GridWorld 项目开发报告

第14小组
杨翼飞 李安吉 吕敬 刘浩然 王凡

1. GridWorld简介

gridworld案例研究提供了一个图形环境。其中视觉对象居住在一个二维网格中并相互作用。
该案例允许设计和创建**actor**对象，将它们添加到**grid**中，并确定参与者是否按照它们的规范进行行为。它提供了一个显示grid和actor的图形用户界面（**GUI**）。
此外，GUI还具有一个可以向网格中添加actor和在它们上调用方法的工具。

```
    Part 1: Provides experiments to observe the attributes and behavior
of the actors.

    Part 2: Defines Bug variations.

    Part 3: Explores the code that is needed to understand and create
actors.

    Part 4: Defines classes that extend the Critter class.

    Part 5: (CS AB only) Explains grid data structures.
```

2. 正文部分

2.1 part 1

2.1.1 Exercises

1.使用setDirection method并完成表格，并给出每个输入代表的方向

Degrees	Compass Direction
0	North
45	NorthEast
90	East
135	SouthEast
180	South
225	SouthWest
270	West
315	NorthWest
360	North

2.使用moveTo方法将Bug移动到不同的位置。你可以向哪个方向移动它？你能把它移多远？如果你将bug移出网格会发生什么？

```
>可以使用moveTo方法将bug移动到任何有效位置。

>当使用moveTo方法移动bug时，bug不会改变它原来的方向。

>必须使用setDirection方法或turn方法来改变bug的方向

>尝试将bug移动出grid外时，将会造成IllegalArgumentException
```

3.用什么方法可以改变bug，花和石头的颜色？

```
setColor方法
```

4.将石头移动到bug上，在将石头移开，会发生什么？

```
>当一块石头移动到bug上时，bug将会消失。

>只剩下石头，再将石头移动到其他位置时，bug就不在那里了

>在网格的任意地方将一个actor移动到另一个actor的位置时，原位置的actor就会消失
```

2.2 Part 2

2.2.1 Exercises

2.2.1.1 CircleBug

1.编写一个与BoxBug相同的类CircleBug，在act方法中调用一次而不是两次turn方法。它的行为与BoxBug有什么不同？

代码传送门： CircleBug.java

CircleBugRunner是一个运行容器可以生成有circlebug的grid

CircleBugRunner.java

运行截图

代码片：

Class CircleBug

```
public class CircleBug extends Bug {
    private int steps;
    private final int sideLength;

    /**
     * Constructs a box bug that traces a square of a given side length
```

```
     * @param length the side length
     */
    public CircleBug(int length) {
        steps = 0;
        sideLength = length;
    }

    /**
     * Moves to the next location of the square.
     * Except that in the <code>act</code> method the <code>turn</code>
method
     * is called once instead of twice.
     */
    public void act() {
        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else {
            turn();
            steps = 0;
        }
    }
}
```

CicleBug 的路径是一个八边形而不是一个正方形

2.2.1.2 SpiralBug

2.模仿BoxBug写一个SpiralBug使Bug延螺旋形状前行，当Bug转动时调整边长。
代码传送门： SpiralBug.java
SpiralBugRunner可以生成SpiralBug的网格世界
SpiralBugRunner.java

运行截图

代码片：
Class SpiralBug

```
public class SpiralBug extends Bug {
    private int steps;
    private int sideLength;

    /**
     * Constructs a spiraled bug
     * @param length the side length
     */
```

```
    public SpiralBug(int length) {
        steps = 0;
        sideLength = length;
    }

    /**
     * Moves to the next location of the square.
     * <p>
     * Adjust the side length when the bug turns so that the
     * bug can drop flowers in a spiral pattern
     */
    public void act() {
        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else {
            turn();
            turn();
            steps = 0;
            sideLength++;
        }
    }
}
```

