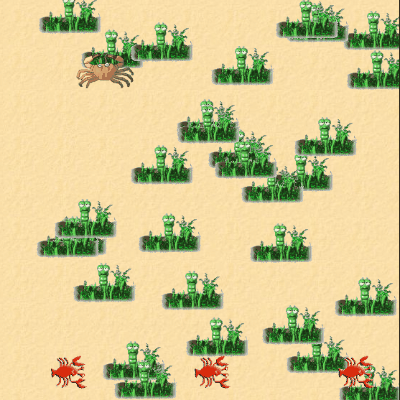
**Little Crab**

****

**Part 1**

1. Create a new folder in your Greenfoot folder and name it **Little Crab**
2. [Download the crab scenario](https://drive.google.com/file/d/1FKYw3GTxuj0G3eVNluhFku1-12b_merT/view?usp=share_link) and unzip it and delete the zip folder.
3. Rename the project as **YourName-crab**
4. Open Greenfoot, then open the crab scenario that you just downloaded.
5. Place a crab into the world and run the program (click the Run button).

*What did you observe?*

1. Open the source code for the Crab.

*Notice the code (or the lack thereof) inside the act method. This is why the crab didn’t do anything.*

1. Look at the classes along the right side.

*Notice that a Crab is an Animal which is an Actor. In Java this is called inheritance. All the public methods in the Animal class and the Actor class are inherited by the Crab. Thus, if an animal can do it then a Crab can do it.*

1. Double-click on the Animal class to view its source code.
2. Using the drop-down menu located in the top corner, change it from source code to documentation. Look over the code for the methods in the Animal class. Change it back to source code and close the source code.
3. Open the source code of the Crab class.
4. Inside the body of the act() method type:

**move();**

*This calls the move() method - or in other words, the code inside the move() method is executed - thus, when act() is called move() is called. Where is the move() method? It certainly is not inside the Crab class - the only thing there is act(). The move() method is inherited from the Animal class.*

1. Place a new crab on the world and click on the Act button two or three times.
2. Now click on the Run button.

*The Act button calls the act() method for all the classes one time, whereas the run button is inside a forever loop that repeatedly calls the act() method forever. This is built into Greenfoot.*

1. Click on the Reset button and then add two or three crabs to the world then click the Run button.

*You will see that the crab can now move across the screen. The move() instruction makes the*

*crab move a little bit to the right. When we click the Act button in the Greenfoot main window,*

*the act method is executed once. That is, the instruction that we have written inside the act*

*method (move()) executes. Clicking the Run button is just like clicking the Act button several times, very quickly. So the act method is executed over and over again, until we click Pause.*

1. Open the Crab source code for editing.
2. Replace move(); with turn(5);

*The 5 is input into this method. In Java this is called a parameter. turn(5) calls the turn method and passes it a value of 5. Where would you find the code that will be executed when this method is called? It is in the Animal class.*

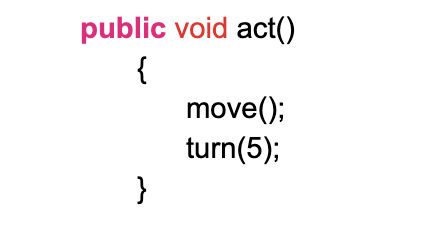
1. Reset the world and add a couple of crabs to the world.
2. Click on the Act button two or three times then click on the Run button.
3. Open the Crab source code.
4. Change the parameter 5 to 90 and compile the class.
5. Reset the project then add a crab and click on the Act button 4 times.

*The crab turns 90° each time it acts. If it acts four times it turns around once.*

1. Double-click on the Animal class and if it is set to documentation change it to source code (use the drop-down menu located in the upper right corner).
2. Find the turn method.

