**Neighbor Counter Presentation Script**

Created: 04/07/21 by Tom Lever

Updated: 04/07/21 by Tom Lever

**1: Neighbor Counter**

I would like to present Neighbor Counter, a Java solution to counting the number of cells in distinct von-Neumann Neighborhoods, and the unions of von-Neumann Neighborhoods, that have been placed in a two-dimensional array, or grid, of cells.

**2: Execution**

I’d like to start by presenting the output of my solution. This output includes an introduction that explains the goal of a Count Requester, and mentions inputs supplied by the Requester, including grid height, grid width, number of nodes in grid around which von-Neumann Neighborhoods will be placed, and range of each von-Neumann Neighborhood.

The introduction also makes our task concrete and fun:

“We seek to count cells settled or farmed in a world of cells with a height of 18 cells and a width of 32 cells”. The world is settled by 4 settlers, each of whom chooses a random location. Each settler farms the land around him or her in a von-Neumann pattern of range 3.”

The output of my solution also includes presentations of our grid, with nodes (or, the settled world), our von-Neumann Neighborhood (or, farming pattern), and our grid, with von-Neumann Neighborhoods placed around nodes (or, the settled and farmed world). At the bottom is our count of the neighbors in distinct von-Neumann Neighborhoods and the unions of von-Neumann neighborhoods (or, the total number of cells settled and farmed in our world).

Any questions at this point?

**3: System Description: Context: CCRi’s Spatial Analysis**

Now, I’d like to continue by describing my solution, starting with a System Description and working down to code.

In my System Description, I considered some context for this project, which is that CCRi distills spatial data to inform client decisions and develops GIS tools for cloud-based computing. This was a great exercise in spatial modeling, analysis, and visualization.

**4: System Description: Opportunity Neighbor Counter Will Address**

In my System Description, I considered the opportunity that Neighbor Counter will address.

Developing Neighbor Counter really got me thinking about spatial modeling, analysis, and visualization.

**5: System Description: What Neighbor Counter Will Do**

Considering what neighbor counter will do,

Neighbor Counter will create a world, have settlers settle the world, and have settlers farm the world, all in the same farming pattern.

The Count Requester will provide world height, world width, number of settlers, and range of farming pattern with a command to start the application. The application will provide an introduction, which will use the provided parameters.

During the course of its run, the program will request permission to display the settled world, the farming pattern, and the settled and farmed world. The Count Requester will provide permission, or not.

Toward the end of its run, the application will provide to the Count Requester the number of cells of the world that were settled and farmed.

**6: System Description: Iterations of Development**

Considering the minimum viable capability and further iterations of development,

At the end of Iteration…

1. Neighbor Counter will have its minimum viable capability of creating a world, having settlers settle the world at random, having settlers farm the world all in the same von-Neumann farming pattern, providing opportunities to display the world and the farming pattern, and provide the total number of cells of the world farmed.
2. Neighbor Counter will have additional functionality to specify initial positions of settlers, have settlers farm the world in different patterns, and customize the shape of the world.
3. Neighbor Counter will have a React-based Graphical User Interface.

**7: Requirements: Use-Case Description: Request Neighbor Count**

At the point, I’d like to move from considering our system as a whole to studying some requirements. I’d like to start with considering just the main flow of the use case “Request Neighbor Count”.

Before I proceed, I’d like to pause and see if there’s any questions, comments, or requests.

Proceeding through the main flow of the use case “Request Neighbor Count”,

1. The Count Requester signals a desire to request a neighbor count / total number of cells of the world farmed.
2. The Requester provides to the Neighbor-Counter system: world height, world width, number of settlers, and range of farming pattern.
3. The system provides to the Requester an introduction to: the goal of the Requester, and the world, settlers, and farming pattern.
4. The system sets up a world based on world height and world width, based on a two-dimensional array of cells.
5. The system has settler organize based on the world and the number of settlers.
6. The system has settlers settle the world.
7. The system asks to display the settled world.
8. The Requester lets the system display the settled world.
9. The system displays the settled world.
10. The system sets up a farming pattern based on the provided range.
11. The system asks to display the farming pattern.
12. The Requester lets the system display the farming pattern.
13. The system displays the farming pattern.
14. The system has the settlers farm the world based on the farming pattern.
15. The system asks to display the settled and farmed world.
16. The Requester lets the system display the settled and farmed world.
17. The system displays the settled and farmed world.
18. The system provides the neighbor count / total number of cells of the world farmed.

I know that was a lot of steps: Any questions at this point?

**8: Requirements: Count Requester and Neighbor Counter Interface Design**

At this point, I just want to mention that I created an Interface Design Document for the Count Requester and the Neighbor Counter, which simply defines all of the inputs and outputs of the Neighbor-Counter system.

**9: Requirements: Decomposition and Sequential Flow**

I had fun creating this elegant diagram for the depicting the guts of the Neighbor-Counter system.

As you can see, the Neighbor Counter consists of an Input / Output Manager; a Data object consisting of world height, world width, number of settlers, and range of farming pattern; a World; Settlers; and a Farming Pattern.

Considering the Sequential Flow for Neighbor Counter,

1. The IO Manager provides the needed Data.
2. The IO Manager provides an introduction based on the provided Data.
3. A World is created based on world height and world width.
4. Settlers organize based on the world and the specified number of settlers.
5. Settlers settle individual cells of the world.
6. The IO Manager presents the settled world.
7. A Farming Pattern is created based on the range of the Farming Pattern.
8. The IO Manager presents the farming pattern.
9. Settlers farm the world based on the farming pattern.
10. And the IO Manager presents the settled and farmed world, and the neighbor count / number of cells of the world farmed.

**10: Design Classes**

At this point, we’re ready to consider some design classes, which bridge between System Design and coding.

I created Design classes for the five entities that you see in the Neighbor Counter Decomposition and Sequential Flow.

I also created Lattice, VonNeumannNeighborhood, Settler, and Location design classes.

Lattice is extended by World and FarmingPattern; all of these are based on two-dimensional grids.

VonNeumannNeighborhood extends FarmingPattern.

The group of Settlers is composed of instances of Settler.

Each settler has a unique Location.

Any questions at this point?

**11: Development Tools**

For a visual break, I’d like to just give a shout-out to Git, TortoiseGit, Eclipse, Ant, and, of course, Java.

**12: At this point, we’re ready to consider my Java solution.**

The project is called count-neighbors-in-von-neumann-neighborhoods.

I have four packages:

1. design\_classes
2. lattice\_utilities
3. neighbor\_counting\_utilities
4. settler\_utilities

I created an Ant build file with a makeUberJar target, which allows me to create executable Java Archives with all their dependencies built in.

Any questions?