2.2.1.3 ZBug

3.编写一个ZBug，让bug延“ **Z** ”字移动，从左上角开始，完成一个Z字型路径后停止移动，在构造函数中提供**Z**的参数。

notice:ZBug运行时，Bug必须面向东（→）

代码传送门：

ZBug.java

ZBugRunner可以生成ZBug的网格世界

ZBugRunner.java

运行截图

代码片：

Class ZBug

```
public class ZBug extends Bug {
    private int steps;
    private final int sideLength;
```

```
    private boolean flag;

    /**
     * Constructs a Z bug that traces a "Z" of a given side length
     * @param length the side length
     */
    public ZBug(int length) {
        steps = 0;
        sideLength = length;
        this.setDirection(90);
        flag = false;
    }

    /**
     * Moves to the next location of the "Z".
     */
    public void act() {

        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else if (steps == sideLength) {
            if (flag) {
                return;
            } else if (this.getDirection() == 90) {
                this.setDirection(225);
            } else if (this.getDirection() == 225) {
                flag = true;
                this.setDirection(90);
            }
            steps = 0;
        }
    }
}
```

2.2.1.4 DancingBug

4.写一个DancingBug类，通过在每次移动前朝不同方向转向实现“**dancing**”

它的构造函数有一个整数数组作为参数，数组中的整数表示Bug在每次移动前turn的次数

每次turn默认为顺时针45 degrees

每次Bug移动前都会按照数组中的条目转动角度，移动后将会继续按照下一个条目转动角度

当执行完最后一次转弯后将会以初始数组值继续移动，使Bug不断重复相同的Dance移动

DancingBugRunner类会创建这一数组，并将其作为一个参数传递给DancingBug的构造函数

代码传送门：

DancingBug.java

DanceBugRunner可以生成DancingBug的网格世界

DancingBugRunner.java

运行截图

代码片：

Class DancingBug

```
public class DancingBug extends Bug {
    // the array of the number of turns when acting
    private final int[] turnArray;
    private int steps;
    private final int sideLength;
    // the times that the bug has acted.
    private int turnIndex;

    /**
     * Constructs a box bug that traces a square of a given side
length.
     * @param length the side length
     * @param turns  the array of the number of turns when acting
     */
    public DancingBug(int[] turns, int length) {
        steps = 0;
        sideLength = length;
        turnArray = turns;
        turnIndex = 0;
    }

    /**
     * Constructs a box bug that traces a square of a given side
length.
     * <p>
     * In this function, the length = 1.
     * @param turns the array of the number of turns when acting
     */
    public DancingBug(int[] turns) {
        this(turns, 1);
    }

    /**
```

```
     * Moves to the next location of the square.
     */
    public void act() {
        if (steps < sideLength && canMove()) {
            move();
            steps++;
        } else {
            for (int i = 0; i < turnArray[turnIndex]; i++) {
                turn();
            }
            turnIndex = (turnIndex + 1) % turnArray.length;
            steps = 0;
        }
    }
}
```

2.2.1.5 BugRunner总结

5.学习BugRunner类的代码，总结向网格世界中添加如BoxBug对象的方法。

```
>创建一个BoxBug对象

    BoxBug bbug = new BoxBug(2);

>将新建的BoxBug对象添加到网格世界的指定位置中

    world.add(new Location(5,5) , bbug);
```

2.3 Part 3

2.3.1 Group Activity

- 小组完成一个名为 Jumper 的类， 它可以让actor每次移动向前移动两个单元格。当遇到岩石和花时可以跳过，跳跃时不会留下任何东西

- 小组讨论并解决了如下问题
  - 如果Jumper的前一格为空，但是前两格位置有花或者石头。
    - 顺时针转 45 度
  - 如果Jumper的前两格位置不在Grid中。
    - 顺时针转 45 度
  - 如果Jumper面对Grid边缘。
    - 顺时针转 45 度
  - 如果Jumper的前两格位置处有两一个actor。

        - 部分实例中会移除原位置的actor
    - 如果Jumper在路径上遭遇另一个jumper。
        - 部分实例中会移除一个Jumper
    - 其他测试

- 对**Jumper**类设计决策
    - Jumper应该继承哪个类
        - Jumper定义为一种新的Bug，所以可能会继承Bug类
    - 是否有和Jumper相似的类
        - Bug类与Jumper类相似，有很多相近的methods
    - 是否需要构造函数，具体需要哪些参数
        - 如果有需要将多个Jumper放入一个Grid，为了便于区分可以在构造函数中添加颜色参数
    - 哪些 methods 需要重写
        - act 为了让Jumper的行为和Actor不同
    - 可能会需要添加哪些新的 methods
        - 类似于Bug中的move和canmove函数，Jumper中需要写新的Jump和Jumper函数
    - 如何测试该类
        - 按照上文中讨论的问题测试

