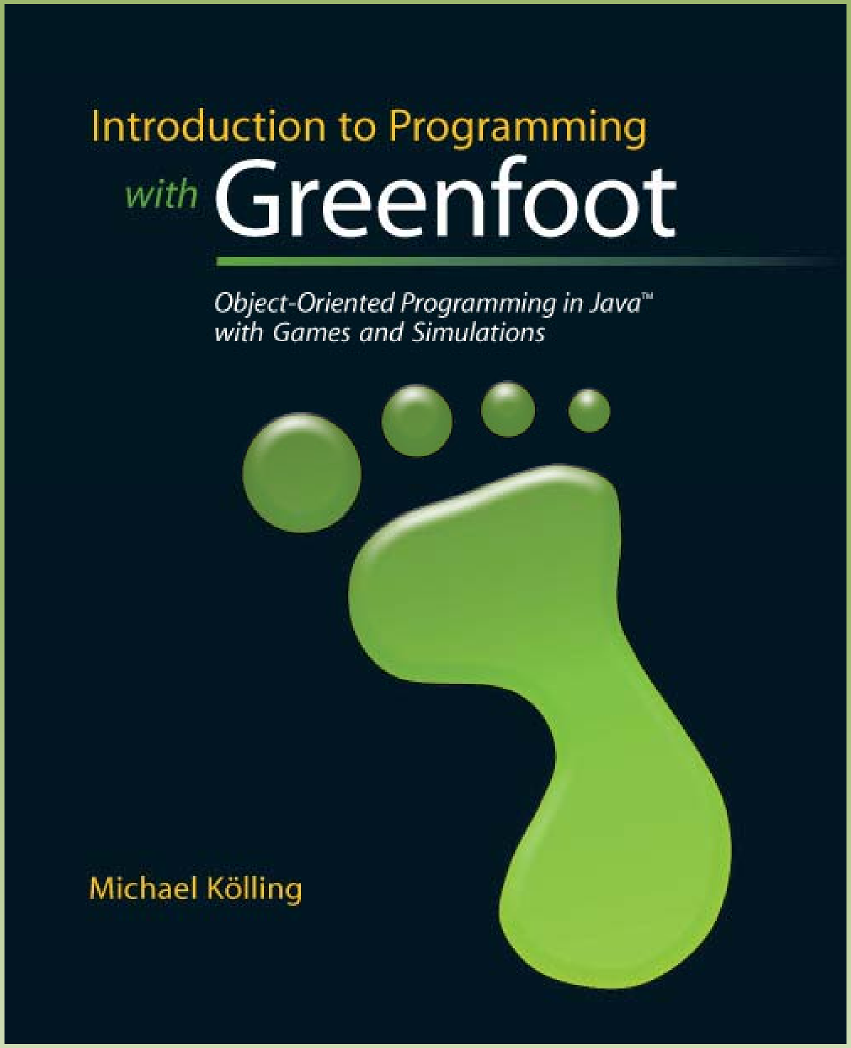
- PS2 and XBox like game pads
- Support classes needed and available (along with documentation and examples)
- http://www.greenfoot.org/doc/gamepad/

