Program #2 - The Matrix 4

Our Java sequel to the Matrix trilogy

Logistics:

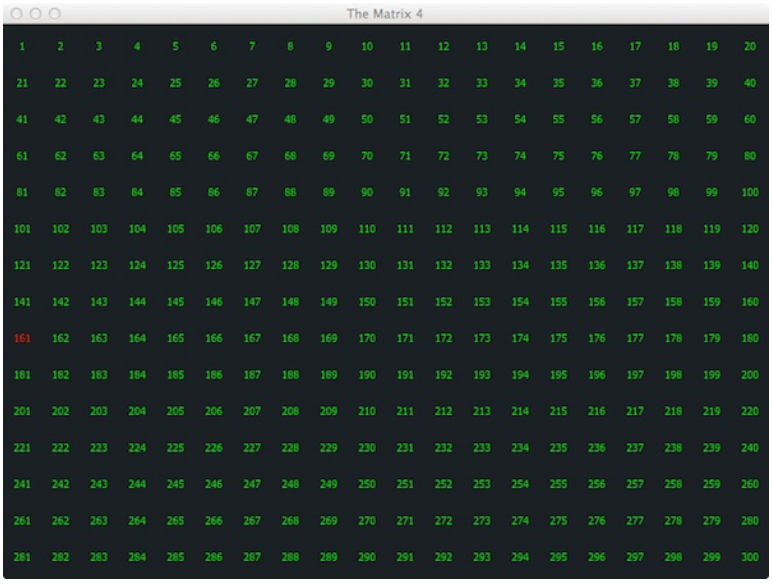
- Due: **Friday the 13th, Feb 2015**
- Worth: **12 points** (12% of your class grade)

/* This one's a little bigger, so a little more time and a couple more points */

1. Description

This used to be The Matrix:

In Program #2, the Matrix is:



In Program #2, we'll create our own simple game simulator. You can think of it as a very simple version of Minecraft or Rogue. In our Matrix, living creatures will wander around in search of assorted goodies. When a creature moves to a neighboring cell, four things can happen:

- If the cell is empty, then the creature moves to that cell
- If the cell is a barrier, then the creature stays put
- If the cell has a goodie, then the creature moves to the cell and collects the goodie
- If the cell is occupied by another living creature, then they battle. The winner of the battle gets the loser's goodies. The loser is removed from the Matrix.

In Program #2, we'll use:

- Inheritance, classes, abstract classes, and interfaces
- Polymorphism
- Java GUI programming and event-handling
- Creating many classes and getting comfortable doing that
- More Java stuff: enums, ArrayList, GridLayout, copy ctors, setters/getters
- The design process and importantly… your creativity. Huzzah!

2. Implementation

There's quite a bit of community code is this one. You'll find it in the common_area folder of my k: drive.

2.1. TheMatrix interface

The Matrix defines our simulator methods. He'll run the show. If your implementation is called FredMatrix, then...

- FredMatrix is-a TheMatrix
- FredMatrix has-a JPanel (and probably has-a JFrame too)
- FredMatrix has-a MatrixElement

Certainly, your Matrix simulator is complex, but the interface has only 3 methods:

- addElement(MatrixElement me) - adds an element to the matrix
- show() - makes your matrix frame visible
- step() - run one step of simulation by getting an action from each element in the matrix (like move or look) and then performing it

Your matrix will also have a ctor. This is a good place to pass in your settings and building your initial, empty matrix. I looks like this:

```
public FredMatrix( MatrixSettings settings) { … }
```

2.2. MatrixSettings class

We'll have many parameters to control the appearance and performance of the Matrix. Let's put them all in MatrixSettings. This will be a rare example of a class with public variables. Here's the setup:

```
public class MatrixSettings {
   public int panelWidth = 800;
   public int panelHeight = 600;
   public Color backgroundColar;   // set in ctor
   // add more setting variables here

   // ctor returns setting with default values
   public MatrixSettings() { … }

   // copy ctor
   public MatrixSettings( MatrixSettings copyMe) { … }
}
```

This approach is nice because 1) it's easy to change initial settings and 2) it keeps settings all together in one location. We make our class variables public here to avoid from having to create methods for each. We have a copy ctor so that when settings are passed to your matrix, you can copy them and prevent any outside changes once your matrix is running.