代码传送

Jumper.java

JumperRunner可以生成Jumper的网格世界

JumperRunner.java

JumperTest是对Jumper类的测试代码

JumperTest.java

Jumper代码片

```
import java.awt.Color;

public class Jumper extends Bug {
    public Jumper() {
        setColor(Color.BLUE);
    }
```

```
    //    带颜色的声明方式
    public Jumper(Color JumperColor) {
        setColor(JumperColor);
    }

    //     一开始理解错了，其实它只要跳就行，不用管能不能前进一格
    @Override
    public boolean canMove() {
        Grid<Actor> gr = getGrid();
        if (gr == null) {
            return false;
        }
//      获取当前位置
        Location loc = getLocation();
//      移动一次的位置
        Location moveNext = loc.getAdjacentLocation(getDirection());
//      出界判断
        if (!gr.isValid(moveNext)) {
            return false;
        }
//      跳跃一次的位置
        Location jumpNext =
moveNext.getAdjacentLocation(getDirection());
        if (!gr.isValid(jumpNext)) {
            return false;
        }
//      判断移动的位置有没有可覆盖的Actor
        Actor moveNeighbor = gr.get(moveNext);
        Actor jumpNeighbor = gr.get(jumpNext);
//      可以跳过花和石头
        boolean jump = moveNeighbor == null || moveNeighbor instanceof
Flower || moveNeighbor instanceof Rock;
//      为了可玩性强一点我还是设定可以覆盖花朵吧
        boolean move = jumpNeighbor == null || jumpNeighbor instanceof
Flower;
        return move && jump;

    }

    @Override
    public void act() {
        if (canMove()) {
            move();
        } else {
            turn();
        }
    }
```

```
    @Override
    public void move() {
        Grid<Actor> gr = getGrid();
        if (gr == null) {
            return;
        }

        Location loc = getLocation();
        Location move = loc.getAdjacentLocation(getDirection());
        Location jump = move.getAdjacentLocation(getDirection());
//      卡一下条件
        if (gr.isValid(move) && gr.isValid(jump)) {
            Actor jumpNeighbor = gr.get(jump);
            if (jumpNeighbor != null) {
                jumpNeighbor.removeSelfFromGrid();
            }
            moveTo(jump);
        }
//        else{
//            removeSelfFromGrid();
//        }
//      留一朵花花，但是好像也没必要
        Flower flw = new Flower(getColor());
        flw.putSelfInGrid(gr, loc);
    }

    //  每次转45度
    @Override
    public void turn() {
        setDirection(getDirection() + Location.HALF_RIGHT);
    }

}
```

测试代码片（**JumperTest**）

```
import static org.junit.Assert.assertEquals;

public class JumperTest {
    private int number = 8;
    private Jumper[] jmps = new Jumper[number];//用来测试的各种Jumper

    @Before
    public void setUp() {
        ActorWorld world = new ActorWorld();
```

```
        for (int i = 0; i < number; i++) {
            jmps[i] = new Jumper();
        }
//      基本设置
        world.add(new Location(4, 1), new Flower());
        world.add(new Location(3, 6), new Flower());
        world.add(new Location(4, 2), new Rock());
        world.add(new Location(4, 3), new Bug());

//      放置jumper
        world.add(new Location(1, 0), jmps[0]); // 测试距离边界只有一格的
情况
        world.add(new Location(0, 1), jmps[1]); // 测试在边界时的情况
        world.add(new Location(5, 0), jmps[2]); // 前方第二格为空，第一格
为空
        world.add(new Location(5, 1), jmps[3]); // 前方第二格为空，第一格
为flower
        world.add(new Location(5, 2), jmps[4]); // 前方第二格为空，第一格
为rock
        world.add(new Location(5, 3), jmps[5]); // 前方第二格为空，第一格
为bug
        world.add(new Location(5, 6), jmps[6]); // 测试前方第二格有flower
时的情况
        world.add(new Location(7, 6), jmps[7]); // 测试前方第二格有其他
actor时的情况
        world.show();
    }

    //  下面是测试各种jumper能不能移动的情况
    @Test
    public void testJumper0() {
        assertEquals(jmps[0].canMove(), false);
    }

    @Test
    public void testJumper1() {
        assertEquals(jmps[1].canMove(), false);
    }

    @Test
    public void testJumper2() {
        assertEquals(jmps[2].canMove(), true);
    }

    @Test
    public void testJumper3() {
        assertEquals(jmps[3].canMove(), true);
    }
```

```
    @Test
    public void testJumper4() {
        assertEquals(jmps[4].canMove(), true);
    }

    @Test
    public void testJumper5() {
        assertEquals(jmps[5].canMove(), false);
    }

    @Test
    public void testJumper6() {
        assertEquals(jmps[6].canMove(), true);
    }

    @Test
    public void testJumper7() {
        assertEquals(jmps[7].canMove(), false);
    }
}
```