Kinect

- http://www.greenfoot.org/doc/kinect/
- Kinect video

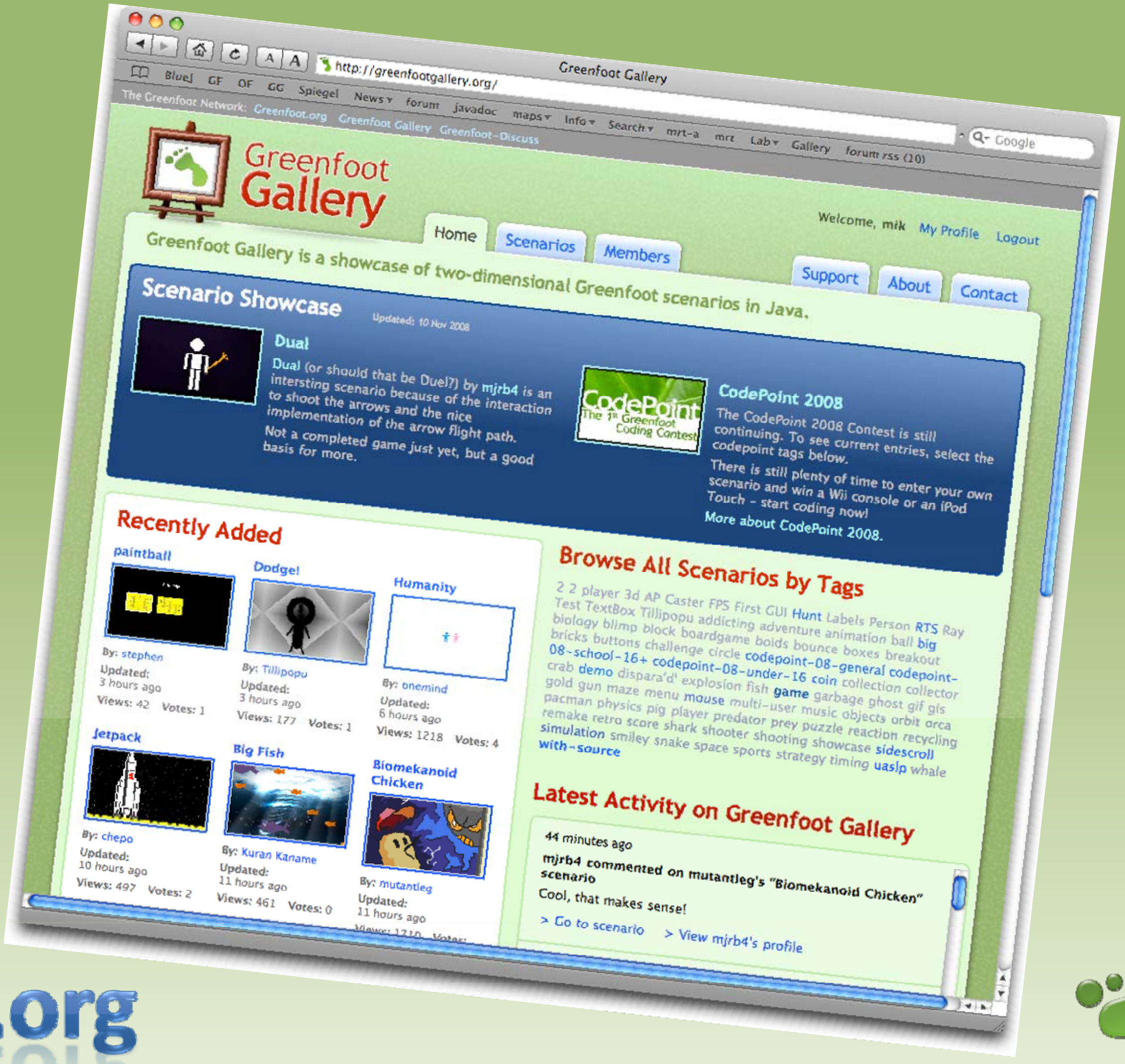
Crabs, and Lobsters, and Bears – oh my!?!

Help, Resources, & Community


Introduction to Programming
with Greenfoot
Object-Oriented Programming in Java™
with Games and Simulations
Michael Kölling

Greenfoot Gallery

- Share!


Greenfoot Gallery
http://greenfootgallery.org/
BlueJ GF OF GG Spiegel News forum javadoc maps Info Search mrt-a mrt Lab Gallery forum rss (10)
Google
The Greenfoot Network: Greenfoot.org Greenfoot Gallery Greenfoot-Discuss
Greenfoot Gallery
Home
Scenarios
Members
Welcome, mik
My Profile
Logout
Support
About
Contact

Greenfoot Gallery is a showcase of two-dimensional Greenfoot scenarios in Java.

Scenario Showcase Updated: 10 Nov 2008

Dual

Dual (or should that be Duel?) by mjrb4 is an intersting scenario because of the interaction to shoot the arrows and the nice implementation of the arrow flight path.

Not a completed game just yet, but a good basis for more.

CodePoint 2008

The CodePoint 2008 Contest is still continuing. To see current entries, select the codepoint tags below.

There is still plenty of time to enter your own scenario and win a Wii console or an iPod Touch - start coding now!