*Notice that there is an integer (int) input into this method named integer and the method gets the current rotation of the animal and adds the angle passed into it.*

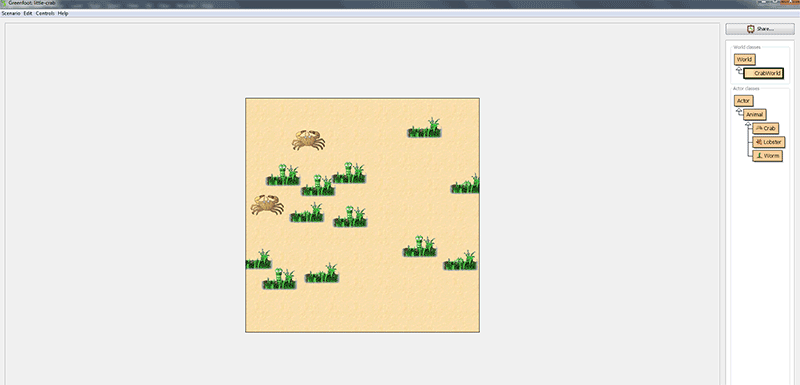
1. Close the source code.
2. Edit the source code for the Crab class so that the crab turns **left** 90° each time it acts.
3. Compile the project, add a crab, and try it out.
4. Edit the source code for the Crab class so that it both moves and turns (change the parameter to 5 as shown below):



1. Compile, put a crab in the world and run the project.
2. Open the source code for the Crab class and remove the semicolon after move();
3. Compile the class.

*Notice that you get a compiler error at the spot where the semicolon used to be. Move your cursor over the compiler error and it will tell you what the problem is.*

1. Put the semicolon back and re-compile the class.

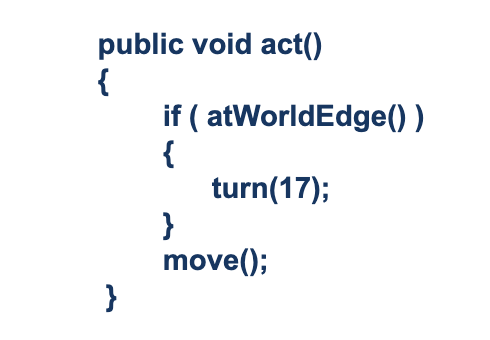


1. Double-click on the Animal class to view the documentation (you may need to change it from source code to documentation).
2. Look over the methods. Which two methods might be useful in getting the crab to turn around once it reaches the world’s edge?

*Notice that before each of the methods it lists the return type (void or boolean). This is the data type of what this method reports. Void means that it reports nothing. This would be like a command block in BYOB. Boolean means that it reports either true or false. This would be like a predicate block in BYOB. What does atWorldEdge() report?*

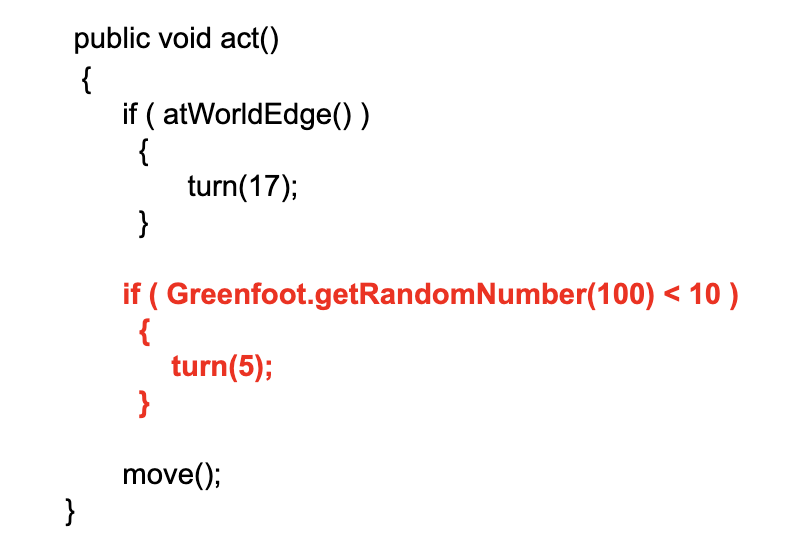
*It is either true or false that the crab is at the world’s edge, therefore atWorldEdge()’s return type is a boolean.*

