AP Computer Science | Project 12 | Name:
Dr. Paul L. Bailey | Tuesday, November 15, 2019

We pause our study of Java to create a small game in the Greenfoot Integrated Development Environment. This will consist of moving crabs around on the beach.

I suggest that you change your folder structure as follows.

```
World               Contains all source files for the class
    Jdk             Contains all projects written with JDK
        P1_Hello
        P2_Geometry
        ...
        P9_Arrays
    BlueJ           Contains all projects written with BlueJ
    Greenfoot       Contains all projects written with Greenfoot
        P12_CrabWorld
    Netbeans        Contains all projects written with Netbeans
```

Program 1. Create a program in which crabs move around on the beach.

(a) Create the Scenario

1. Open the Greenfoot programming environment. Select `Scenario/New Scenario`. Name the scenario `CrabWorld`.
2. Go to the web site `http://plbailey79.github.io/portal` and download `PCScix10_CrabPrereqs.zip`. Install the images and sounds into the appropriate places of the `CrabWorld` project folder.
3. Right click the `World` tab. Select `New subclass`. Name it `CrabWorld`. Select a background image and push the `Ok` button.
4. Right click the `Actor` tab. Select `New subclass`. Name it `Crab`. Select an image and push the `Ok` button.
   `Compile` and `Run`.
   (N.B. there used to be a compile button, but Greenfoot 3.0 compiles automatically.)
5. Right click on the `Crab` tab. Select `New Crab` and place the crab in the world. `Save the world`. `Compile` and `Run`.
   `Pause`. Insert several more crabs into the world. `Run`.
6. Questions:
   (a) What is the relationship between `World` and `CrabWorld`?
   (b) What is the relationship between `Actor` and `Crab`?
   (c) What is the relationship between `CrabWorld` and `Crab`?
   (d) What does `Compile` do? What do `Run`, `Pause`, and `Reset` do?

(b) Assign Behavior

1. Right click the `Crab` tab and select `Open editor`.
2. Find the line of code which says `// Add your action code here.`
3. Replace this with `move(3);`
   `Compile`. Place a crab into the world. `Run`
4. Questions:
   (a) What does the 3 do in `move(3)`?
   (b) Where is the `move` method defined?

(c) Make Motion Continuous

1. In the Crab's act method, after the line containing `move`, insert this code:

```
if (getX() <= 1 ||
    getX() >= getWorld().getWidth() - 1  ||
    getY() <= 1 ||
    getY() >= getWorld().getHeight() - 1)
{
    turn(7);
}
```

2. `Compile`. Insert a crab into the world. `Run`.
   `Pause`. Insert several more crabs into the world. `Run`.
3. Questions:
   (a) What do `getX` and `getY` do?
   (b) Where are the methods `getX`, `getY`, and `turn` defined?

(d) Drunken Crab

1. In the Crab's act method, after the line containing `move`, insert this code:

```
if (Greenfoot.getRandomNumber(100) < _%ageProbabilty_)
{
    turn(Greenfoot.getRandomNumber(_#degrees_));
}
```

   `Compile`. Insert a crab into the world. `Run`.
2. Note that the crab always turns in one direction. Correct this.
3. `Compile`. Insert a crab into the world. `Run`.
   `Pause`. Insert several more crabs into the world. `Run`.
4. Questions:
   (a) Where is the `getRandomNumber` method defined? Can we see it?
   (b) Is a positive turn clockwise or counterclockwise?

(e) Refactoring Methods

Create new methods, cut code from the `act` method, and paste into the new methods.

1. Cut and paste to create a `atWorldsEdge` method after the `act` method:

```
public boolean atWorldsEdge()
{
    return
       (getX() <= 1 ||
        getX() >= getWorld().getWidth() - 1  ||
        getY() <= 1 ||
        getY() >= getWorld().getHeight() - 1);
}
```

2. Cut and paste to create a `richochet` method after the `act` method:

```
public void richochet()
{
    if (atWorldsEdge())
    {
        turn(7);
    }
}
```

3. Cut and paste to create a `stagger` method after the `act` method:

```
public void stagger()
{
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(30) - 15);
    }
}
```

4. Modify the `act` method to use the new `ricochet` and `stagger` methods.
5. `Compile`. Insert a crab into the world. `Run`.
   `Pause`. Insert several more crabs into the world. `Run`.
6. Questions:
   (a) What is the point of this exercise?

(f) Feed the Crabs

1. Create a subclass of the `Actor` class, and call it `Worm`. Assign the worm image to this class.
2. The worm does not act. Instead, when a crab is close to a worm, it eats it. Create the following `eat` method in `Crab` class:

```
public void eat()
{
    Actor worm;
    worm = getOneObjectAtOffset(0, 0, Worm.class);
    if (worm != null)
    {
        World world;
        world = getWorld();
        world.removeObject(worm);
    }
}
```

3. Invoke the `Crab.eat` method from the `Crab.act` method.
4. Place many worms and a few crabs in the world, and `Run`. Then `Pause`, `Reset`.
5. Place many worms and a few crabs in the world, Save the World, and `Run`. Then `Pause`, `Reset`.
6. Find the `slurp.wav` file and place it in the `sounds` directory of your scenario. Add this line of code at the bottom of the `eat` method: `Greenfoot.playSound("slurp.wav");` and `Run`.
7. Questions:
   (a) What does `getOneObjectAtOffset` do? What does "offset" probably mean? What do the parameters `(0, 0, Worm.class)` do?
   (b) What does `getWorld` do?
   (c) What does Save the World do?

