Part 4：Interacting Objects

被吴沛霖添加，被吴沛霖最后更新于Aug 15, 2015

Part 4: Interacting Objects

The Critter Class

Critters are actors that share a common pattern of behavior, but the details may vary for each type of critter. When a critter acts, it first gets a list of actors to process. It processes those actors and then generates the set of locations to which it may move, selects one, and moves to that location.

Different types of critters may select move locations in different ways, may have different ways of selecting among them, and may vary the actions they take when they make the move. For example, one type of critter might get all the neighboring actors and process each one of them in some way (change their color, make them move, and so on). Another type of critter may get only the actors to its front, front-right, and front-left and randomly select one of them to eat. A simple critter may get all the empty neighboring locations, select one at random, and move there. A more complex critter may only move to the location in front or behind and make a turn if neither of these locations is empty.

Each of these behaviors fits a general pattern. This general pattern is defined in the act method for the Critter class, a subclass of Actor. This act method invokes the following five methods.

```
ArrayList<Actor> getActors()
void processActors(ArrayList<Actor> actors)
ArrayList<Location> getMoveLocations()
Location selectMoveLocation(ArrayList<Location> locs)
void makeMove(Location loc)
```

These methods are implemented in the Critter class with simple default behavior--- see the following section. Subclasses of Critter should override one or more of these methods.

It is usually not a good idea to override the *act* method in a *Critter* subclass. The *Critter* class was designed to represent actors that process other actors and then move. If you find the *act* method unsuitable for your actors, you should consider extending Actor, not *Critter*.

Do You Know?

Set 7

The source code for the Critter class is in the critters directory

1. What methods are implemented in Critter?
2. What are the five basic actions common to all critters when they act?
3. Should subclasses of Critter override the getActors method? Explain.
4. Describe the way that a critter could process actors.
5. What three methods must be invoked to make a critter move? Explain each of these methods.
6. Why is there no Critter constructor?

Default Critter Behavior

Before moving, critters process other actors in some way. They can examine them, move them, or even eat them.

There are two steps involved:

1. Determination of which actors should be processed
2. Determination of how they should be processed

The getActors method of the Critter class gets a list of all neighboring actors. This behavior can be inherited in subclasses of Critter. Alternatively, a subclass can decide to process a different set of actors, by overriding the getActors method.

The processActors method in the Critter class eats (that is, removes) actors that are not rocks or critters. This behavior is either inherited or overridden in subclasses.

When the critter has completed processing actors, it moves to a new location. This is a three-step process.

1. Determination of which locations are candidates for the move
2. Selection of one of the candidates
3. Making the move

Each of these steps is implemented in a separate method. This allows subclasses to change each behavior separately.

The Critter implementation of the getMoveLocations method returns all empty adjacent locations. In a subclass, you may want to compute a different set of locations. One of the examples in the case study is a CrabCritter that can only move sideways.

Once the candidate locations have been determined, the critter needs to select one of them. The Critter implementation of selectMoveLocation selects a location at random. However, other critters may want to work harder and pick the location they consider best, such as the one with the most food or the one closest to their friends.