2.4 Part 4

2.4.1 Exercises

2.4.1.1 processActors

1.完善ChameleonCritter类中的processActors方法，使要处理的actors列表为空的话，ChameleonCritter的颜色会想flower一样变暗。

代码传送门： ChameleonCritter.java

ChameleonRunner是一个运行容器可以生成有chameleon critters的grid

ChameleonRunner.java

代码片：

Method processActors

```
public void processActors(ArrayList<Actor> actors) {
        int n = actors.size();
        // if the list of actors to process is empty
        // the color while darken.
        if (n == 0) {
            // same code as in the Flower class
```

```
            Color c = getColor();
            int red = (int) (c.getRed() * (1 - DARKENING_FACTOR));
            int green = (int) (c.getGreen() * (1 - DARKENING_FACTOR));
            int blue = (int) (c.getBlue() * (1 - DARKENING_FACTOR));
            setColor(new Color(red, green, blue));
            return;
        }
        int r = (int) (Math.random() * n);

        Actor other = actors.get(r);
        setColor(other.getColor());
    }
```

2.4.1.2 ChameleonKid

2.编写一个以ChameleonCritter为基类的ChameleonKid类来扩展上一题里的ChemeleonCritter类。ChameleonKid把它的颜色变为前面或后面一个actors的颜色。如果这两处都没有actors，那么ChameleonKid就会像ChameleonCritter一样变暗。

代码传送门： ChameleonKid.java

ChameleonKidRunner可以生成ChameleonKid的网格世界

ChameleonKidRunner.java

代码片：

Class ChameleonKid

```
public class ChameleonKid extends ChameleonCritter {

    /**
     * @return the actors that located in the front or behind of the
critter.
     */
    @Override
    public ArrayList<Actor> getActors() {
        ArrayList<Actor> res = new ArrayList<>();
        ArrayList<Actor> neighbors =
getGrid().getNeighbors(getLocation());
        int dir = getDirection();
        Location front = getLocation().getAdjacentLocation(dir);
        Location back = getLocation().getAdjacentLocation(dir + 180);
        for (Actor a : neighbors) {
            if (a.getLocation().equals(front) ||
a.getLocation().equals(back)) {
                res.add(a);
            }
        }
```

```
        return res;
    }

}
```

2.4.1.3 RockHound

3.编写一个Critter的扩展类RockHound。RockHound让actors以像Critter一样的方式被处理。它将从grid中删除该列表中的所有Rocks。RockHound像Critter一样移动。

代码传送门：

RockHound.java

RockHoundRunner可以生成RockHound的网格世界

RockHoundRunner.java

代码片：

Class RockHound

```
public class RockHound extends Critter {
    /**
     * The method to process the actors. Remove the rocks in the list.
     * @param actors the list of actors to be processed.
     */
    @Override
    public void processActors(ArrayList<Actor> actors) {
        // remove actors that are rocks
        for (Actor a : actors) {
            if (a instanceof Rock) {
                a.removeSelfFromGrid();
            }
        }
    }
}
```

2.4.1.4 DancingBug

4.创建一个Critter的扩展类BlusterCritter。一个BlusterCritter看向它当前位置两步内的所有地点。（对于不靠近网格边缘的BlusterCritter，看向的地点有24个）。它计算了这些地点的Critters数量。如果少于c，BlusterCritter的颜色会变得更亮（颜色值增加）。如果有c或更多的动物，蓝色动物的颜色会变暗（颜色值会减少）。其中，c是一个表示小动物的courage

