Stage 2 - Part 5 Report

19335286 郑有为

Stage 2 - Part 5 Report
1 - SparseBoundedGrid 稀疏阵列实现
2 - SparseBoundedGrid 哈希表实现
3 - DynamicUnboundedGrid 动态分配

1 - SparseBoundedGrid 稀疏阵列实现

- **类的说明**：“稀疏数组”是一个链表的数组列表。每个链表条目都包含一个网格居住者和一个列索引。数组列表中的每个条目都是一个链表，如果该行为空则为空。有两种方法实现链表，下面分别是他们的结节单元结构：

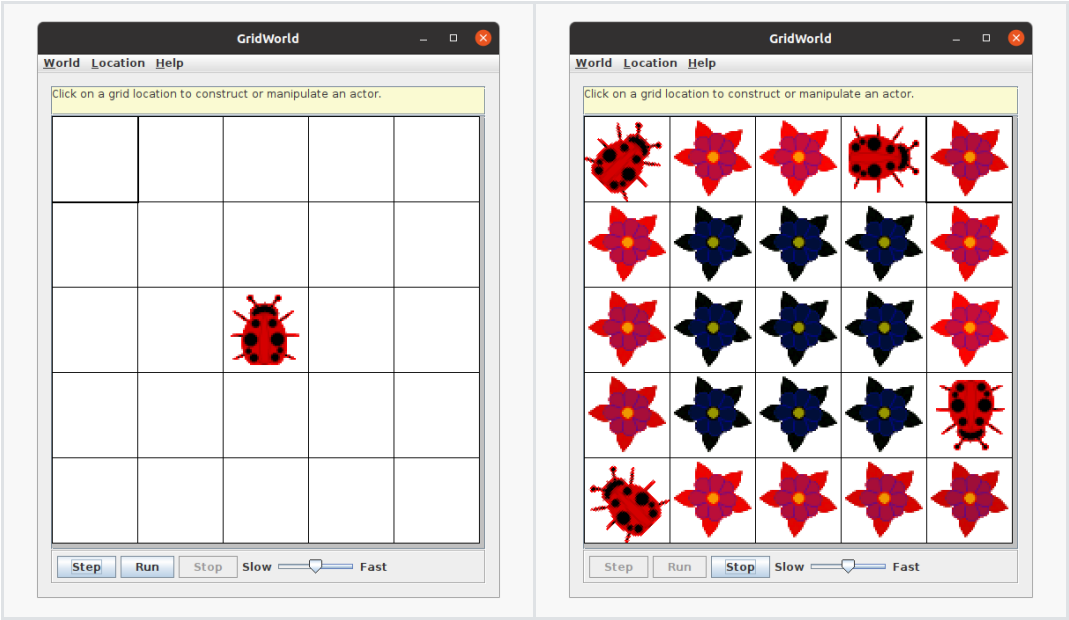
```
public class SparseGridNode
{
    private Object occupant;
    private int col;
    private SparseGridNode next;
    ... ...
}
```

```
// 使用 LinkedList
public class OccupantInCol
{
    private Object occupant;
    private int col;
    ... ...
}
```