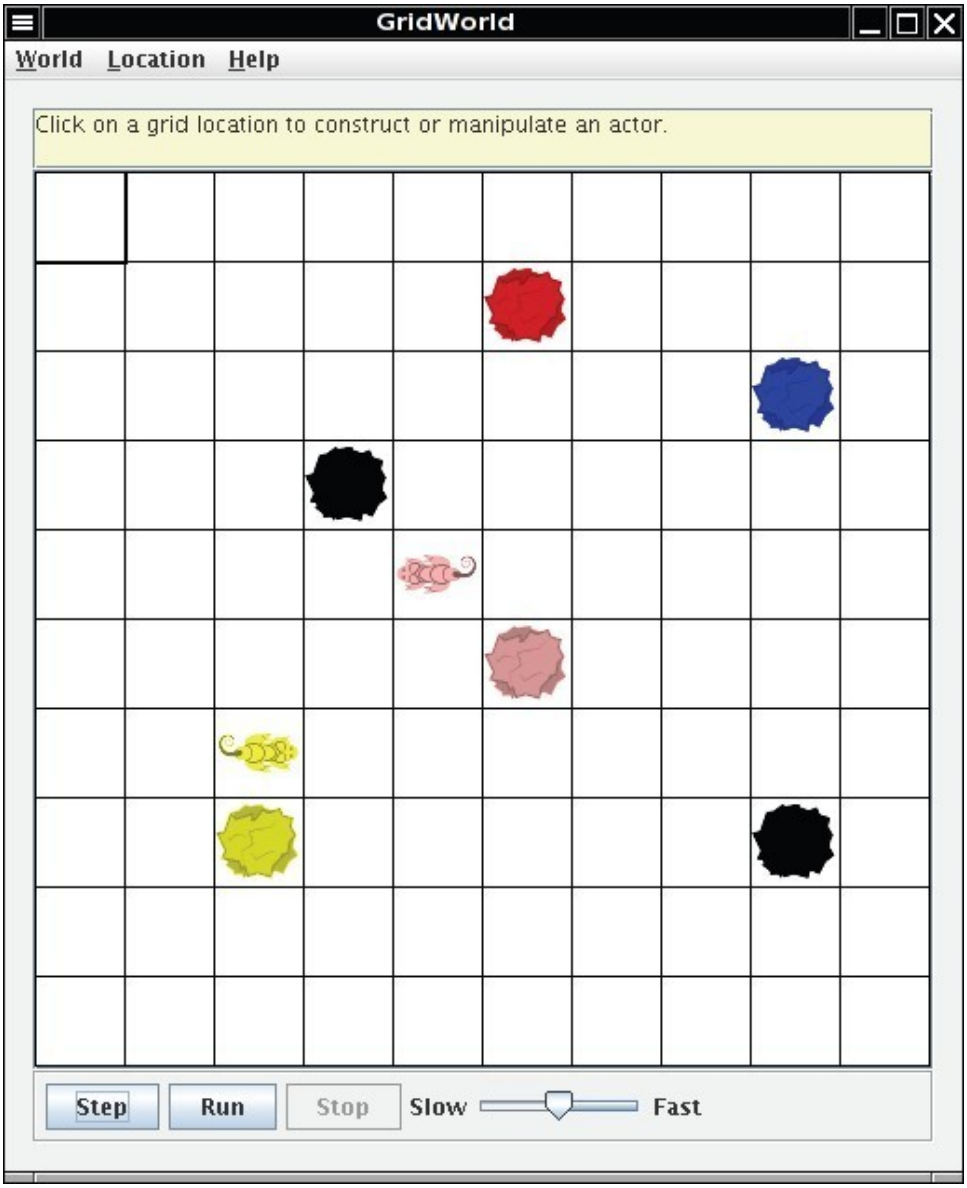
Finally, when a location has been selected, it is passed to the makeMove method. The makeMove method of the Critter class simply calls moveTo, but you may want to override the makeMove method to make your critters turn, drop a trail of rocks, or take other actions.

Note that there are postconditions on the five Critter methods called by act. When you override these methods, you should maintain these postconditions.

The design philosophy behind the Critter class is that the behavior is carried outin separate phases, each of which can be overridden independently. Thepostconditions help to ensure that subclasses implement behavior that is consistentwith the purpose of each phase.

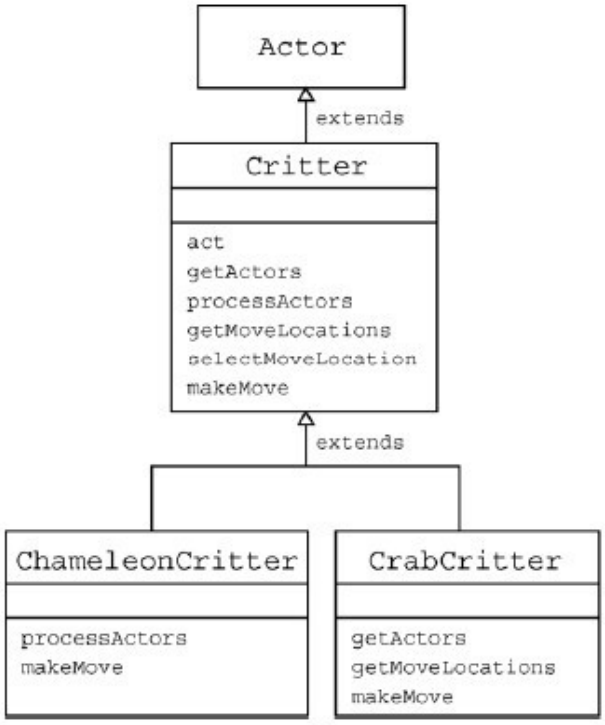
Extending the Critter Class

The ChameleonCritter class defines a new type of critter that gets the same neighboring actors as a Critter. However, unlike a Critter, a ChameleonCritter doesn't process actors by eating them. Instead, when a ChameleonCritter processes actors, it randomly selects one and changes its own color to the color of the selected actor.



The ChameleonCritter class also overrides the makeMove method of the Critter class. When a ChameleonCritter moves, it turns toward the new location.

The following figure shows the relationships among Actor, Critter, andChameleonCritter, as well as the CrabCritter class that is discussed in the following section.