的值，应该在构造函数中被设置。

代码传送门：

BlusterCritter.java

BlusterRunner可以生成DancingBug的网格世界

BlusterRunner.java

代码片：

Class BlusterCritter

```
public class BlusterCritter extends Critter {

    static final double DARKENING_FACTOR = 0.05;
    private final int courage;

    public BlusterCritter(int c) {
        super();
        courage = c;
    }

    /**
     * @return the actors within two steps of its current location.
     */
    @Override
    public ArrayList<Actor> getActors() {
        ArrayList<Actor> actors = new ArrayList<>();
        Location loc = getLocation();
        for (int r = loc.getRow() - 2; r < loc.getRow() + 3; r++) {
            for (int c = loc.getCol() - 2; c < loc.getCol() + 3; c++) {
                Location tmpLoc = new Location(r, c);
                if (getGrid().isValid(tmpLoc)) {
                    Actor a = getGrid().get(tmpLoc);
                    if (a != null && a != this) {
                        actors.add(a);
                    }
                }
            }
        }
        return actors;
    }

    /**
     * The method to process the actors.
     * <p> if the number of actors is more than c, bright the color,
and if otherwise, darken it.</p>
     *
```

```
     * @param actors the actors to be processed
     */
    @Override
    public void processActors(ArrayList<Actor> actors) {
        int n = actors.size();
        if (n < courage) {
            brighten();
        } else {
            darken();
        }
    }

    /**
     * brighten the color of the critter
     */
    private void brighten() {
        Color c = getColor();
        int red = (int) (c.getRed() * (1 + DARKENING_FACTOR) > 255 ?
255 : c.getRed() * (1 + DARKENING_FACTOR));
        int green = (int) (c.getGreen() * (1 + DARKENING_FACTOR) > 255
? 255 : c.getGreen() * (1 + DARKENING_FACTOR));
        int blue = (int) (c.getBlue() * (1 + DARKENING_FACTOR) > 255 ?
255 : c.getBlue() * (1 + DARKENING_FACTOR));
        setColor(new Color(red, green, blue));
    }


    /**
     * Repeat the codes. I want to make a method to do this.
     * <p> darken the critter </p>
     */
    private void darken() {
        Color c = getColor();
        int red = (int) (c.getRed() * (1 - DARKENING_FACTOR));
        int green = (int) (c.getGreen() * (1 - DARKENING_FACTOR));
        int blue = (int) (c.getBlue() * (1 - DARKENING_FACTOR));
        setColor(new Color(red, green, blue));
    }
}
```

2.4.1.5 QuickCrab

5.创建一个CrabCritter的扩展类QuickCrab。一个QuickCrab处理行为的方式和CrabCritter一样。如果QuickCrab的左右两格都是空的，则QuickCrab随机移动到其中一格。否则，QuickCrab就像CrabCritter一样移动。

代码传送门： QuickCrab.java

QuickCrabRunner可以生成QuickCrab的网格世界

QuickCrabRunner.java

代码片：

Class QuickCrab

```
public class QuickCrab extends CrabCritter {
    /**
     * @return the locations to move to.
     */
    @Override
    public ArrayList<Location> getMoveLocations() {
        ArrayList<Location> locs = new ArrayList<>();
        int[] dirs = {Location.LEFT, Location.RIGHT};
        Grid<Actor> gr = getGrid();

        for (int dir : dirs) {
            Location loc =
getLocation().getAdjacentLocation(getDirection() + dir);
            if (gr.isValid(loc) && gr.get(loc) == null) {
                Location next = loc.getAdjacentLocation(getDirection()
+ dir);
                if (gr.isValid(next) && gr.get(next) == null) {
                    locs.add(next);
                }
            }
        }

        return locs;
    }
}
```

2.4.1.6 KingCrab

6.创建一个CrabCritter的扩展类KingCrab。KingCrab让actors像CrabCritter一样处理actors。KingCrab使每个actor的一个位置移动到远离KingCrab的地方。如果actor不能离开，KingCrab就会把它从网格中移除。当KingCrab完成了对actors的处理后，它就像CrabCritter一样移动。