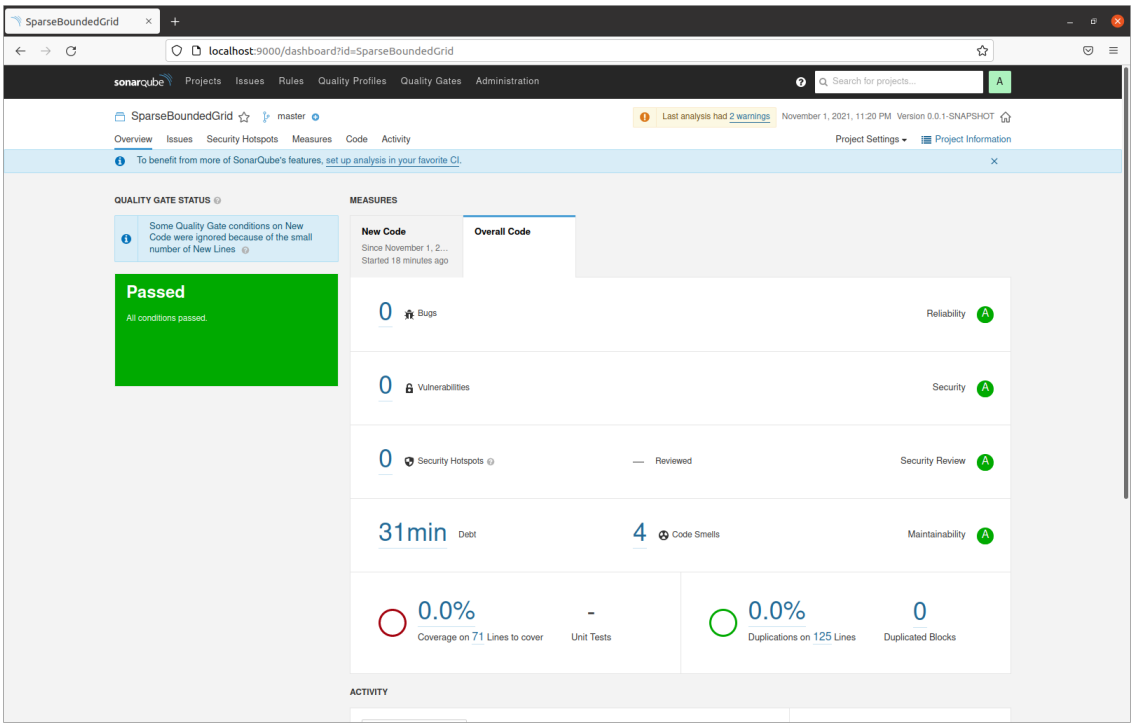
我们选择第一种进行实现，参考 `SparseGridNode` 类，由私有属性和它们的 `get` 和 `set` 方法组成。

对于一个有 r 行 c 列的网格，稀疏数组的长度为 r ，每个链表的最大长度为 c 。使用稀疏矩阵的时间复杂度更低，主要体现在 `getOccupiedLocations` 方法上。使用稀疏矩阵的时间复杂度为 $O(n+r)$，其中r是遍历外层数组的耗时，而使用普通矩阵的复杂度为$O(r*c)$，在网格比较稀疏的情况下，前者的复杂度低于后者。

- **实现说明**
  - `SparseGridNode` 类：由私有属性和它们的 `get` 和 `set` 方法组成；
  - `SparseBoundedGrid` 类：使用 `SparseGridNode[]` 来储存每一行的首个Occupant，提供 `isValid`、`get`、`set`、`remove`、`getOccupiedLocations` 五个方法。加入链表从链表头部插入，复杂度为$O(1)$，删除链表需要处理指针，维持链表结构。
- **运行结果**
  - 如下图：生成了一个5*5的网格，Bug在里面能够正常移动。