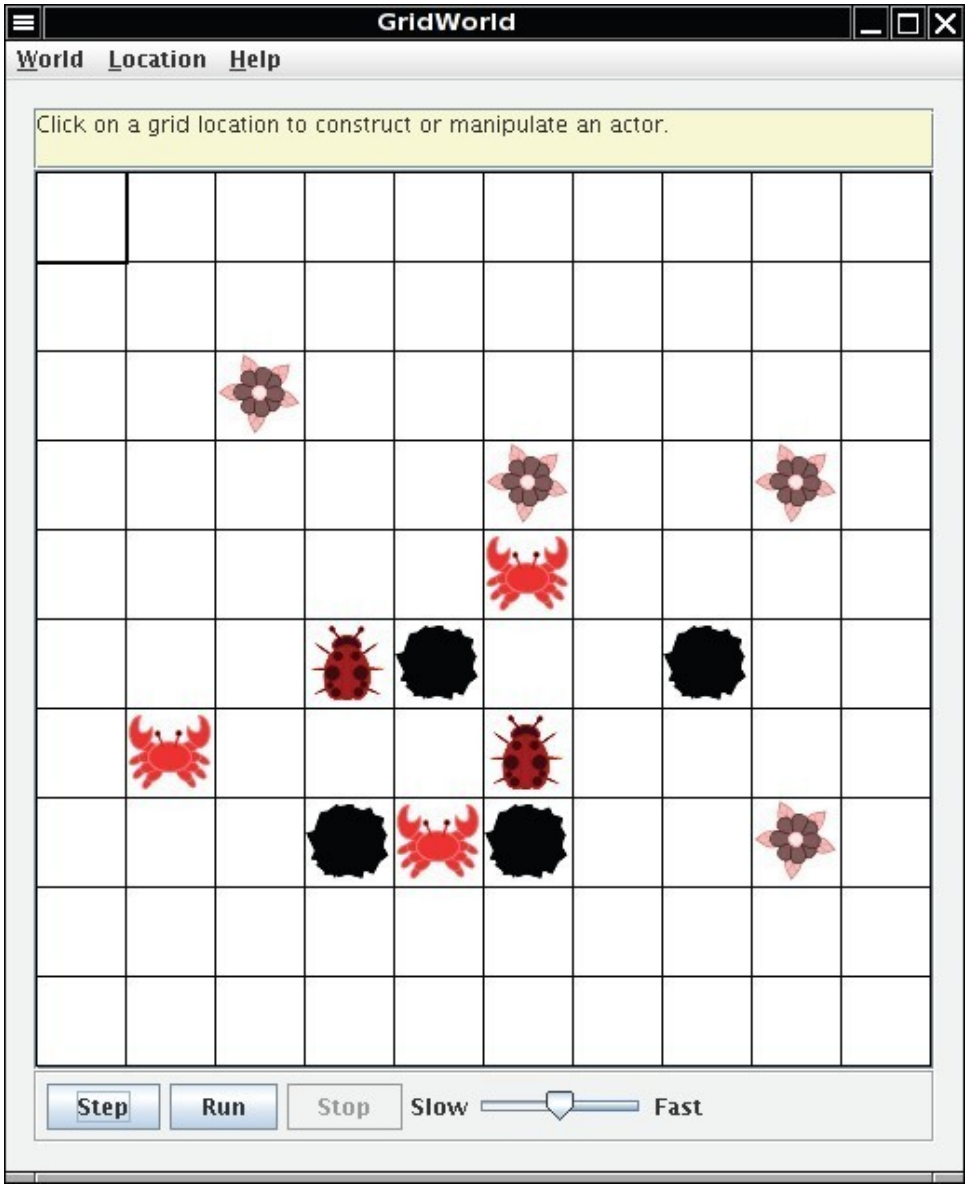
Do You Know?
Set 8

The source code for the ChameleonCritter class is in the critters directory

1. Why does act cause a ChameleonCritter to act differently from a Critter even though ChameleonCritter does not override act?
2. Why does the makeMove method of ChameleonCritter call super.makeMove?
3. How would you make the ChameleonCritter drop flowers in its old location when it moves?
4. Why doesn't ChameleonCritter override the getActors method?
5. Which class contains the getLocation method?
6. How can a Critter access its own grid?

Another Critter

A CrabCritter is a critter that eats whatever is found in the locations immediately in front, to the right-front, or to the left-front of it. It will not eat a rock or another critter (this restriction is inherited from the Critter class). A CrabCritter can move only to the right or to the left. If both locations are empty, it randomly selects one. If a CrabCritter cannot move, then it turns 90 degrees, randomly to the left or right.



Do You Know?

Set 9

The source code for the CrabCritter class is reproduced at the end of this part of GridWorld.

1. Why doesn't CrabCritter override the processActors method?
2. Describe the process a CrabCritter uses to find and eat other actors. Does it always eat all neighboring actors? Explain.
3. Why is the getLocationsInDirections method used in CrabCritter?
4. If a CrabCritter has location (3, 4) and faces south, what are the possible locations for actors that are returned by a call to the getActors method?
5. What are the similarities and differences between the movements of a CrabCritter and a Critter?
6. How does a CrabCritter determine when it turns instead of moving?
7. Why don't the CrabCritter objects eat each other?