1. Close the documentation.
2. Reset the world and add a crab to the world.
3. Right-click on the crab and select *inherited from Animal → atWorldEdge()*. What does it report? Can you get it to report true?
4. Open the source code for the Crab class.
5. Change the act() method of the Crab class to the following:

****

1. Add two or three crabs to the world and run the program.
2. Experiment with different values for the parameter to the turn method. Find one that looks good.
3. Open the source code of the Crab class for editing.
4. Change the act() method as follows (add the code in red):

public void act()



1. Edit → Auto-layout.
2. Now try out the program with these changes.

*If the crab turns, it always turns the same amount (5 degrees), and secondly, it always turns right, never left. What we would really like to see is that the crab turns a small, but random amount to either its left or its right.*

1. Find the turn(5) in the act() method of the Crab class then change the 5 to a random number between 0 and 44. The part you need to change is shown in red:



1. Compile the class then close the source code then add a crab to the world and run the project.

*The crab doesn’t always turn the same amount but it still only turns right.*

1. Open the source code for the Crab class and change the amount the crab turns to this:

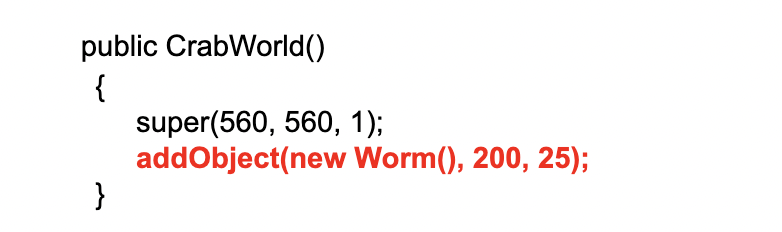
turn(Greenfoot.getRandomNumber(**91**) **- 45**);



1. Right-click on the Animal class and choose New subclass.
2. Name this class **Worm** and select the worm image.
3. Add some worms to your world. Also add some crabs. Run the scenario.

*What do you observe? What do the worms do? What happens when a crab meets a worm?*

1. Reset your world.
2. Open the source code for the CrabWorld class.
3. Add code to the constructor so that it will add a worm at location (200, 25). Your code will look like this (the part you are adding is red):



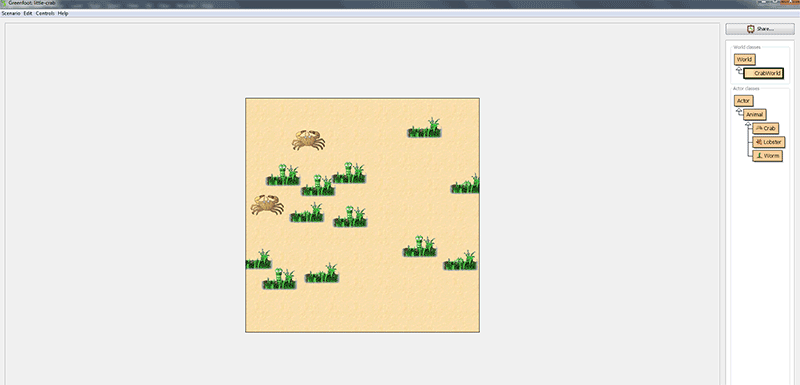
1. Compile and run the program. Reset it.