- **Sonar 测试结果**：Passed



2 - SparseBoundedGrid 哈希表实现

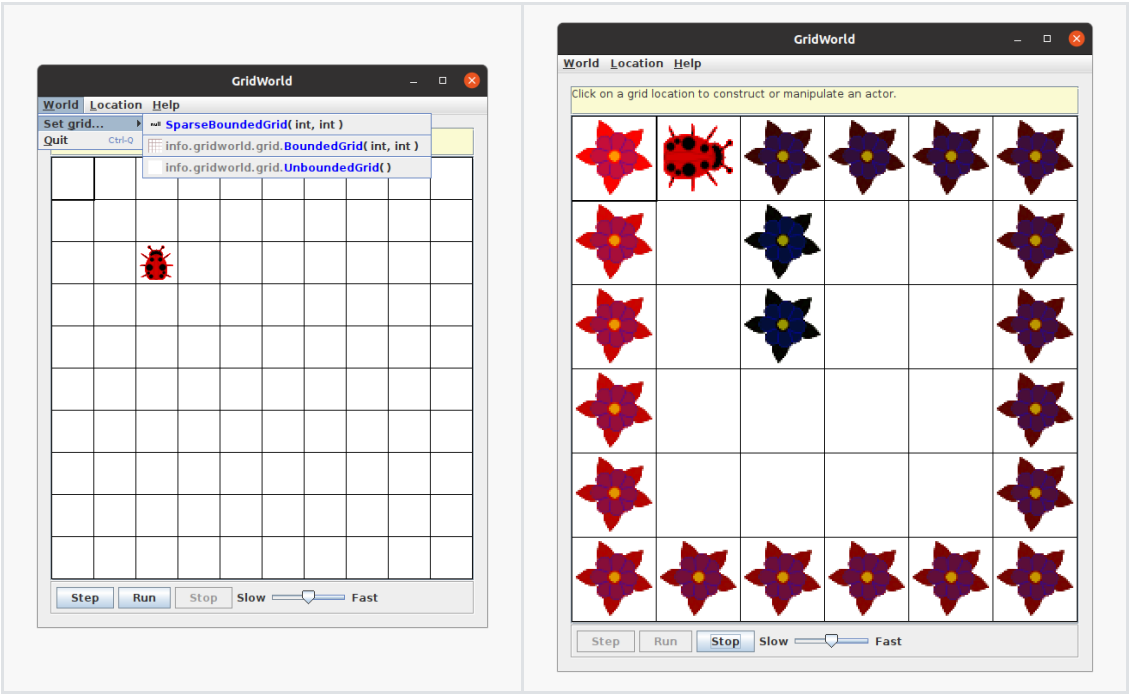
- **类的说明**：用哈希表同样能达到稀疏矩阵的效果，实现上类似于UnboundedGrid，只是对Location进行了约束。

  和UnboundedGrid类的实现一致的方法有：`getOccupiedLocations`，`get`，`put`，`remove`

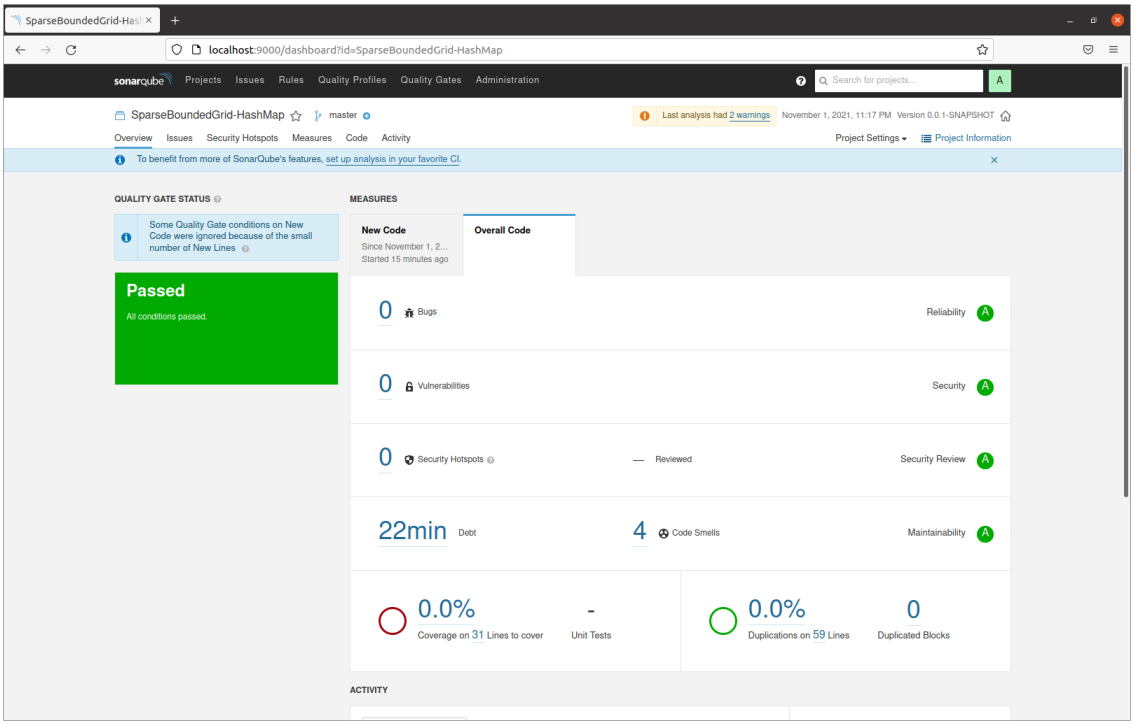
  以下是各种BoundGrid实现方式的复杂度汇总：设r = 行数，c = 列数，n = 非空位置总数