Exercises

1. Modify the processActors method in ChameleonCritter so that if the list of actors to process is empty, the color of the ChameleonCritter will darken (like a flower).

In the following exercises, your first step should be to decide which of the five methods~~ getActors, processActors, getMoveLocations, selectMoveLocation, and makeMove~~ should be changed to get the desired result.

2. Create a class called ChameleonKid that extends ChameleonCritter as modified in exercise 1. A ChameleonKid changes its color to the color of one of the actors immediately in front or behind. If there is no actor in either of these locations, then the ChameleonKid darkens like

the modified ChameleonCritter.

3. Create a class called RockHound that extends Critter. A RockHound gets the actors to be processed in the same way as a Critter. It removes any rocks in that list from the grid. A RockHound moves like a Critter.

4. Create a class BlusterCritter that extends Critter. A BlusterCritter looks at all of the neighbors within two steps of its current location. (For a BlusterCritter not near an edge, this includes 24 locations). It counts the number of critters in those locations. If there are fewer than c critters, the BlusterCritter's color gets brighter (color values increase). If there are c or more critters, the BlusterCritter's color darkens (color values decrease). Here, c is a value that indicates the courage of the critter. It should be set in the constructor.

5. Create a class QuickCrab that extends CrabCritter. A QuickCrab processes actors the same way a CrabCritter does. A QuickCrab moves to one of the two locations, randomly selected, that are two spaces to its right or left, if that location and the intervening location are both empty. Otherwise, a QuickCrab moves like a CrabCritter.

6. Create a class KingCrab that extends CrabCritter. A KingCrab gets the actors to be processed in the same way a CrabCritter does. A KingCrab causes each actor that it processes to move one location further away from the KingCrab. If the actor cannot move away, the KingCrab removes it from the grid. When the KingCrab has completed processing the actors, it moves like a CrabCritter.

Group Activity

Organize groups of 2--4 students.

1. Specify: Each group specifies a new creature that extends Critter. The specifications must describe the properties and behavior of the new creature in detail.
2. Design: Determines the needed variables and basic algorithms for the creature according to the spaecifications.
3. Code: Each group implements the code for the creature in the specification.
4. Test: Each group writes test cases for the creature and verify that the implementations meet the specifications.

CrabCritter.Java

```
import info.gridworld.actor.Actor;
import info.gridworld.actor.Critter;
import info.gridworld.grid.Grid;
import info.gridworld.grid.Location;
import java.awt.Color;
import java.util.ArrayList;
/**
* A CrabCritter looks at a limited set of neighbors when it eats and moves.
* This class is not tested on the AP CS A and AB Exams.
*/
public class CrabCritter extends Critter
{
    public CrabCritter()
    {
        setColor(Color.RED);
    }
/**
* A crab gets the actors in the three locations immediately in front, to its
* front-right and to its front-left
* @return a list of actors occupying these locations
*/
public ArrayList<Actor> getActors()
{
    ArrayList<Actor> actors = new ArrayList<Actor>();
    int[] dirs ={ Location.AHEAD, Location.HALF_LEFT, Location.HALF_RIGHT };
    for (Location loc : getLocationsInDirections(dirs))
    {
        Actor a = getGrid().get(loc);
        if (a != null)
            actors.add(a);
    }
    return actors;
}
/**
* @return list of empty locations immediately to the right and to the left
*/
public ArrayList<Location> getMoveLocations()
```

```
{
    ArrayList<Location> locs = new ArrayList<Location>();
    int[] dirs = { Location.LEFT, Location.RIGHT };
    for (Location loc : getLocationsInDirections(dirs))
        if (getGrid().get(loc) == null)
            locs.add(loc);
    return locs;
}
/**
* If the crab critter doesn't move, it randomly turns left or right.
*/
public void makeMove(Location loc)
{
    if (loc.equals(getLocation()))
    {
        double r = Math.random();
        int angle;
        if (r < 0.5)
            angle = Location.LEFT;
        else
            angle = Location.RIGHT;
        setDirection(getDirection() + angle);
    }
    else
        super.makeMove(loc);
}
/**
* Finds the valid adjacent locations of this critter in different
* directions.
* @param directions - an array of directions (which are relative to the
* current direction)
* @return a set of valid locations that are neighbors of the current
* location in the given directions
*/
public ArrayList<Location> getLocationsInDirections(int[] directions)
{
    ArrayList<Location> locs = new ArrayList<Location>();
    Grid gr = getGrid();
    Location loc = getLocation();
    for (int d : directions)
    {
        Location neighborLoc = loc.getAdjacentLocation(getDirection() + d);
        if (gr.isValid(neighborLoc))
            locs.add(neighborLoc);
    }
    return locs;
}
}
```

Like Be the first to like this None

评论

张子轩 发表：

Aug 20, 2015

Sofa.