*Notice that the worm is always placed there. It is created when the world is created and put at location (200,25).*

1. Now change the constructor for the CrabWorld class so that it adds 100 worms at random locations. Replace the line of code that adds one Worm to the world with the code shown in red:

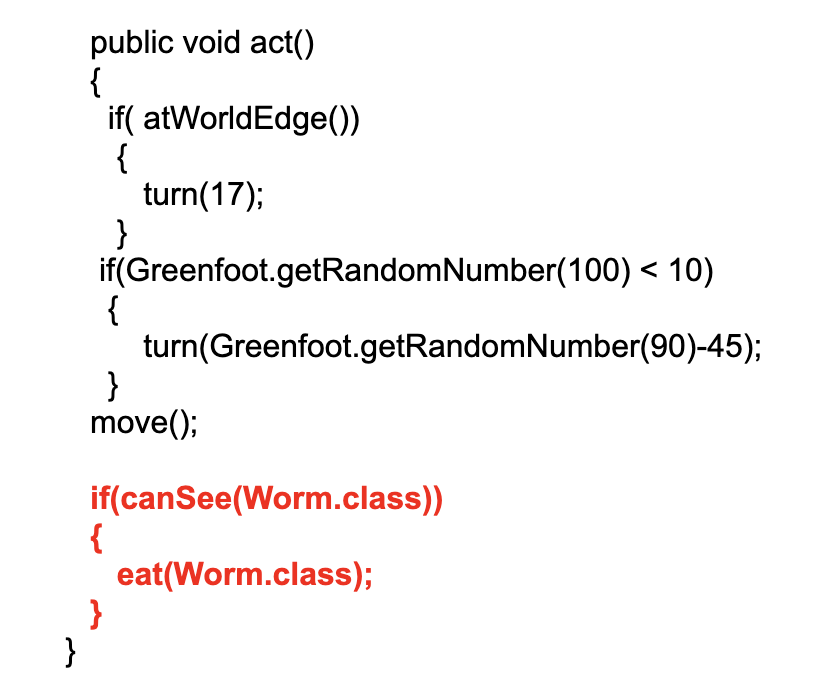


1. Edit → Auto-layout.
2. Reset it several times and notice that the worms are always in a different location.

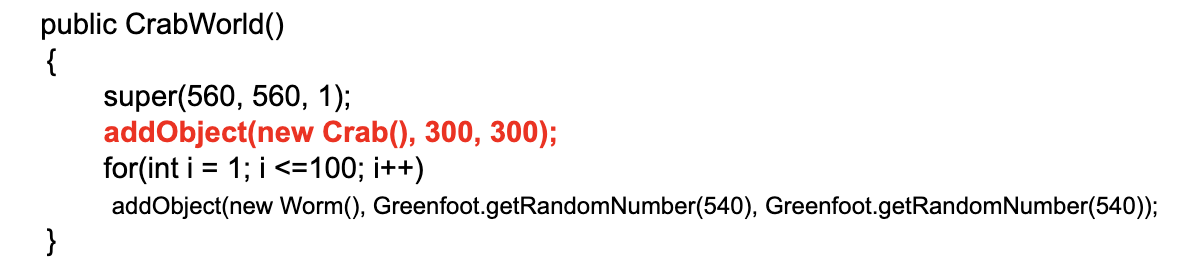


*We now want to add new behavior to the crab: When the crab runs into a worm, it eats it.*

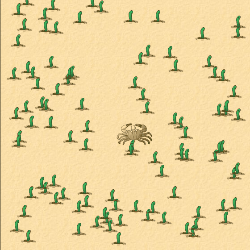
1. Open the documentation for the Animal class. What two methods would be useful for finding a worm and eating it? What input do these methods need? What do they return? Close the documentation.
2. Add code to the act() method of the Crab class so that if it can see a worm then it eats the worm. Just add the code shown in red:



1. Add code to the constructor for the CrabWorld class so that it automatically adds a Crab to the world:



1. Run the program. Worms should disappear as the Crab moves through the world.



1. Create a new subclass of Animal named Lobster and use the image lobster.png as the default image for this class.
2. Delete the act() method in the Lobster class - the whole thing including the header and the curly braces.
3. Copy the complete act() method from the Crab class and paste it into the Lobster class. The code you need to copy is shown highlighted in blue below.