Methods	`SparseGridNode` version	`LinkedList<OccupantInCol>` version	`HashMap` version	`TreeMap` version
`getNeighbors`	$O(c)$	$O(c)$	$O(1)$	$O(\log n)$
`getEmptyAdjacentLocations`	$O(c)$	$O(c)$	$O(1)$	$O(\log n)$
`getOccupiedAdjacentLocations`	$O(c)$	$O(c)$	$O(1)$	$O(\log n)$
`getOccupiedLocations`	$O(r+n)$	$O(r+n)$	$O(n)$	$O(n)$
`get`	$O(c)$	$O(c)$	$O(1)$	$O(\log n)$
`put`	$O(c)$	$O(c)$	$O(1)$	$O(\log n)$
`remove`	$O(c)$	$O(c)$	$O(1)$	$O(\log n)$

- **实现说明**：与UnboundedGrid类似，使用一个哈希表 `Map<Location, E> occupantMap` 来记录。
- **运行结果**
  - 图1：右键World创建一个 SparseBoundGrid（6*6）；
  - 图2：Bug在 SparseBoundGrid 中正常运动。

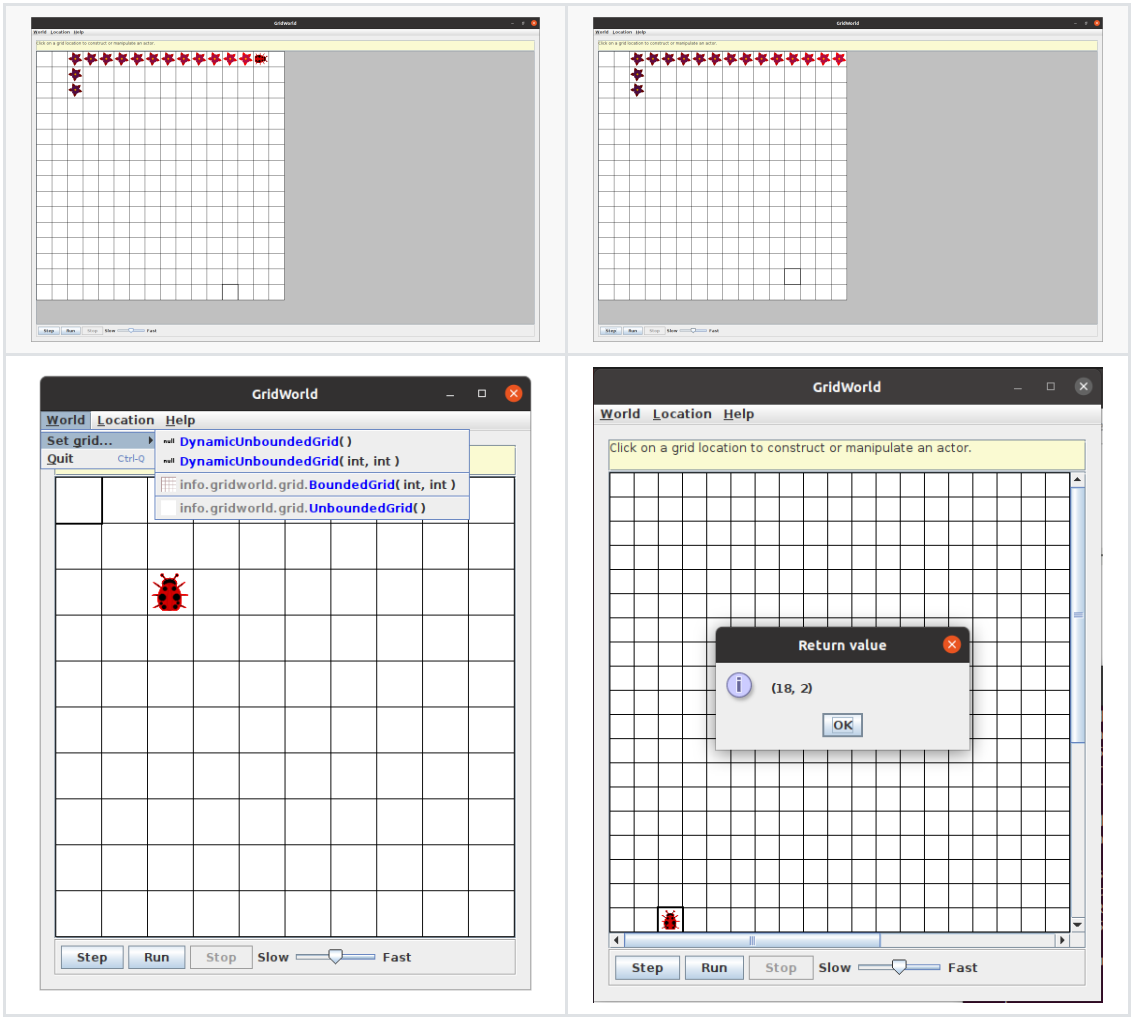


- **Sonar 测试结果**：Passed



3 - DynamicUnboundedGrid 动态分配

- **类的说明**