徐欣 发表：

Aug 20, 2015

捕捉一只野生的紫萱女神0.0

张子轩 发表： Aug 21, 2015

= =

何朝东 发表： Aug 21, 2015

k

刘健坤 发表： Aug 21, 2015

So difficult (@.@)

罗思成 发表： Aug 21, 2015

Part 4: 交互对象
Critter

Critter共享共同行为模式，但细节可能不同。当一个Critter的执行动作时，它首先得到一个 actor 列表来处理。它处理这些 actor，然后产生一组它可能移动到的位置，选择一个，并移动到该位置

不同类型的Critter可能会用不同的方式来选择移动位置。例如，一种Critter可能会获取的所有相邻 actor 并且以某种方式（改变它们的颜色，使它们移动，等等）处理它们中的每一个。另一种Critter可能只获取他前面，前右和前左的 actor 并且随机选择其中一个吃掉。一个简单Critter可能会获取周边空位，随机选择一个并移动到那里。更复杂的Critter移动它前面或后面，并且在前面和后面都不为空时转弯

所有这些行为符合一种普遍模式。这种普遍模式在 Critter 类中的 act 方法中定义。act 方法调用以下五种方法。

ArrayList<Actor> getActors()
void processActors(ArrayList<Actor> actors)
ArrayList<Location> getMoveLocations()
Location selectMoveLocation(ArrayList<Location> locs)
void makeMove(Location loc)

这些方法在Critter 类实现，并且有简单的默认行为

Critter的子类应重载一个或多个这些方法。

重载Critter的子类的act方法通常不是一个好主意。Critter类是被设计来处理其他 actor，然后移动的。如果您发现act方法不适合你的actor，你应该考虑继承 Actor，而不是Critter。

你知道吗?

Set 7

Critter 类源代码在 critters 目录中

1.Critter类中实现了什么样的方法?
2.五个适用于所有的critters（当他们执行 act 时）的基本动作是什么?
3.Critter的子类应该覆盖getActors方法吗？请解释。
4.描述一个critter可能的处理actor的方式。
5.为了使critter移动，哪三种方法必须被调用？解释这些方法。
6.为什么没有Critte的构造函数?

默认的Critter行为

移动前，critters 用某种方式处理其他 actor。他们可以检查它们，移动它们，甚至吃掉他们（蛤蛤。

两个步骤:

1.确定哪些actor应该被处理
2.确定应如何处理

在critter类的getActors方法获取所有相邻的actor。此行为可在critter的子类继承。

另外，一个子类可以通过覆盖getActors方法, 决定处理一组不同的actor，

所述在critter类的processActors方法吃（即移除）掉不是岩石或critter的 actor。这种行为可以被子类继承或重写。

当critter已完成了处理actors，它移动到一个新的位置。这是一个三步过程。

1.测定哪些位置来移动
2.选择其中之一
3.移动

每个步骤是在一个单独的方法来实现的。这使得子类可以分别改变各行为

该critter的getMoveLocations方法将返回所有空相邻位置。在子类中，您可能要计算一组不同的位置。其中一个例子是只能向侧面移动的CrabCritter

