

Lab 04 - Hands-on Programming (Part 2)!

PREPARED BY NICOLAS BERGERON

420-141-VA - GAME PROGRAMMING 1 - VANIER COLLEGE



Outline

Adaptation of the Crab Tutorial from Greenfoot.org by using Stride instead of Java (Part 2)

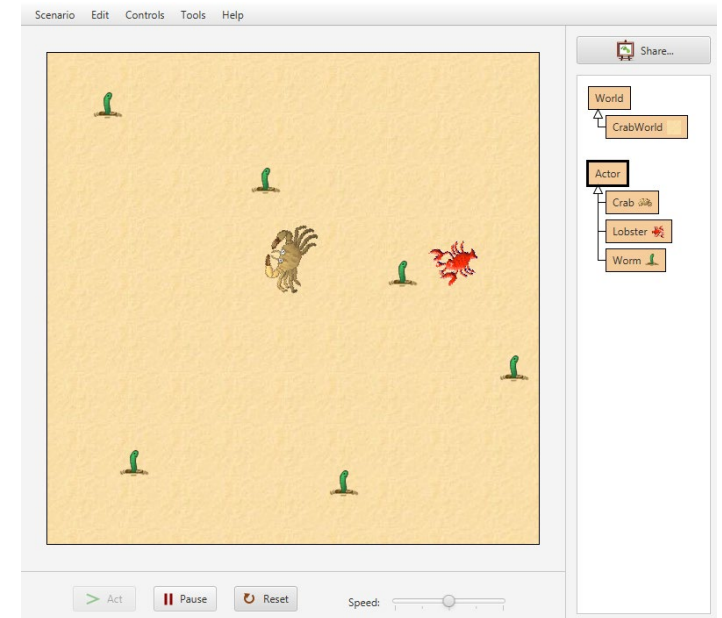
Adding Worm Actors

- Detecting actor intersection and Removing Actors from the World
- Refactoring code by adding Methods

Adding the Lobster Enemy

- Generating random numbers
- Using random numbers to change direction

Exercises



Setting up the Crab Scenario

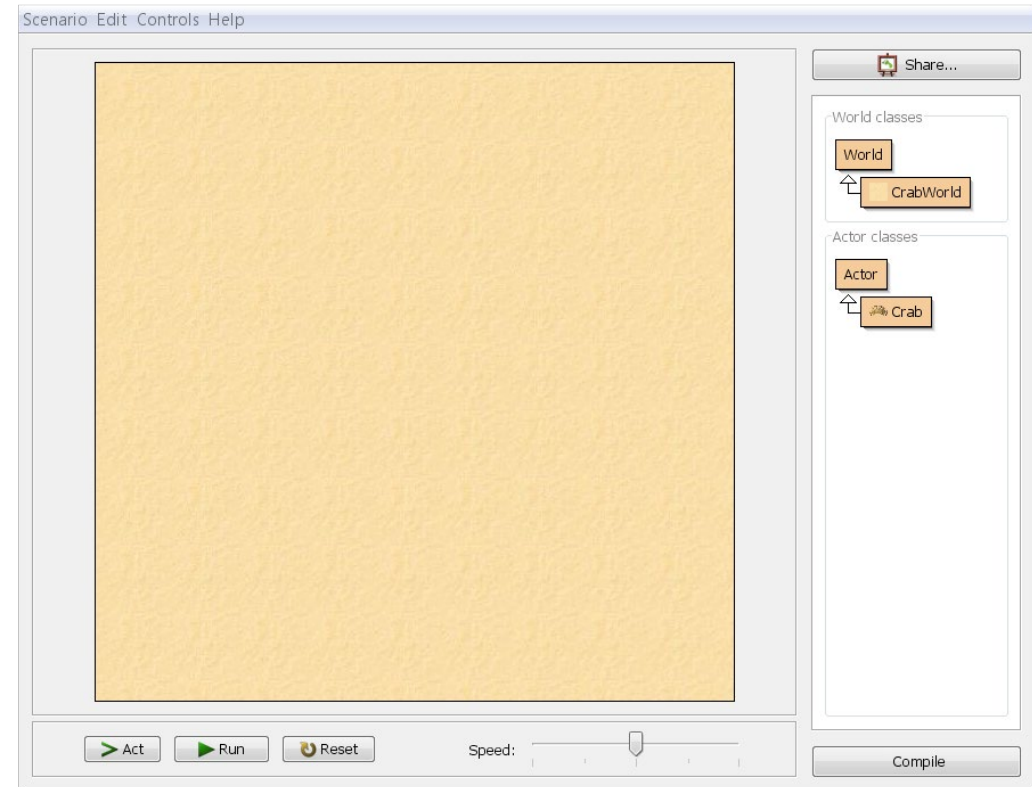
The Crabs Scenario

Download the Lab04.zip file from Omnivox, which contains the modern-crab Scenario from Lab 03.