  - 考虑一个无界网格的实现，其中所有有效位置都具有非负的行和列值。构造函数分配一个16 x 16的数组。当使用超出当前数组边界的行或列索引调用put方法时，将两个数组边界加倍，直到它们足够大，使用这些边界构造一个新的正方形数组，并将现有的占位者放入新的数组中；
  - 复杂度分析：
    - `get`方法：$O(1)$
    - `put`方法：当不需要扩大网格时，复杂度为$O(1)$，当需要扩大网格时，复杂度为$O(r*c)$。
- **实现说明**
  - 储存对象依然使用一个数组`Object[][]`，用属性`rows`和`cols`分别记录此时Grid的总行数和总列数；
  - 在`put`时可能要进行Grid的扩大，我们提供`doubleExpand()`方法，该方法每次将Grid扩大一倍（即行数和列数分别乘以2），一次动态分配的时间复杂度为用$O(r*c)$。
- **运行结果**
  - 图3：创建一个 DynamicUnboundedGrid，提供两种创造方式，第一种默认构造一格16*16的初始网格，后一种可以指定行数和列数；
  - 图1：Bug在网格中正常运行；
  - 图2：由于DynamicUnboundedGridRunner的程序不够完善，此时Grid虽然已经扩大了但没有显示出来，Bug移动到了(0, 16)，但无法显示出来。由于Grid和World解耦，Grid不能操控World来进行`show()`，因此此结果是正常的；
  - 图4：DynamicUnboundedGridRunner主函数指定创建一个使用DynamicUnboundedGrid的网格，并在(18, 2)放置一个Bug，可以看到网格正常扩大并显示了出来。



- **Sonar 测试结果**：Passed