2.3. MatrixElement interface

The MatrixElement has methods that describe the appearance of the element: getColor(), toString(), getImage().

There are methods to define what the element can do: isCollectible(), isBarrier(), isAlive().

It also has methods that interact with the Matrix during simulation: nextAction(), move(), battle(), look(), and speak(). These need work.

2.4. TBD = To Be Designed

We'll dedicate class time to discuss issues that remain in our design:

- How will the simulator communicate with elements? Moving. Looking. Speaking. Etc. We'll flesh this out in class.
- Initially, we'll control our simulation steps from the console. Eventually, we'll want to do this graphically. In a separate frame?
- It would be nice to get a report of the current elements in the matrix and what goodies they have.
- When does the simulation stop? Do we need an explicit ending or just keep going until the user exits?
- What about adding elements during the simulation?
- Clicking on a cell show info about the element in that cell. What about right-click?

You are encouraged to make additions to your matrix. This may require a change to some of our interfaces. That's fine. Please show me if you you're going to change an interface.

2.5. Some Details

We will define a bunch of enums. See page 571 of our textbook if you need an enum refresher. Here we go:

- **MatrixAction** - the things a MatrixElement can do during its turn: Look, Move, Nothing, Speak.
- **BattleAction** - these are the options a Creature has during a battle: Rock, Paper, Scissors. Rock smashes scissors. Paper covers… oh, you know this one.
- **Direction** - North, South, East, West.

Some random notes:

- Use a GridLayout of JLabels to quickly build a Matrix panel.
- Each JLabel can show text: setText() and getText(). Or, JLabel can show an image: setIcon() and getIcon(). Each label can have its own colors: setForeground() and setBackground().
- Your main() might look like this:

```
main( String[] args) {
    Create matrix settings
    Create your matrix using settings
    Add matrix elements
    Show the matrix
    Simulate step by step
}
```

- Clicking on a cell should show what element is in the cell (and other info?)
- I created a MatrixCell class that I use internally in my matrix. It stores the element and label in each cell. I like having one structure that holds this information together. So, my matrix has an array of MatrixCell objects that I initialize in the ctor and use in the simulation method step().

3. Grading

Create a `program2` folder in your k: drive.
Place these files in that folder:

- A `README` file describing the state of your program.
- All the Java files that comprise your Program #1 solution

All your code must follow our 161 Coding Guidelines. Ugly code will be penalized with a 0-100% reduction in points. A program that doesn't even compile is worth 0 points.

3.1. Your creativity

In your README, please describe three important additions you made to your implementation of The Matrix. It may be a point system. A cool output window. A better way to run the simulator. And so on. The key is to have fun and exercise your creativity!

3.2. Process

The biggest obstacle some of you faced was an undefined process.
Here are some ideas.

- Use the default package in NetBeans. It's just a little simpler.
- If you struggled with Eclipse at all… switch to NetBeans. NOW! Now is the time to do it.
- Put your main() in a class Program02. Do this for every lab/program, then you've simplified your process a bit.
- Always have code that compiles and is **green** (in the upper right corner)
- In NetBeans, frequently Source/Format your code so that it always looks nice. I equate this to an auto mechanic keeping his work area clean.
- Pay attention to the Snippets file in class. I see people struggling to solve problems that are already available in Snippets.
- Write down (yes, on a piece of paper) the baby steps you will take to be successful: 1. Create an empty frame, 2. Add MatrixSettings, 3. Use settings to change the background color, 4. Use a GridLayout and JLabels that are numbered, and so on...

I see this program as having two distinct phases:

- ★ PHASE 1 - get your matrix setup and drawing and be able to add collectible goodies and barrier elements. You can run with this right now!
- ★ PHASE 2 - add living creatures to your matrix and simulate their movement, collecting and battles. We will do lots of design work to get to get here.

That's it.
That's enough.
My name… is Neo.
Good luck.
yow, bill

PS - Program #2 **bonus**. I am offering bonus grading on your Matrix.
Here's the deal:

- 2 extra points… 14 total points instead of 12
- And 3 extra days… due Mon Feb 16 before class, instead of Fri Feb 13
- For a better, spiffier matrix that you are really happy with and proud of