1. Change the Lobster code so that it looks for crabs, rather than worms. You can do that by changing both occurrences of “Worm” in the source code to “Crab”. For instance, where Worm.class is mentioned, change it to Crab.class.
2. Place a Lobster into the world. Does the crab manage to eat all the worms before it is caught by a lobster?
3. Delete the code that causes the Crab to do a random turn from the act() method of the Crab class and replace it with code that makes the crab turn left whenever the left arrow key is pressed. The code you are deleting is shown in blue and the code that you are replacing it with is shown in red:

**if(Greenfoot.getRandomNumber(100) < 10)**

**{**

**turn(Greenfoot.getRandomNumber(90)-45);**

**}**

*Use Greenfoot’s auto-complete feature to help you do this. Type Greenfoot. then do Control-Space then double-click on isKeyDown then add “left” as the parameter.*

**if (Greenfoot.isKeyDown("left"))**

**{**

**turn(-4);**

**}**

1. Add code that makes the crab turn right when the right arrow key is pressed. Just copy paste the left turning code and change “left” to “right” and change the -4 to 4.
2. Run the project and use the arrow key to control the Crab.
3. Open the source code for the Lobster class and find this line of code:

eat(Crab.class);

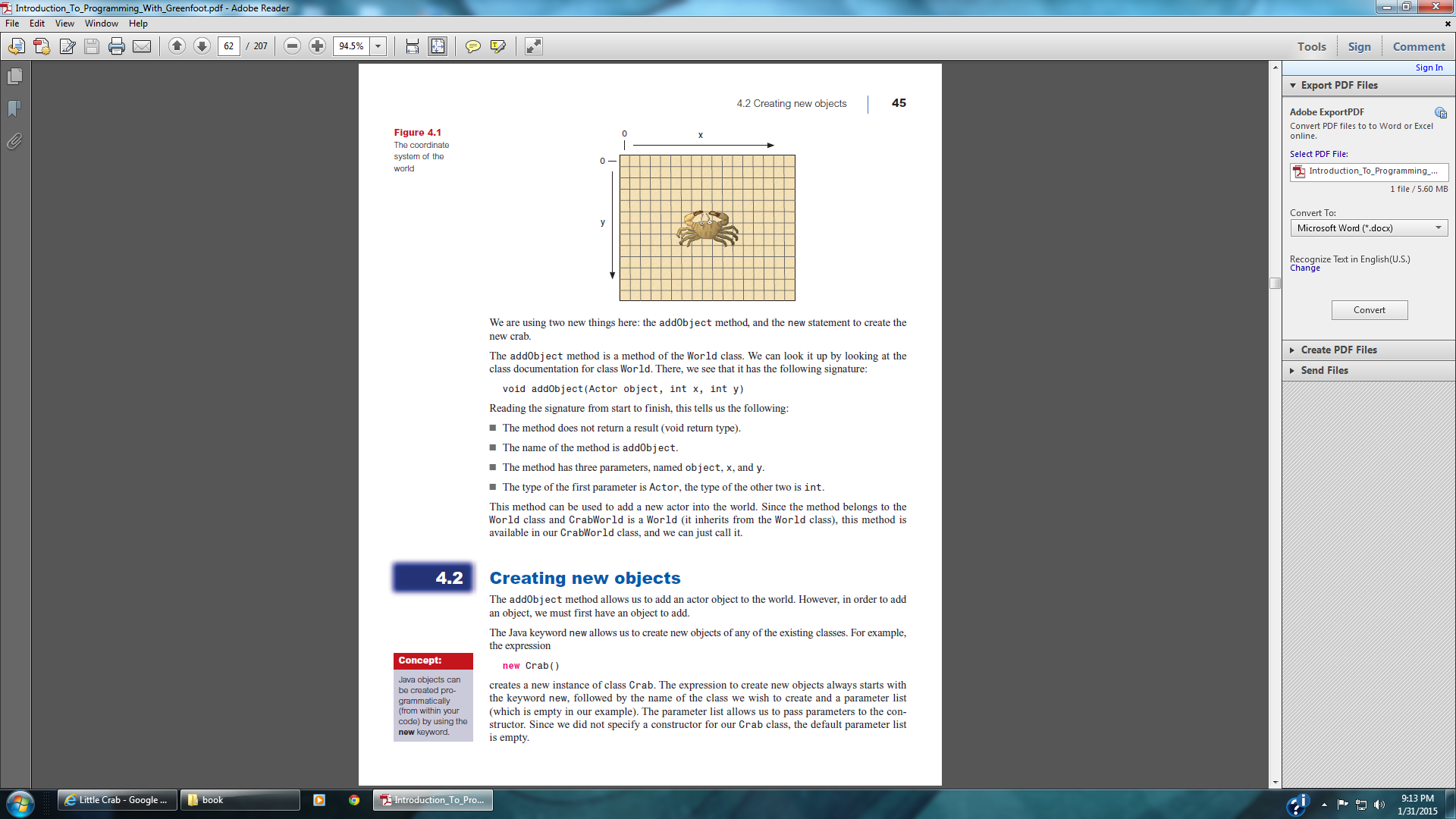
1. Underneath this line of code type **Greenfoot.** then Control-Space then find the method that causes the program to stop and double-click on this method. My guess is you can figure out which method stops the program by looking over the names of all the choices.