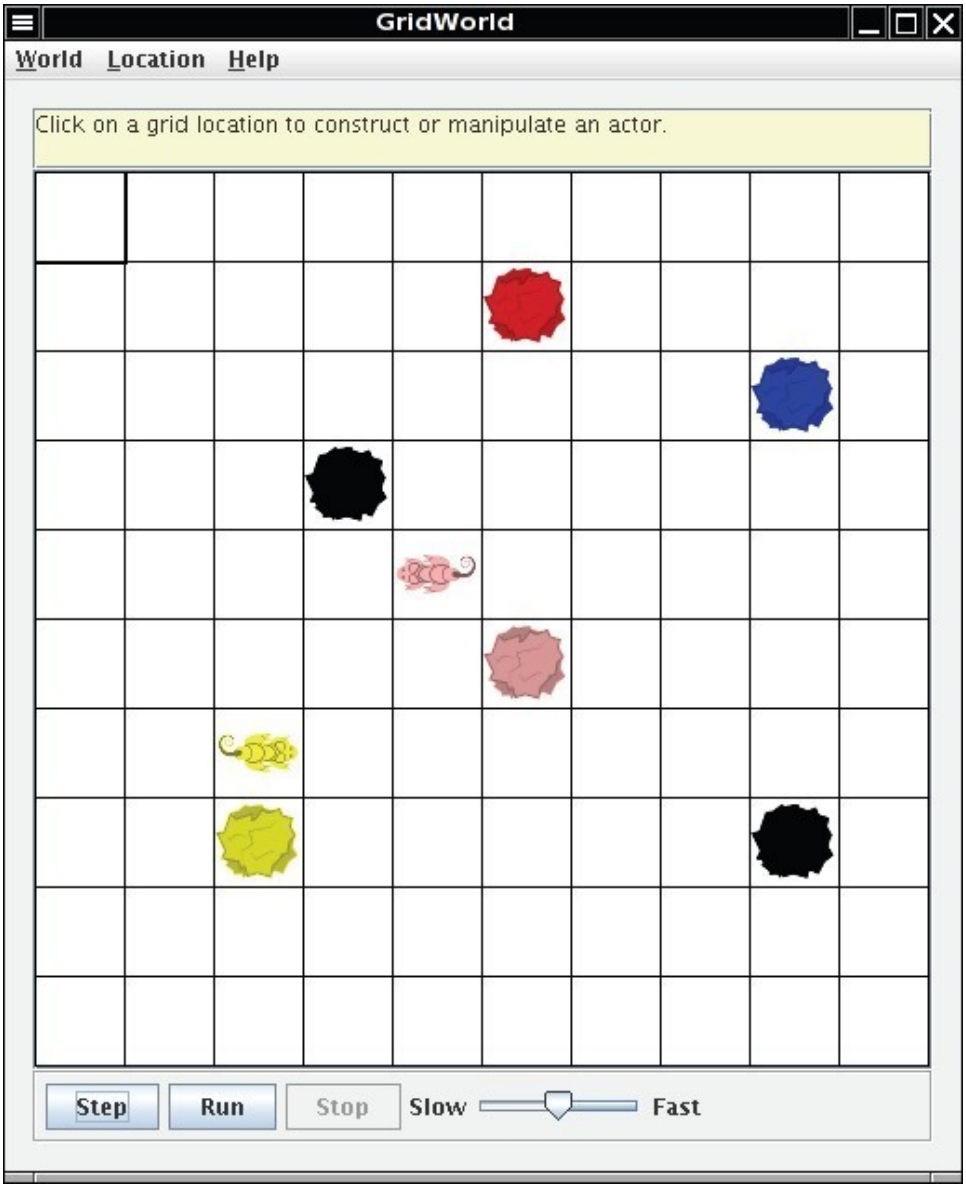
一旦候选位置已被确定时，critter需要选择其中之一。该critter的selectMoveLocation方法可能会随机选择一个位置。然而，其他的critter可能会挑选他们认为最佳的位置，如一个有最多的食物的食物或最接近他们的朋友的位置。

最后，当一个位置已被选择时，它会被传递给makeMove方法。critter类的makeMove方法调用 moveTo，但你可能要重写makeMove方法，使你的critter旋转，留下一串石头的轨迹，或做其他事情。

请注意，五种critter的方法中有后置条件。当你重写这些方法，你应该维护这些后置条件。

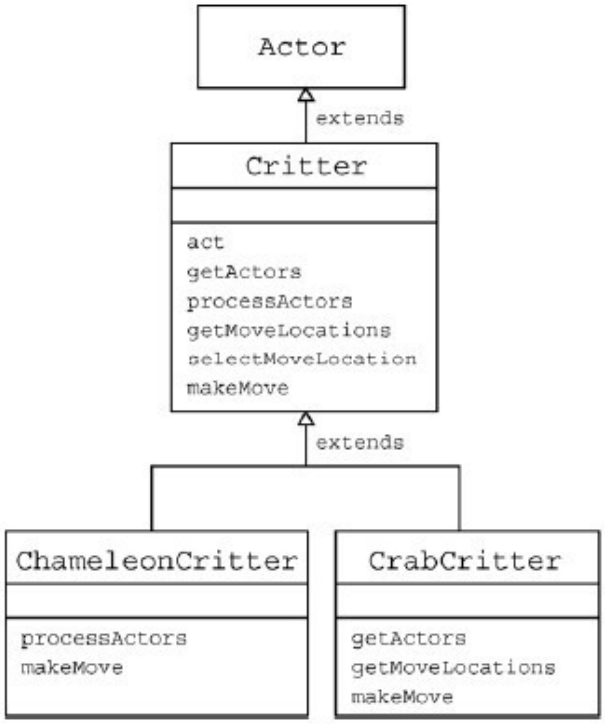
Critter类背后的设计理念是行为在不同的段执行，其中每一个都可以独立被覆盖。后置条件有助于确保子类实现与每个阶段的目的是一致的的行为，

扩展critter类
该ChameleonCritter类定义了一种新型的critter， 和critter一样，获取相邻的 actor。然而和critter不同的是，一ChameleonCritter不会吃掉 actor。相反，当一个ChameleonCritter处理actor，它随机选择一个，并改变它自己的颜色到选定 actor的颜色。