Unzip the contents to somewhere on your hard disk.

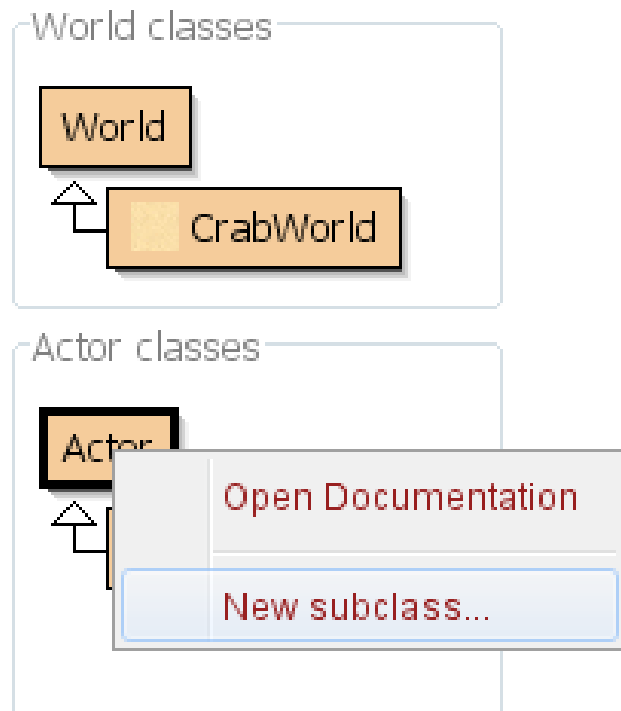
Open the scenario in that location with Greenfoot

You should see the standard Greenfoot interface, with an empty sandy world

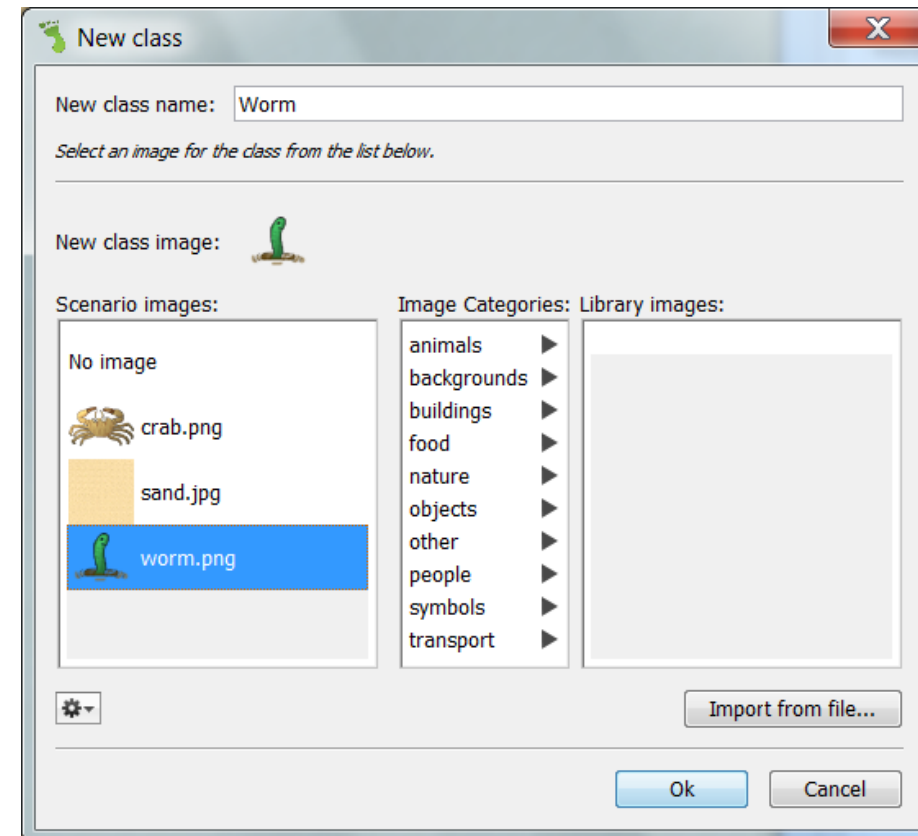


Create the Worm class

Right-click on the Actor class and select new subclass.



