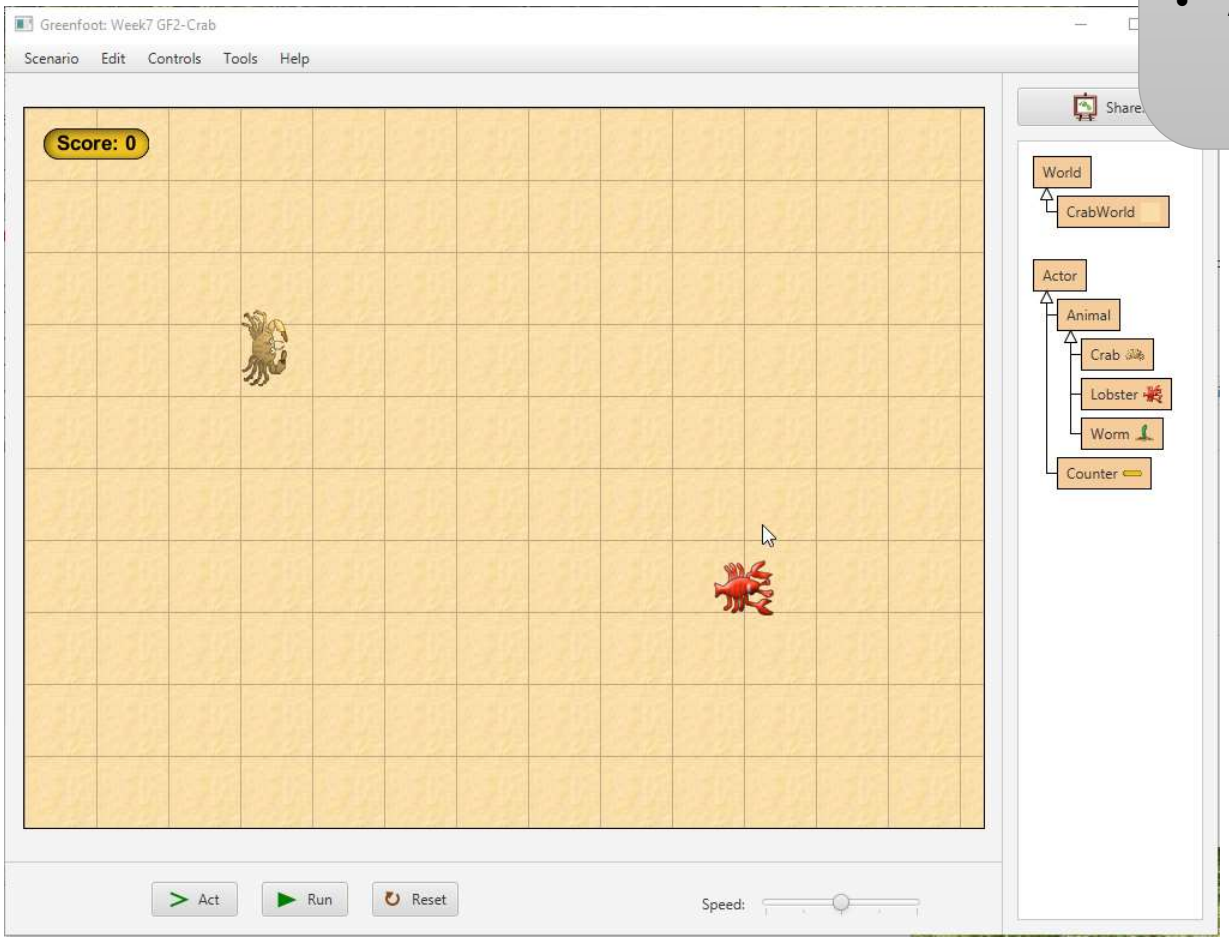
Greenfoot Games

By Derek Peacock

Animal class

- Crab, Lobster, Worm are kinds of Animal
- Animal and Counter are kinds of Actor
- All animals are therefore also Actors



What Can Animals Do: move()

```java
public class Animal extends Actor
{
    private static final int WALKING_SPEED = 5;

    /**
     * Move forward in the current direction.
     */
    public void move()
    {
        while(!atWorldEdge())
            move(WALKING_SPEED);
    }


    /**
     * Test if we are close to one of the edges of the world.
     * Return true if we are.
     */
    public boolean atWorldEdge()
    {
        if(getX() < 20 || getX() > getWorld().getWidth() - 20)
            return true;
        if(getY() < 20 || getY() > getWorld().getHeight() - 20)
            return true;
```

- Simple move from left to right
- Stops at the edge of the world
- Add a new Animal, right click on move()

What Can Animals Do: eat()

```java
/**
 * Return true if we can see an object of class 'anyClass'
 * within the set distance. False if there is no such object here.
 */
public boolean canSee(Class anyClass, int distance)
{
    Actor actor = getOneObjectAtOffset(distance, distance, anyClass);
    return actor != null;
}
```

```java
/**
 * Try to eat an object of class 'clss'. This is only successful if there
 * is such an object where we currently are. Otherwise this method does
 * nothing.
 */
public void eat(Class foodClass)
{
    if(isTouching(foodClass))
    {
        removeTouching(foodClass);
    }
}
```

- Notice you cannot use the word **class** as class is a reserved word as is **Class**

The Animal class is not much use directly so subclasses are created.