More about CodePoint 2008.

Recently Added

paintball
By: stephen
Updated: 3 hours ago
Views: 42 Votes: 1

Dodge!
By: Tillipopu
Updated: 3 hours ago
Views: 177 Votes: 1

Humanity
By: onemind
Updated: 6 hours ago
Views: 1218 Votes: 4

Jetpack
By: chepo
Updated: 10 hours ago
Views: 497 Votes: 2

Big Fish
By: Kuran Kaname
Updated: 11 hours ago
Views: 461 Votes: 0

Biomekanoid Chicken
By: mutantleg
Updated: 11 hours ago
Views: 1210 Votes:

Browse All Scenarios by Tags

2 2 player 3d AP Caster FPS First GUI Hunt Labels Person RTS Ray Test TextBox Tillipopu addicting adventure animation ball big biology blimp block boardgame boids bounce boxes breakout bricks buttons challenge circle codepoint-08-general codepoint-08-school-16+ codepoint-08-under-16 coin collection collector crab demo dispara'd' explosion fish game garbage ghost gif gis gold gun maze menu mouse multi-user music objects orbit orca pacman physics pig player predator prey puzzle reaction recycling remake retro score shark shooter shooting showcase sidescroll simulation smiley snake space sports strategy timing uaslp whale with-source

Latest Activity on Greenfoot Gallery

44 minutes ago

mjrb4 commented on mutantleg's "Biomekanoid Chicken" scenario

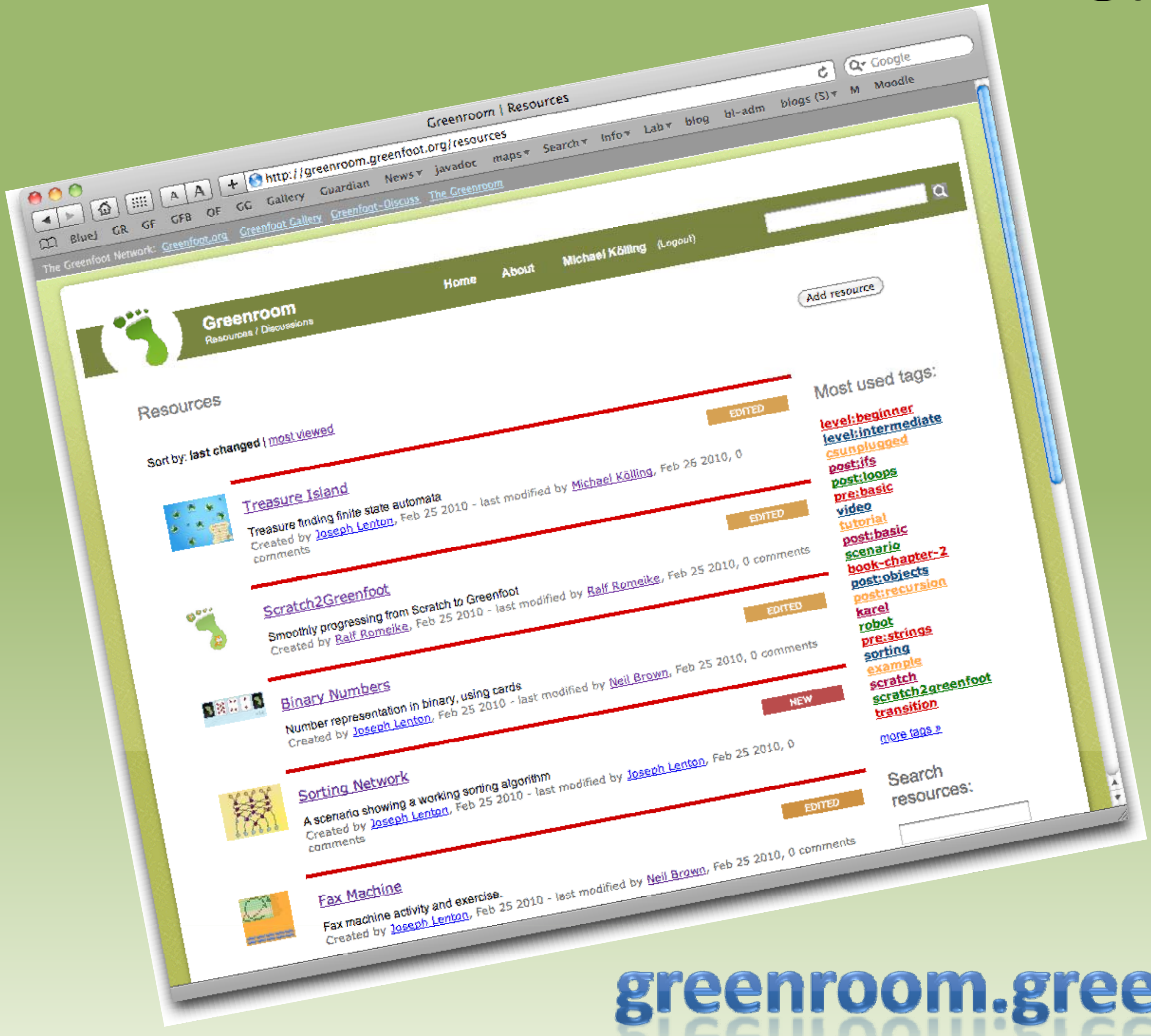
Cool, that makes sense!