*We want it to play a slurping noise everytime the Crab eats a worm. Any sound files used in a Greenfoot project must be stored in the folder named sounds that is inside the scenario folder. There are already sound files in this folder including one name “slurp.wav” that plays a slurping sound.*

1. Open the Crab class and find the line of code where the crab eats a worm. Underneath this line of code type **Greenfoot.** then do Control-Space. Double-click on the method that causes a sound to play then change the input to “slurp.wav” (with quotes).
2. When a Lobster eats the Crab, add code that will play the “au.wav” sound. You will need to add it after the line of code that causes a Lobster to eat the Crab and before the line of code that stops the program.
3. Open the source code for the CrabWorld class.
4. Add the following line to the constructor beneath the first line of code:

**addObject( new Crab(), 150, 100 );**

*This means to add a new crab at location x = 150 and y = 100. The crab will be automatically added there when the world is created.*



1. Add code to automatically create three **lobsters** in the CrabWorld. You can choose the locations for them in the world.

*Animation is achieved with a simple trick: We have two different images of the crab in our scenario (they are called crab.png and crab2.png), and we simply switch the crab’s image between these two versions fairly quickly. The position of the crab’s legs in these images is slightly different.*

crab.png crab2.png

1. Open the source code for the Crab class.
2. Add these two lines to the top of the class and make sure they are not inside of any method:

**private GreenfootImage image1;**

**private GreenfootImage image2;**

1. Add a default constructor to the Crab class:

**public Crab()**

**{**

**}**

*Constructor has the same name as the class.*

1. Add the following code to the body of the constructor:

**image1 = new GreenfootImage("crab.png");**

**image2 = new GreenfootImage("crab2.png");**

1. Add this line as the last line of the Crab constructor:

**setImage(image1);**

*When the Crab is created, it will use image1 to represent what it looks like. You can always add code to change the image as the program executes.*

1. Add the following code to the act() method of the Crab class underneath all the code that is already there. Look over the code and see if you can figure out what it does:

**if(getImage() == image1)**

**{**

**setImage(image2);**

**}**

**else**

**{**

**setImage(image1);**

**}**

1. Open the Crab class and add the following variable. Put it underneath the two that are already there:

**private static int numWormsDevoured;**

1. In the constructor, initialize this variable to 0. The code you need to add is shown in red:

public Crab()

{

image1 = new GreenfootImage("crab.png");

image2 = new GreenfootImage("crab2.png");

setImage(image1);

**numWormsDevoured = 0;**

}

1. Place code just after a Crab eats a Worm so that the count is increased by 1 and if the count is 8 or more it plays a victory sound and stops the program:

**numWormsDevoured++;**

**if( numWormsDevoured >= 8)**

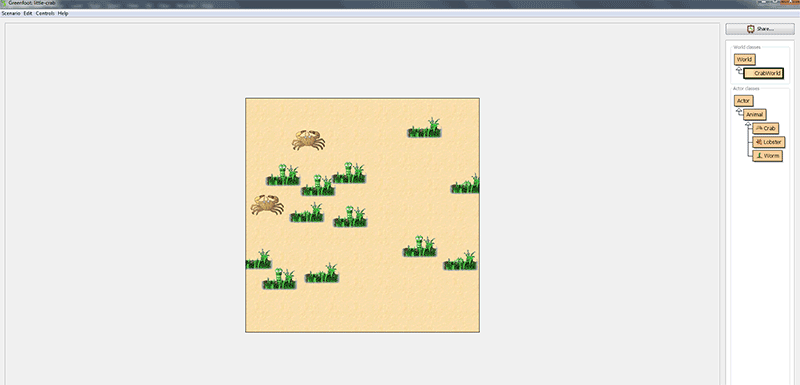
**{**

**Greenfoot.playSound("fanfare.wav");**

**Greenfoot.stop();**

**}**

1. Run the project and try it out.



1. Save the following image into the images folder which is inside your *crab* scenario folder:

[worm.gif](http://www.phswebs.com/storage/images2/worm.gif)

1. Edit → Import class and import the GifImage class.
2. Add the following line of code to the Worm class.

private GifImage img = new GifImage("name.gif");

1. Change name.gif to worm.gif so it matches the name of the picture that you just downloaded. The change is shown in red:

private GifImage img = new GifImage("**worm**.gif");

1. Add the following code to the act() method of the Worm class then close the class:

setImage(img.getCurrentImage());

1. Right-click on the Worm class and choose Set image and choose worm.gif as the image for this class.
2. Click on the Reset button then click Run. The worms should animate.
3. Open the source code for the CrabWorld class and change the for loop so that it goes from 1 to 30 instead of 1 to 100.

*Right now all the worms start with the exact same frame and they animate the exact same way. The next two steps are intended to add some variety. They start the animation for each worm on a random frame.*

1. Add the following method to the GifImage class:

public void setCurrentIndex()

{

currentIndex = Greenfoot.getRandomNumber(images.length);

}

1. Add the following constructor to the Worm class:

public Worm()

{

img.setCurrentIndex();

}

1. Run the project. The worms should pop up and down at different times.

*The two previous steps added code so that each worm starts with a different frame of the gif animation so they don’t all pop up and down at the same time.*

*Programming tasks are frequently put into blocks of code called methods. This makes the code more organized and more readable and if the tasks need to be repeated it does not require copy and pasting large blocks of code. Instead you simply call the method several times.*

*We are going to make methods for several tasks and then call these methods in either the constructor or the act method. Our program will still work exactly the same, but it will be better organized.*

*When adding methods to a class, the method (colored yellow by Greenfoot) should be inside the scope delimiters for the class { } but not inside the scope delimiters for any other method in the class.*

1. Add the following methods to the Crab class. You can copy and paste them into the class:

**public void lookForWorm()**

**{**

**}**

**public void turnAtEdge()**

**{**

**}**

**public void checkKeyPressed()**

**{**

**}**

1. Cut (Control-x) the code that does each task from the act() method and paste it into the corresponding method. The color of the code shown below matches the color of the method that it goes into. The code that has no color stays in the act() method:

**if ( atWorldEdge() )**

**{**

**turn(17);**

**}**

**if (Greenfoot.isKeyDown("left"))**

**{**

**turn(-4);**

**}**

**if (Greenfoot.isKeyDown("right"))**

**{**

**turn(4);**

**}**

move();