What Can Crabs Do: constructor()

```java
public class Crab extends Animal
{
    protected int width;
    protected int height;

    protected int speed = 3;
    protected int turnAngle = 4;

    protected GreenfootImage image;

    private CrabWorld world;

    public Crab()
    {
        image = getImage();

        width = image.getWidth();
        height = image.getHeight();

        image.scale((int)(width * 0.8), (int)(height * 0.8));

        setRotation(90);
    }
```

The Crab image is reduced in size and the Crab is turned so that it faces to the right

What Can Crabs Do: act()

```
/**
 * Act - do whatever the MovingSprite wants to do.
 * This method is called repeatedly whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move4Ways();

    if(isTouching(Worm.class))
    {
        eat(Worm.class);
        Greenfoot.playSound("slurp.wav");

        world = (CrabWorld)getWorld();
        world.score();
    }
}
```

- Every time the crab moves a check is made to see if it is touching a worm.
- If it is then the worm is eaten, a sound is played and the score kept in the world is updated.
- The sound file is stored in a sounds folder

Greenfoot API

https://www.greenfoot.org/files/javadoc/

Actor
Color
Font
Greenfoot
GreenfootImage
GreenfootSound
MouseInfo
UserInfo
World

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

greenfoot

Class Actor

java.lang.Object
greenfoot.Actor

```
public abstract class Actor
extends java.lang.Object
```

An Actor is an object that exists in the Greenfoot world. Every Actor has a location in the world

An Actor is not normally instantiated, but instead used as a superclass to more specific objects behaviour.

One of the most important aspects of this class is the 'act' method. This method is called when implementations.

Version:

2.5

Author:

Poul Henriksen

What Do Crabs Do: move4Ways()

```java
/**
 * This method moves the paddle around in four directions
 * left, right, up and down using arrow keys.
 */
public void move4Ways()
{
    int x = getX(); int y = getY();
    int halfWidth = width / 2;

    if(Greenfoot.isKeyDown("left") && x > halfWidth)
    {
        setRotation(270);
        x -= speed;
    }

    if(Greenfoot.isKeyDown("right") && !isAtEdge())
    {
        setRotation(90);
        x += speed;
    }

    if(Greenfoot.isKeyDown("down") && !isAtEdge())
    {
        setRotation(180);
```

```java
    if(Greenfoot.isKeyDown("down") && !isAtEdge())
    {
        setRotation(180);
        y += speed;
    }

    if(Greenfoot.isKeyDown("up") && y > speed)
    {
        setRotation(0);
        y -= speed;
    }

    setLocation(x, y);
}
```

What Do Crabs Do: turnAndMove()

```java
/**
 * This method rotates the worm a small amount to the
 * left or to the right, and then the worm moves in that
 * direction
 */
public void turnAndMove()
{
    if(Greenfoot.isKeyDown("left"))
    {
        turn(-turnAngle);
    }

    if(Greenfoot.isKeyDown("right"))
    {
        turn(turnAngle);
    }

    if(Greenfoot.isKeyDown("space"))
    {
         move(speed);
    }
}
```

This kind of movement first came out in Turtle Graphics as a simpler way of controlling moving objects

The Crab image must face right, please rotate it in an image editor or Windows 11.

Setting Up CrabWorld

```java
import greenfoot.*;  // (World, Actor, GreenfootI
import java.util.Random;

/**
 */
public class CrabWorld extends World
{
    public static final int MAXN_WORMS = 20;

    private Crab crab;
    private Lobster lobster;

    private Worm[] worms;
    private int wormSize = 30;
    private int remainingWorms = MAXN_WORMS;

    private Random generator = new Random();
    private Counter score;
```

We need **Random** numbers in order to create a different world each time

We need lots of worms so an array or an ArrayList can be used.

This class controls the game and decides when it is won or lost.

CrabWorld Constructor()

```
public CrabWorld()
{
    // Create a new world with 600x400
    super(800, 600, 1);

    crab = new Crab();
    addObject(crab, 200, 200);

    lobster = new Lobster();
    addObject(lobster, 600, 400);

    worms = new Worm[MAXN_WORMS];

    wormSize = 30;
    addWorms();

    setupScore();
}
```

The Crab and Lobster start at the same position each game. That should change.

CrabWorld createWorms()

```java
/**
 * Add MAXN_WORMS to the world in random positions
 */
public void addWorms()
{
    createWorm();
}

/**
 * This method creates & adds a single worm to a random position in
 * the world that is not too near the edge of the screen
 */
private void createWorm()
{
    Worm worm = new Worm();

    int x = generator.nextInt(getWidth());
    int y = generator.nextInt(getHeight());

     addObject(worm, x, y);
}
```

This just creates 1 worm at a random position

Game Enhancements

- Add more worms
- Increase the score when a worm is eaten
- Display Game Won when all the worms are eaten
- Get the Lobster moving around trying to eat the Crab
- If the Crab is eaten display lose game.
- Try adding solid objects like rocks
- Try making the game more difficult each time it is won.