> Go to scenario > View mjrb4's profile

greenfootgallery.org

Greenroom

- Meet!


Greenroom | Resources
http://greenroom.greenfoot.org/resources
Google
BlueJ GR GF GFB OF GG Gallery Guardian News javadoc maps Search Info Lab blog bl-adm blogs (5) M Moodle
The Greenfoot Network: Greenfoot.org Greenfoot Gallery Greenfoot-Discuss The Greenroom
Greenroom
Resources / Discussions
Home About Michael Kölling (Logout)
Add resource

Resources

Sort by: last changed | most viewed

EDITED
Treasure Island
Treasure finding finite state automata
Created by Joseph Lenton, Feb 25 2010 - last modified by Michael Kölling, Feb 26 2010, 0 comments

EDITED
Scratch2Greenfoot
Smoothly progressing from Scratch to Greenfoot
Created by Ralf Romeike, Feb 25 2010 - last modified by Ralf Romeike, Feb 25 2010, 0 comments

EDITED
Binary Numbers
Number representation in binary, using cards
Created by Joseph Lenton, Feb 25 2010 - last modified by Neil Brown, Feb 25 2010, 0 comments

NEW
Sorting Network
A scenario showing a working sorting algorithm
Created by Joseph Lenton, Feb 25 2010 - last modified by Joseph Lenton, Feb 25 2010, 0 comments

EDITED
Fax Machine
Fax machine activity and exercise.
Created by Joseph Lenton, Feb 25 2010 - last modified by Neil Brown, Feb 25 2010, 0 comments

Most used tags:

level:beginner
level:intermediate
csunplugged
post:ifs
post:loops
pre:basic
video
tutorial
post:basic
scenario
book-chapter-2
post:objects
post:recursion
karel
robot
pre:strings
sorting
example
scratch
scratch2greenfoot
transition

more tags »

Search resources:

greenroom.greenfoot.org

More information

Greenfoot – The Java Object World

http://www.greenfoot.org/

greenfoot

HOME ABOUT GREENFOOT GETTING STARTED DOCUMENTATION DOWNLOAD SCENARIOS

Now available: Greenfoot 1.1

Greenfoot 1.1 is available for download now.

Greatly improved support for games with newly added sound support. View a list of new features and improvements here.

Also available: Asteroids, a new scenario. Watch a short video here or download and run the scenario yourself.

Make music with Greenfoot! A new scenario, named Piano is available for download.

You have questions? We have answers! Join the mailing list

Look at screenshots

The Greenfoot Tutorial — If you are new to Greenfoot, start here.

On Walkabout with Greenfoot — Meanderings around Greenfoot – a blog.

maintained by Poul Henriksen and Michael Kölling

- www.greenfoot.org
- discussion group
- scenario repository
- tutorials (text and video)
- Greenfoot Gallery
- Greenroom
- Adrienne: newyork-hub@greenfoot.org
- Fran: newjersey-hub@greenfoot.org

Questions?

Discussion Time