代码传送门：

KingCrab.java

KingCrabRunner可以生成KingCrab的网格世界

KingCrabRunner.java

代码片：

Class KingCrab

```
public class KingCrab extends CrabCritter {
    /**
     * the method to process the actors. <br />
     *
     * @param actors the actors to be processed
     */
    @Override
    public void processActors(ArrayList<Actor> actors) {
        Grid<Actor> gr = getGrid();
        Location loc = getLocation();
        for (Actor a : actors) {
            Location aLoc = a.getLocation();
            int dir = loc.getDirectionToward(aLoc);
            Location next = aLoc.getAdjacentLocation(dir);
            if (gr.isValid(next) && gr.get(next) == null) {
                a.moveTo(next);
            } else {
                a.removeSelfFromGrid();
            }
        }
    }
}
```

2.5 Part 5

2.5.1 Exercises

2.5.1.1 SparseBoundedGrid

1.假设一个程序需要一个非常大的有界网格，它包含很少的对象，并且程序经常调用getOccupiedLocations方法（例如，ActorWorld）。创建一个使用“稀疏数组”实现的SparseBoundedGrid类。您的解决方案不需要是一个通用类；您可以只需存储Object类型的使用者。

“稀疏数组”是一个链接表的数组列表。每个链表条目同时包含一个网格占用者和一个列索引。数组列表中的每个条目都是一个链接列表，但如果该行为空，则为null。

代码传送门：

SparseBoundedGrid.java

SparseBoundedGrid2.java

SparseBoundedGrid3.java

SparseBoundedGridRunner运行时你可以选择上述三种不同实现方法的网格类

SparseBoundedGridRunner.java

代码片：

Class SparseBoundedGrid3

```
public class SparseBoundedGrid3<E> extends AbstractGrid<E> {
    private TreeMap<Location, E> occupantMap;
    private final int rows;
    private final int cols;

    public SparseBoundedGrid3(int r, int c) {
        if (r <= 0)
            throw new IllegalArgumentException("rows <= 0");
        if (c <= 0)
            throw new IllegalArgumentException("cols <= 0");
        rows = r;
        cols = c;
        occupantMap = new TreeMap<>();
    }

    @Override
    public int getNumRows() {
        return rows;
    }

    @Override
    public int getNumCols() {
        return cols;
    }

    @Override
    public boolean isValid(Location loc) {
        return (0 <= loc.getRow()) && (loc.getRow() < getNumRows())
                && (0 <= loc.getCol()) && (loc.getCol() <
getNumCols());
    }

    @Override
    public ArrayList<Location> getOccupiedLocations() {
        return new ArrayList<>(occupantMap.keySet());
    }

    @Override
    public E get(Location loc) {
        if (loc == null)
            throw new NullPointerException("loc == null");
        return occupantMap.get(loc);
```

```
    }

    @Override
    public E put(Location loc, E obj) {
        if (loc == null)
            throw new NullPointerException("loc == null");
        if (obj == null)
            throw new NullPointerException("obj == null");
        return occupantMap.put(loc, obj);
    }

    @Override
    public E remove(Location loc) {
        if (loc == null)
            throw new NullPointerException("loc == null");
        return occupantMap.remove(loc);
    }
}
```

2.5.1.2 UnboundedGrid

2.考虑使用HashMap或TreeMap来实现SparseBoundedGrid。如何使用UnboundedGrid类来完成此任务？哪些UnboundedGrid的方法可以使用而不改变？填写下面的图表来比较SparseBoundedGrid的每个实现的预期Big-oh效率。