(g) Control Your Crab

1. Add this `control` method to the Crab class:

```
public void control()
{
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-3);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(3);
    }
}
```

2. Modify the crab's `act` method to invoke the `control` method. `Compile` and `Run`.
3. Modify the `control` method to accept input for forward motion if the `up` key is pressed and backward motion if the `down` key is pressed. `Compile` and `Run`.
4. Comment out the `move`, `ricochet` and `stagger` calls from the `act` method.
5. Open the `CrabWorld` class in the editor. Modify the `prepare` method so that there is only one crab. `Compile` and `Run`.
6. Questions:
   (a) What class contains the `isKeyDown` method, and why?
   (b) What other methods are in this class might be useful in the game?

(h) Anteating Crabs

1. Create a subclass of `Actor` called `Ant`.
2. Add `move(2)`, `stagger()`, and `ricochet()` to the `Ant.act` method.
   Place lots of ants into the world and save the world. `Compile` and `Run`
3. Modify the Crab class so that a crab can eat ants if he catches them. Since the crab still eats worms, there are at least two ways to make this modification: copying the `eat` class to make an `eatAnts` class, or making a parameter such as `eat(Class cls)`. Of course, the sound of eating an ant should be a crunch instead of a slurp.

Program 2. We continue improving our Crab game by adding an antagonist - lobsters that eat crabs!

(a) Refactor to Create an Animal Class

1. Create a subclass of `Actor` and call it `Animal`. This class does not need an image.
2. Copy the `stagger`, `ricochet`, `control`, and `atWorldsEdge` methods into the `Animal` class.
3. Modify the `Crab` class to inherit from `Animal`, and remove the `stagger`, `ricochet`, `control`, and `atWorldsEdge` methods from the `Crab` class.
4. Repeat the above step for the `Ant` class.
   `Compile` and `Run`.

(b) Create a Lobster Class

1. Create a subclass of `Animal` called `Lobster`, and assign an image to it.
2. Modify the Lobster's `act` method to move, stagger, and ricochet. It should move a little faster than a crab.
3. Modify the Lobster to eat crabs. `Compile`. Put some lobsters in the world. `Run`.

(c) End of Game - Loosing

1. Create a subclass of `Actor` and call it `Message`; it does not need an image.
2. Edit the source code of the `Message` class to be:

```
import greenfoot.*;

public class Message extends Actor
{
    private int width;
    private int height;
    private String msgTxt;

    public Message(int width, int height, String msgTxt)
    {
        this.width = width;
        this.height = height;
        this.msgTxt = msgTxt;
        updateImage();
    }

    public void updateImage()
    {
        GreenfootImage image = new GreenfootImage(width, height);
        image.setColor(Color.CYAN);
        image.fill();
        GreenfootImage txtImg = new GreenfootImage(
            msgTxt,
            36,
            Color.YELLOW,
            Color.BLUE);
        image.drawImage(
            txtImg,
            (image.getWidth() - txtImg.getWidth()) / 2,
            (image.getHeight() - txtImg.getHeight()) / 2);
        setImage(image);
    }
}
```

3. Insert the following method into the `CrabWorld` class:

```
    public void act()
    {
        if (getObjects(Crab.class).isEmpty())
        {
            addObject(
                new Message(getWidth() / 3, getHeight() / 3, "GAME OVER"),
                getWidth() / 2,
                getHeight() / 2);
            Greenfoot.playSound("gameover.wav");
            Greenfoot.stop();
            return;
        }
    }
```

(d) End of Game - Winning (Keeping Score)

1. Add an instance variable to the Crab class by inserting this line of code after the opening brace of the class:

   ```
   private int score = 0;
   ```

2. Modify the `eat` method(s) to store 1 points for each worm and 2 points for each ant eaten.
3. Modify the game to end with an appropriate message and/or sound when the crab reaches 25 points.

(e) Showing the Score

1. Create a new subclass of `Actor` and call it `Counter`, using the `counter.png` image. Place a Counter object in the lower left corner of the world and Save the World.
2. Edit the source code of `Counter` class to be:

```
import greenfoot.*;

public class Counter extends Actor
{
    private static final Color transparent = new Color(0, 0, 0, 0);
    private GreenfootImage background;
    private int value;

    public Counter()
    {
        background = getImage();
        value = 0;
        updateImage();
    }

    public int getValue()
    {
        return value;
    }

    public void setValue(int newValue)
    {
        value = newValue;
        updateImage();
    }

    private void updateImage()
    {
        GreenfootImage image = new GreenfootImage(background);
        GreenfootImage text = new GreenfootImage(
            "" + value,
            22,
            Color.BLACK,
            transparent);
        image.drawImage(
            text,
            (image.getWidth() - text.getWidth())/2,
            (image.getHeight() - text.getHeight())/2);
        setImage(image);
    }
}
```

3. Modify the `Crab` class to find the `Counter` object in its world, and invoke that counter's `setValue` method to update the score.

(f) Cleaning up the code: Go through all your classes and make any necessary updates to make sure you are following the our proper conventions:

- All class names should be capitalized.
- All variable and method names should start with lower case.
- Class, variable, or method names which consist of more than one word should capitalize the all words following the first word. For example, `deadend` should be `DeadEnd` if it is a class, or `deadEnd` if it is a variable.
- Following the Allman brace placement and indentation style. According to Wikipedia, *This style puts the brace associated with a control statement on the next line, indented to the same level as the control statement. Statements within the braces are indented to the next level.* For example:

```
// Do it this way.
if (x == y)
{
    something();
    somethingmore();
}
else
{
    somethingelse();
}
```

Don't use the K & R style, which puts braces on the same line with the control statement:

```
// Don't do it this way:
if (x == y) {
    something();
    somethingmore();
} else {
    somethingelse();
}
```

The AP Computer Science exam uses the Allman style.