Name the class Worm and select the worm image



Getting Crabs to eat Worms

When the crab walks over a worm, it will eat the worm. The first step is to detect when the worm collides with the crab, and the second step is to remove the worm from the World when it happens.

Step 1 – Retrieve Worms at Crab location

Step 2 – When a Worm is found, remove it from the world

In Java and Stride, we can assign an Object to a variable. When a variable contains no object, its value is **null**. The if conditional statements will execute only then a worm is intersecting with the crab.

Open the code for the **Crab class** and add the code in the red outline:

Act - do whatever the Crab wants to do. This method is called whenever the 'Act' or 'Run' button gets pressed in the environment.

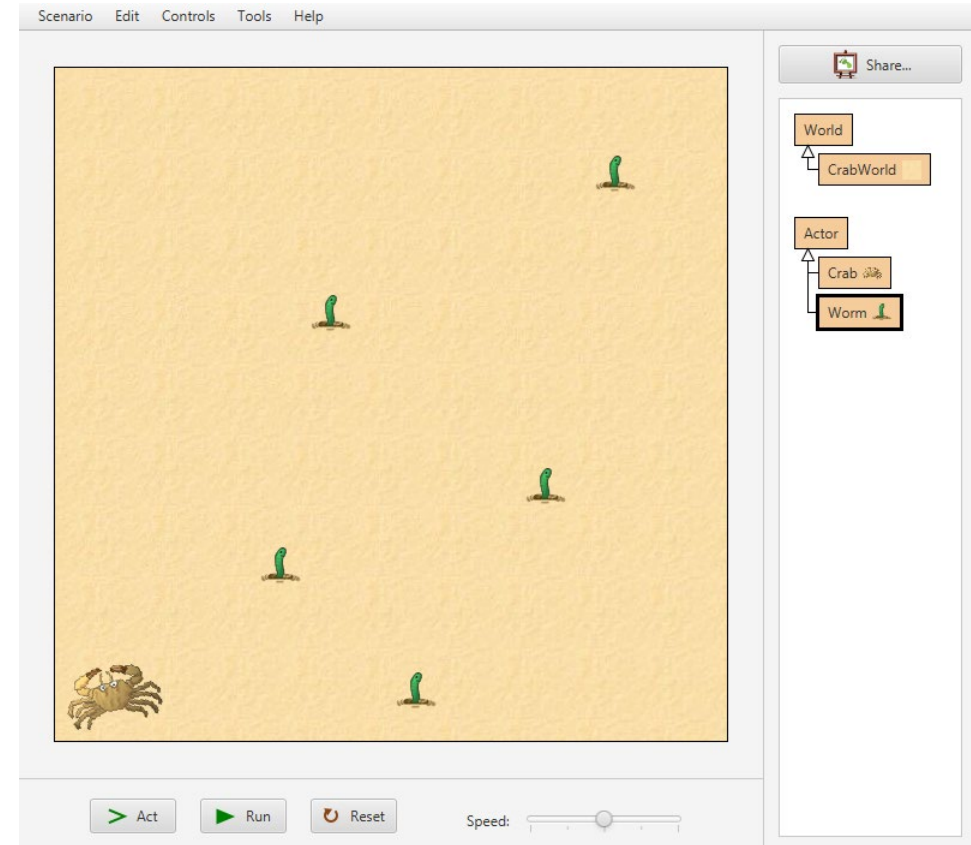
```
public void act() overrides method from Actor  
  
    move(4)  
    if ( Greenfoot.isKeyDown( "left" ) )  
        turn(-3)  
    if ( Greenfoot.isKeyDown( "right" ) )  
        turn(3)  
  
    var Actor worm = getOneIntersectingObject(Worm.class)  
    if ( worm != null )  
        var World world = getWorld()  
        world.removeObject(worm)
```

Test code - Instantiate Worms and a Crab

Instantiate a Crab and a few Worms inside the world.