该ChameleonCritter类还覆盖了critter类的makeMove方法。当ChameleonCritter移动时，它转向新的位置。

下图显示了actor，critter，ChameleonCritter和CrabCritter之间的关系



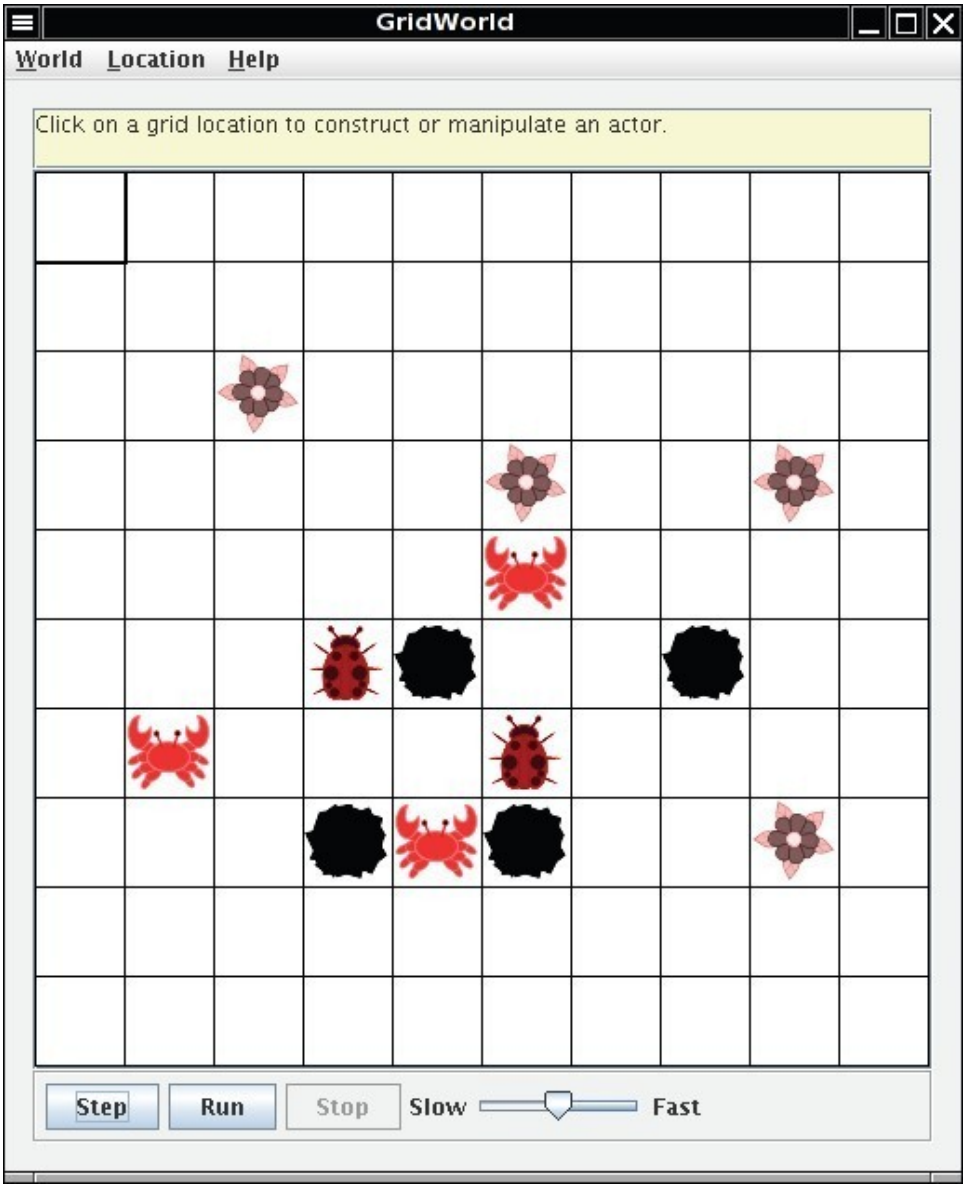
你知道吗?
Set 8

ChameleonCritter 类源代码在 critters 目录中

1.为什么ChameleonCritter没有重写 act 方法，却在执行 act 会产生和 Critter不同的效果?
2.为什么ChameleonCritter的makeMove方法调用super.makeMove?
3.你会如何让ChameleonCritter移动时在其原来的位置留下花朵?
4.ChameleonCritter为什么不覆盖getActors方法?
5.哪些类包含getLocation方法?
6.如何可以将critter访问自己的网格?

又一个critter

CrabCritter是critter的子类，当他发现前方，右前方，左前方有 actors，会吃掉他们。它不会吃石头或其他critter（此限制由critter类继承）。CrabCritter只能向右或向左移动。如果这两个位置是空的，它随机选择一个。如果一个CrabCritter不能移动，那么它旋转90度，随机向左或向右移动。



你知道吗?

Set 9

The source code for the CrabCritter class is reproduced at the end of this part of GridWorld.

1.CrabCritter为什么不覆盖processActors方法?
2.描述CrabCritter查找和吃其他actor的过程。它是否总是吃所有相邻的actor？请解释。
3.为什么CrabCritter类使用getLocationsInDirections方法?
4.如果CrabCritter在位置（3,4）和坐北朝南，getActors方法返回的actor可能在哪里?
5.CrabCritter和critter行为之间的异同?
6.CrabCritter怎么决定它旋转而不是移动呢?
7.为什么CrabCritter对象不会吃掉另一个CrabCritter?