代码传送门：

UnboundedGrid2.java

代码片：

Class UnboundedGrid2

```
public class UnboundedGrid2<E> extends AbstractGrid<E> {
    private Object[][] occupantArray;
    private final int size = 16;

    public UnboundedGrid2() {
        occupantArray = new Object[size][size];
    }

    @Override
    public int getNumRows() {
        return -1;
    }

    @Override
    public int getNumCols() {
        return -1;
```

```
    }

    @Override
    public boolean isValid(Location loc) {
        return loc != null;
    }

    @Override
    public ArrayList<Location> getOccupiedLocations() {
        ArrayList<Location> ans = new ArrayList<>();

        for (int i = 0; i < occupantArray.length; i++) {
            for (int j = 0; j < occupantArray[i].length; j++) {
                Location loc = new Location(i, j);
                if (get(loc) != null) {
                    ans.add(loc);
                }
            }
        }

        return ans;
    }

    @Override
    public E get(Location loc) {
        if (!isValid(loc)) {
            throw new IllegalArgumentException("Location " + loc + " is
not valid");
        }

        if (loc.getRow() >= occupantArray.length || loc.getCol() >=
occupantArray[0].length) {
            return null;
        }

        return (E) occupantArray[loc.getRow()][loc.getCol()];
    }

    @Override
    public E put(Location loc, E obj) {
        if (!isValid(loc)) {
            throw new IllegalArgumentException("Location " + loc + " is
not valid");
        }

        if (obj == null) {
            throw new NullPointerException("obj == null");
        }
```

```
        if (loc.getRow() >= occupantArray.length || loc.getCol() >=
occupantArray[0].length) {
            changeSize(loc);
        }

        E old = get(loc);
        occupantArray[loc.getRow()][loc.getCol()] = obj;
        return old;
    }

    private void changeSize(Location loc){
        int pSize = Math.max(loc.getRow(), loc.getCol()) + 1;
        // double both array bounds until they are large enough
        int newSize = size;
        while (newSize < pSize) {
            newSize <<= 1;
        }

        Object[][] newArray = new Object[newSize][newSize];

        // There seems to be a problem with the way it is written.
//        System.arraycopy(occupantArray, 0, newArray, 0,
occupantArray.length);

        for (int i = 0; i < newSize; i++) {
            System.arraycopy(occupantArray[i], 0, newArray[i], 0,
newSize);
        }

        occupantArray = newArray;
    }

    @Override
    public E remove(Location loc) {
        if (!isValid(loc)) {
            throw new IllegalArgumentException("Location " + loc + " is
not valid");
        }

        if (loc.getRow() >= occupantArray.length || loc.getCol() >=
occupantArray[0].length) {
            return null;
        }

        E old = get(loc);
        occupantArray[loc.getRow()][loc.getCol()] = null;
        return old;
```

```
        }
    }
```

Methods	SparseGridNode Version	LinkedList Version	HashMap Version	TreeMap Version
getNeighbors	O(c)	O(c)	O(1)	O(logn)
getEmptyAdjacentLocations	O(c)	O(c)	O(1)	O(logn)
getOccupiedAdjacentLocations	O(c)	O(c)	O(1)	O(logn)
getOccupiedLocations	O(c+n)	O(r+n)	O(n)	O(n)
get	O(c)	O(c)	O(1)	O(logn)
put	O(c)	O(c)	O(1)	O(logn)
remove	O(c)	O(c)	O(1)	O(logn)

3.考虑一个无界网格的实现，其中所有有效的位置都有非负的行值和列值。构造函数分配一个16 x 16的数组。当调用具有当前数组边界之外的行或列索引的put方法时，加倍两个数组边界，直到它们足够大，用这些边界构造一个新的正方形数组，并将现有的使用者放置到新的数组中。使用此数据结构实现网格接口指定的方法。get方法的Big-oh效率是多少？当行和列索引值在当前数组范围内时，put方法的效率如何？需要调整阵列大小时的效率如何？

- get方法的Big-Oh效率是O(1)
- 行列索引值在当前数组范围内时，put方法的效率是O(1)
- 当需要调整阵列大小时，put方法的效率是O(n^2)，n是数组size。

2.6 MazeBug：深度优先算法实现

2.6.1 任务说明

本实验要求使用改进的Grid World软装置中实现深度优先算法，从而使虫子走出迷宫

2.6.2 深度优先算法基本步骤

将迷宫中所有可到达的位置记为一个节点，完成以下步骤：

```
1．将所有树的节点标记为“未访问”状态
2．输出起始节点，将起始节点标记为”已访问”状态。
3．将起始节点入栈。
4．当栈非空时重复执行以下步骤：
    a.取当前栈顶节点。
    b．如果当前栈顶节点是结束节点（迷宫出口），输出该节点，结束搜索。
    c．如果当前栈顶节点存在”未访问”状态的邻接节点，则选择一个未访问节点，置为”已访问”状态，并将它入栈，继续步骤a。
    d．如果当前栈顶节点不存在”未访问”状态的邻接节点，则将栈顶节点出栈，继续步骤a。
```

2.6.3 算法实现

补充软装置中的act()、canMove()等函数，实现虫子走迷宫的深度优先算法，同时注意虫子在有多个方向可以选择时，使用随机算法选择下一步位置。并在此基础上增加方向的概率估计，当向某个方向的移动次数较多时，该方向被随机选择的概率更大。具体代码可见 MazeBug.