Run the Scenario

Worms should disappear when the crab intersects with them



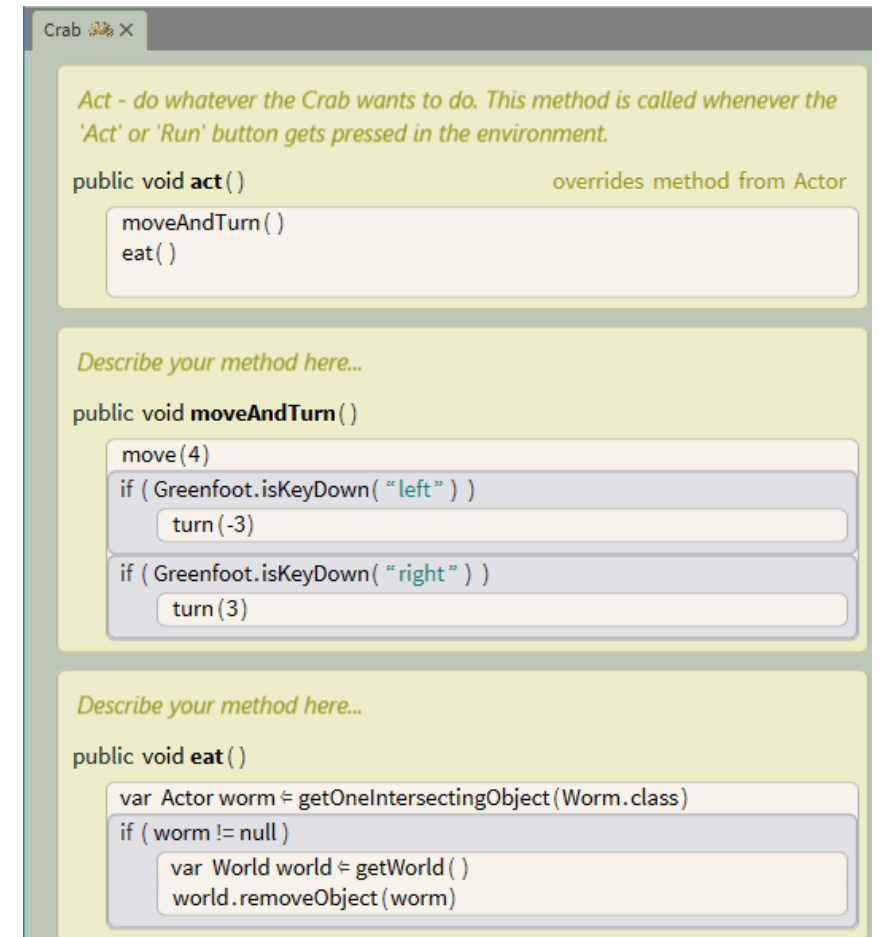
Refactor the Crab class


Refactoring is a term we use in Programming when we cleanup our code.

- In our example, the act method becomes a bit too complex. It updates the crab movement and makes the crab eat the worms.
- To improve the readability of our code, we can add methods to split these two behaviors

1. Create a method `moveAndTurn()`
2. Create a method `eat()`
3. Call these methods from `act()`

Note that you can drag and drop the code from `act()` to their respective method



```
Crab  X

Act - do whatever the Crab wants to do. This method is called whenever the
'Act' or 'Run' button gets pressed in the environment.

public void act()                                     overrides method from Actor
{
    moveAndTurn()
    eat()
}

Describe your method here...
public void moveAndTurn()
{
    move(4)
    if ( Greenfoot.isKeyDown( "left" ) )
    {
        turn(-3)
    }
    if ( Greenfoot.isKeyDown( "right" ) )
    {
        turn(3)
    }
}

Describe your method here...
public void eat()
{
    var Actor worm = getOneIntersectingObject(Worm.class)
    if ( worm != null )
    {
        var World world = getWorld()
        world.removeObject(worm)
    }
}
```


Saving the World

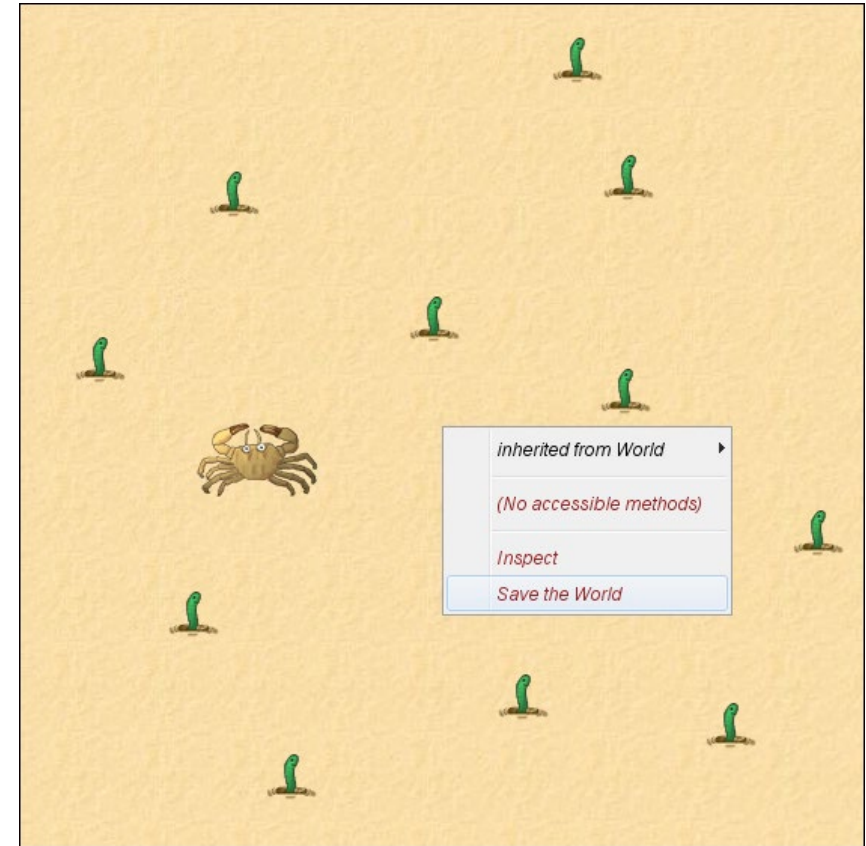
You are probably tired of always having to add new objects to the world every time we modify the code