练习
1.修改ChameleonCritter的processActors方法，修改效果为：如果要处理的actor列表是空的，ChameleonCritter的颜色会变暗（像一朵花）。

在下面的练习，你的第一个步骤应该是决定这五种方法 - getActors，processActors，getMoveLocations，selectMoveLocation和makeMove--应该被修改来达到期望的结果。

2.创建一个扩展ChameleonCritter的ChameleonKid类（在练习1中修改的）. ChameleonKid改变其颜色为他前方或后方的actor的颜色。如果在前方或后方都没有actor，则ChameleonKid像练习1的ChameleonCritter一样变暗。

3.创建一个扩展critter的RockHound类。RockHound 像 critter 一样获得 actor。它会从获得的 actor 中删除所有的岩石。RockHound像critter一样移动。

4.创建一个扩展critter的BlusterCritter类。BlusterCritter看着都在两步之内的所有邻居。（对于不靠近边缘的BlusterCritter，这包括24个位置）。它计算在这些位置的critter的数量。如果数量比C少，BlusterCritter的颜色变亮（颜色值增加）。如果数量比C多，BlusterCritter的颜色变深（色值减少）。在这里，c是一个值，指示critter的勇气。它应该在构造函数中进行设置。

5.创建一个扩展CrabCritter的QuickCrab类。A QuickCrab moves to one of the two locations, randomly selected, that are two spaces to its right or left, if that location and the intervening location are both empty. Otherwise, a QuickCrab moves like a CrabCritter.

6.创建一个扩展CrabCritter的KingCrab类。KingCrab让他处理的每个 actor 移动（远离 KingCrab）。如果actor不能移动，KingCrab将它从网格中删除。当KingCrab已完成处理的actor，它的像一个CrabCritter移动。

小组活动
组织的2-4的学生群体

1.指定：每个小组选择一个新critter，继承critter。必须详细描述了新的critter的属性和行为。
2.设计：根据你的选择确定为new critter所需要的变量和基本算法。
3.代码：编写用于实现new critter的代码。
4.测试：每组写测试用例来测试new critter和验证你们的实现符合你们的设想。

CrabCritter.Java

```
-----------------------------------------------
import info.gridworld.actor.Actor;
import info.gridworld.actor.Critter;
import info.gridworld.grid.Grid;
import info.gridworld.grid.Location;
import java.awt.Color;
import java.util.ArrayList;
/**
* A CrabCritter looks at a limited set of neighbors when it eats and moves.
* This class is not tested on the AP CS A and AB Exams.
*/
public class CrabCritter extends Critter
{
public CrabCritter()
{
setColor(Color.RED);
}
/**
* A crab gets the actors in the three locations immediately in front, to its
* front-right and to its front-left
* @return a list of actors occupying these locations
*/
public ArrayList<Actor> getActors()
{
ArrayList<Actor> actors = new ArrayList<Actor>();
int[] dirs ={ Location.AHEAD, Location.HALF_LEFT, Location.HALF_RIGHT };
for (Location loc : getLocationsInDirections(dirs))
{
Actor a = getGrid().get(loc);
if (a != null)
actors.add(a);
}
return actors;
}
/**
* @return list of empty locations immediately to the right and to the left
*/
public ArrayList<Location> getMoveLocations()
{
```

```
ArrayList<Location> locs = new ArrayList<Location>();
int[] dirs = { Location.LEFT, Location.RIGHT };
for (Location loc : getLocationsInDirections(dirs))
if (getGrid().get(loc) == null)
locs.add(loc);
return locs;
}
/**
* If the crab critter doesn't move, it randomly turns left or right.
*/
public void makeMove(Location loc)
{
if (loc.equals(getLocation()))
{
double r = Math.random();
int angle;
if (r < 0.5)
angle = Location.LEFT;
else
angle = Location.RIGHT;
setDirection(getDirection() + angle);
}
else
super.makeMove(loc);
}
/**
* Finds the valid adjacent locations of this critter in different
* directions.
* @param directions - an array of directions (which are relative to the
* current direction)
* @return a set of valid locations that are neighbors of the current
* location in the given directions
*/
public ArrayList<Location> getLocationsInDirections(int[] directions)
{
ArrayList<Location> locs = new ArrayList<Location>();
Grid gr = getGrid();
Location loc = getLocation();
for (int d : directions)
{
Location neighborLoc = loc.getAdjacentLocation(getDirection() + d);
if (gr.isValid(neighborLoc))
locs.add(neighborLoc);
}
return locs;
}
}
```

何家豪 发表： Aug 22, 2015

A QuickCrab moves to one of the two locations, randomly selected, that are two spaces to its right or left, if that location and the intervening location are both empty. Otherwise, a QuickCrab moves like a CrabCritter.

曾坤 发表： Aug 22, 2015

Tired..

邱永臣 发表：

Aug 22, 2015

蛮力之世界最强魔兽...