MazeBug类中各方法实现的功能说明如下：

- `act()` :当MazeBug可以继续用移动时，移动到下个位置，并增加相应方向的权重，否则结束移动。
- `getValid()` :判断当前位置下四个方向可移动到下一步位置。
- `directionPrediction()` ：对四个方向进行有权重的随机选择。
- `canMove()` 判断栈中是否还有可移动节点
- `move()` ：继承Bug类的移动方法。

2.6.4 装置启动

运行`MazeBugRunner.java`后，点击菜单Map—loadMap,选择`MazeBug`文件夹下的地图文件，即可加载地图。点击`Run`，小虫会自动使用深度优先算法走出迷宫。

不同地图下的运行结果可见MazeBug/result

2.7 N-Puzzle：广度优先搜索和A*算法实现

2.7.1 任务目标

1. 使用广度优先算法求出8-数码问题的最优解
2. 利用启发式搜索算法求解随机生成的24-数码问题

2.7.2 算法基本思想

和DFS相反，BFS算法会尽可能”广”地搜索每一个节点的邻接点，能够找到从源结点到目标结点的最短路径，因而本实验采用它来求8-数码问题的最优解。

算法步骤如下：

```
1.将起始节点放入一个open列表中。
2.如果open列表为空，则搜索失败，问题无解；否则重复以下步骤：
    a．访问open列表中的第一个节点v，若v为目标节点，则搜索成功，退出。
    b．从open列表中删除节点v，放入close列表中。
    c．将所有与v邻接且未曾被访问的节点放入open列表中。
```

在盲目搜索的基础上，A*算法利用问题已有的信息进行搜索，动态确定搜索节点数顺序，达到降低搜索范围的目的。

N-数码问题中，每搜索到每一个节点时，通过”估价函数”对该节点进行”评估”，然后优先访问”最优良”节点的邻接节点，能够大大减少求解的时间。

计算估价函数的方法有多种，例如：

1. 所有 放错位的数码 个数
2. 后续节点不正确的个数
3. 正确节点与不正确节点的曼哈顿距离/欧拉距离
4. ...

本次我们将综合几种估价方法计算权重。

A*算法步骤如下：

```
1.将起始节点放入一个列表中。
2.如果列表为空，则搜索失败，问题无解；否则重复以下步骤：
    a．访问列表中的第一个节点v，若v为目标节点，则搜索成功，退出。
    b．从列表中删除节点v。
    c．利用估价函数，对所有与v邻接且未曾被发现的节点进行估价，按照估价大小（小的在前）插入列表中。
```

2.7.3 算法实现

本次实验我们使用Jigsaw软装置，将拼图抽象为`JigsawNode`类，求解拼图的过程在`Jigsaw`类和`Solution`类完成。

文件的设置如下：

属性	介绍
JigsawNode	拼图的数据结构
Jigsaw	搜索算法基类
Solution	实现Jigsaw抽象方法
Runners*	演示脚本
main	测试脚本

JigsawNode类是拼图的数据结构，包含节点状态和节点操作这两个重要的元素，在3.1节已作介绍，5.1节有详细说明。

Jigsaw类则是完成搜索的地方，其中存储了拼图的初始状态、目标状态以及当前状态，以及与拼图游戏相关的其他数据和方法。演示脚本RunnerDemo.java使用了ASearch(JigsawNode bNode, JigsawNode eNode)求解随机8-数码问题（3*3拼图）；实验任务一要求在Solution类的BFSearch(JigsawNode bNode, JigsawNode eNode)中修改广度优先搜索算法；实验任务二要求修改estimateValue(JigsawNode jNode)方法，完成用启发式搜索求解24-数码问题（5×5拼图）。

Runners目录中包含了1个基本的演示脚本和2个实验任务演示脚本。

main脚本评判求解效率。

具体代码可见 jiasaw,测试结果可见jiasaw/result