It is possible to add some code to automatically create some worms and a crab for you, in fact, it is possible to get Greenfoot to write that code for you!

Populate the world with a Crab and Worms, and Right-click on the World. Select **Save the World**.

This will implement a **prepare()** method in your CrabWorld class.

Pressing **Reset** or instantiating a new CrabWorld will call **prepare()** to initialize actors in the world.



Playing Sounds

The Greenfoot class provides a method to play sounds:

- `Greenfoot.playSound(soundFile)`
- Documentation on the right.

In the project, there is a folder called sounds, with a sound effect named **eating.wav**

Playback this sound in the eat method

playSound

```
public static void playSound(java.lang.String soundFile)
```

Play sound from a file. The following formats are supported: AIFF, AU and WAV.

The file name may be an absolute path, a base name for a file located in the project directory or in the sounds directory of the project directory.

Parameters:

`soundFile` - Typically the name of a file in the sounds directory in the project directory.

Throws:

`java.lang.IllegalArgumentException` - If the sound can not be loaded.

public void eat ()

```
var Actor worm ← getOneIntersectingObject (Worm.class)
```

```
if ( worm != null )
```

```
var World world ← getWorld ( )
```

```
world.removeObject ( worm )
```

```
Greenfoot.playSound ( "eating.wav" )
```

Adding an Enemy

Fun games require skills to overcome challenges.
In this game, we will add a lobster moving in
random directions.

Yes, lobsters eat crab!

(and you may not know, yet ...)

Zombie lobsters turn Worms into Lobsters!



Random Numbers

Greenfoot provides a method that will give a random number (aka RNG, random number generator):

`getRandomNumber`

```
public static int getRandomNumber(int limit)
```

Return a random number between 0 (inclusive) and limit (exclusive).

Parameters:

`limit` - An upper limit which the returned random number will be smaller than.

Returns:

A random number within 0 to (limit-1) range.

Use this in your games to make them less repetitive. Nobody would play Monopoly without dice, or poker with cards always in the same order. But be careful to not make randomness overpower skills ([Extra Credits](#))



Lobster Random Movement

To make the Lobster move randomly, we will make it change direction 10% of time steps

- When random number between 0 ... 9 == 1

When it changes direction, it will turn between -45 ... 45 degrees

- (Random number between 0 and 90) - 45

To prevent the lobster getting stuck in a corner or on an edge, we also make it do a 90 degree turn when the lobster is on the edge.

Act - do whatever the Lobster wants to do. This method is called whenever the 'Act' or 'Run' button gets pressed in the environment.

```
public void act()
```

overrides method from Actor

```
    moveAround()
```

Describe your method here...

```
public void moveAround()
```

```
    move(4)
```

```
    if ( Greenfoot.getRandomNumber(10) == 1 )
```

```
        turn ( Greenfoot.getRandomNumber(90) - 45 )
```

```
    if ( isAtEdge() )
```

```
        turn (180)
```

Exercises: Finish coding the Lobster

Finish the lobster implementation

- make your crab disappear when it intersects with the lobster
- play a scary or game over sound when it happens!

Randomize the initial direction of lobsters when they get instantiated

- Implement a constructor and call turn with a random value between 0 and 359

To make things interesting, transform worms into lobsters when they get touched by a lobster

- Remove the worm on intersection with lobster
- Add a new lobster at this position
 - Look at Greenfoot documentation to find out how to add an actor
 - Hint: World AddObject method - Actor getX() and getY()

If you have time left, add your own twist to the game!!