Program #2 Design Notes

Feb 2015

1. Main

What does your `main()` look like? Something like this: easy as 1, 2, 3:

main() {

1. create settings, change what you want, and then
create The Matrix using those settings

2. add elements to The Matrix: barriers, goodies, creatures

3. run your simulation one step at a time

}

2. MatrixSettings class

As shown above, the MatrixSettings are created and set in main(). But then, how do you communicate the `MatrixSettings` to `YourMatrix`? I did it in the ctor:

```
public YourMatrix( MatrixSettings settings) {
	// copy the settings passed in using copy ctor!
}
```

I ask for a copy ctor in your MatrixSettings. The copy ctor creates a new object that is a copy of an existing one. Look at page 554 in our text for a simple example.

3. Internals - the MatrixCell class

Well, most people will set an array of JLabels to get started. But we need more than that to move around the Matrix. For each cell, we (probably) need:

- The `JLabel` for that cell (required)
- The `MatrixElement` in the cell (can be null if empty)
- And (maybe) the cell number (0 thru N-1, for convenience sake)

So, I created a class called `MatrixCell` to manage this information:

```
public class MatrixCell {
	int num;
	JLabel label;
	MatrixElement element;

	// methods here...
```

```
}
```

Now, go back to my `YourMatrix` and create an array (or ArrayList) of `MatrixCell` objects… in my loop I set each cell's num, and create a `JLabel` for each cell, and set the element to null.

With this setup, it is now easy to write methods in `YourMatrix` like:

```
// get label at a cell
private JLabel getLabel( int cellNum) { … }
// get element at a cell, may be null
private MatrixElement getElement( int cellNum) { … }
// get adjacent cell in given direction
private int getAdjacentCell( int cellNum, Direction dir) { … }
```

Another important thing in the `YourMatrix` ctor… creating all the cells for your empty matrix. So, somewhere in your ctor, pseudo-code:

```
create an ArrayList<MatrixCell>
for cellNum from 0 to N-1 {
        // create a JLabel for this cell
        // use the settings for color and font choices

        // create a MatrixCell too, tell it about the JLabel you created
        // add the cell to the ArrayList
}
```

4. The step() method

The step() method is YourMatrix is probably your most complex code. He performs moves and collects goodies and issues battle requests and all. My pseudo-code looks like this:

```
create a new, empty ArrayList of elements called alreadyMoved
for each matrix cell {
        if( the cell isn't empty) {
                get the element at this cell
                if( the element is alive and isn't in the alreadyMoved list) {
                        get a direction from the element
                        move element in that direction (more on this later!)
                        add the element to the alreadyMoved list
                }
        }
}
```

I needed to track which elements were already moved, so I didn't move them more than once.

5. Internal methods used by step()

I have two important private methods in ProfBillMatrix that make moving happen:

```
// Move element in the cell into adjacent cell in direction d
void moveElement( MatrixCell cell, Direction d)

// return the cell number adjacent in the given direction
int adjacentCell( int cell, Direction d)
```

The `moveElement()` callscalls `adjacentCell()`. With the adjacent cell in hand, I check if there is an element in the cell. If the adjacent cell is empty, then move the element there. If a barrier is present, then do nothing. If it's a goodie, then collect it and move the element there. If it's another creature… battle!

"Moving" an element from one cell to another means 1) Removing the label and text from the old cell, and 2) setting the new cell's label or image using the element.

6. Barrier and Goodie classes

My barriers and goodies are uninteresting. Yours may not be. But I got tired of the same code in two barriers, for example: a BrickWall and a Hotel. So, I created a Barrier class. Barrier is-a MatrixElement. And BrickWall is-a Barrier. Hotel is-a Barrier too. For me, the only things different between two barriers was their name, image, and color.

So, three choices:

- Separate class for each barrier: `public class BrickWall implements MatrixElement { … }`
- Abstract Barrier class, separate class for each using the abstract: `public abstract class Barrier implements MatrixElement { … }` and `public class BrickWall extends Barrier { … }`
- Full Barrier class with class variables like name, image, and color. Pass the values in via the ctor. `public class Barrier implements MatrixElement { … }`. Then a brick wall is a new Barrier: `MatrixElement wall = new Barrier( "Brick wall", image, Color.RED);`

This choice is up to you.

7. Etc

When moving around your matrix… what happens at the edge? I think it would be cool to wrap-around. It shouldn't be hard to implement either.

I added random static methods to our `Direction` and `BattleAction` enums. They're in the k: drive common_area. Check them out. You call them like this:

```
// get a random Direction
Direction d = Direction.randomDirection();

// get a random BattleAction: Rock, Paper or Scissors
BattleAction ba = BattleAction.randomBattleAction();
```

I have shared some code in the class `MatrixHelper`. He's in the common_area too. The biggie is a method to read and resize a square image.

```
public static ImageIcon readMatrixImage(String fileName, int size) { … }
```

Also, there's a method to pause your program (in between simulation steps?)

```
public static void pause( int millis) { … }
```

Remember, static methods don't need an object to be called, just the class name. Like this:

```
// pause for half a second
MatrixHelper.pause( 500);
```

**** The design notes below are from the Sunday Feb 8 chat ****

Finding your matrix cell in an ActionListener

If your MatrixCell class has the MatrixElement AND the JLabel, then you can find the cell selected in an ActionListener by traversing the list of MatrixCell's, looking for the label.

```
MatrixCell findCellWithLabel( JLabel label) {
        for each matrix cell {
                if cell's label == label, then return the cell
        }
}
```

This kind of searching is why we created MatrixCell. We need to find the MatrixElement, given a JLabel. And we need to find the JLabel for any cell in the matrix.

The transition from JLabel to MatrixCell

Most of us start our matrix with an array or ArrayList of JLabels. Something like this:

```
for( int i = 0; i < numCells; i++) {
      JLabel lab = new JLabel( Integer.toString( i);
      panel.add( lab);
}
```

When you switch to MatrixCell, you need to add a couple steps.

```
ArrayList<MatrixCell> cells = new ArrayList();
for( int i = 0; i < numCells; i++) {
      JLabel lab = new JLabel( Integer.toString( i);
      pabel.add( lab);

      // add the matrix cell for this cell num
      MatrixCell cell = new MatrixCell( i, lab);
      cells.add( cell);
}
```

How's that?

What does an A/B graded Program #2 look like?

Program #2 is worth 14 points (almost an exam) and we've spent 3 weeks on it (gasp).
Here's what I'll look for so you can get at least and 11/14.

#1. Beautiful code!!!

See the code review sheet
And don't forget your README. Update it from earlier this week.

#2. Correct internals

Including:

- ➢ YourMatrix is-a TheMatrix
- ➢ Barrier, goodies and creatures is-a MatrixElement
- ➢ Use MatrixSettings for YourMatrix parameters (passed in cia the ctor)
- ➢ Copy ctor for MatrixSettings
- ➢ Add matrix elements in your main, outside YourMatrix

#3. Features

At a minimum, YourMatrix should:

- ➢ Draw the matrix
- ➢ Add elements to the matrix
- ➢ Correctly simulate moving elements: stop at barriers, collect goodies, battle creatures
- ➢ Add 3 extra features on your own. Identify them in your README

Program #2 is due Mon Feb 16 at the beginning of class.
thanks… yow, bill