**if(canSee(Worm.class))**

**{**

**eat(Worm.class);**

**numWormsDevoured++;**

**if( numWormsDevoured >= 8)**

**{**

**Greenfoot.playSound("fanfare.wav");**

**Greenfoot.stop();**

**}**

**Greenfoot.playSound("slurp.wav");**

**}**

if( getImage() == image1 )

{

setImage(image2);

}

else

{

setImage(image1);

}

1. Call each of these methods in the act() method. The code you need to add is shown in red:

public void act()

{

move();

**lookForWorm();**

**turnAtEdge();**

**checkKeyPressed();**

if( getImage() == image1 )

{

setImage(image2);

}

else

{

setImage(image1);

}

}

1. Add the following line of code as the first line of the act() method of the Crab class:

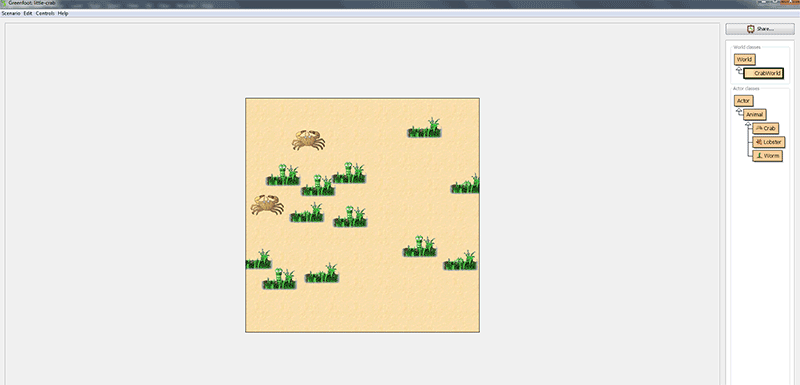
getWorld().showText("Number of worms devoured is " + numWormsDevoured, 50, 50);

1. In the constructor of the CrabWorld class, add the following line of code (put it underneath the super line):

setPaintOrder(Crab.class, Lobster.class, Worm.class);

1. Run the project. It should show the score so you will know when you are close to winning (8 worms devoured).

**The End!**



**Little Crab Part 2 - Challenge**

Open your crab scenario and File → Save as, and name it **YourName-crab2**. Make changes to the Little Crab Game to make it unique. You have lots of choices. Here are some ideas:

* using different images for the background and the actors;
* using more Actors; using different Actors
* not moving forward automatically, but only when the up-arrow key is pressed;
* building a two-player game by introducing a second keyboard-controlled class that listens to different keys;
* making new worms pop up when one is eaten (or at random times);
* and many more that you can come up with yourselves.

Do a combination of these or come up with your own creative ideas.

Once you are finished, do Scenario → Scenario Information then delete all the text that is there and list the modifications that you made. Once you have done this,

Here are some good examples. Be sure to have volume up and headphones on and enjoy the show!

[Moving Worms](https://www.greenfoot.org/scenarios/23029) - Move the Crabs with the arrow keys. Move the Lobsters with the A, S, D, and W keys.

[Teleporting Crabs](https://www.greenfoot.org/scenarios/23030) - Move the crab with the arrow keys. If you move over one of the two teleporters it will teleport the crab to a different location. If a lobster hits a bomb there is an explosion and the lobster is removed from the world.

[One Worm at a Time](https://www.greenfoot.org/scenarios/23031) - Only one worm is displayed at a time and when the crab eats this worm another worm appears in a random location. If the crab eats 8 worms it wins.

[Major League Little Crab Game](https://www.greenfoot.org/scenarios/23032) - Instructions are included with the game. This one has multiple scenes, music, sound effects, and both the crab and the lobsters are controlled by the keys.

[Pacman Crab Game](https://www.greenfoot.org/scenarios/23033) - It doesn't actually have any crabs, but it is still a Little Crab Game modification. The dots need to watch out for the powerful jaws of Pacman while Pacman needs to avoid the ghost.

**Turn in a GFAR of your Part 2 only to Canvas. In the comment section of the submission, list the new features/changes you made to your own version.**