HW 6
Akash Kulkarni, Victor Dong, Prakhar Dubey, Gautham Nagaraju

Framework Rationale:-

In our framework implementation, we have a main Framework class that connects the components of the framework together. This main Framework class houses the view (GUI) of the framework, the representation of the world in the game (A 2-dimensional array of cells) and the Rule (class which has the logic to decide next state of the world) to use. The interface of the Framework is as follows:-

```
public interface Framework {
	public void init();
	public void init(Rule r, Grid g, Icon[] imageMap, Color[] colorMap);
	public void init(Grid g);
	public void nextGrid();
	public void display();
	public void advance(int num);
	public void step(int num);
	public void setColor(Color[] c);
	public void setRule(Rule r);
	public void setImage(Icon[] i);
	public Icon[] getPictureMap();
	public Color[] getColorMap();
	public Grid getGrid();
}
```

More info can be found in the javadocs.

We have given the client the freedom to implement his own rules. To make a custom configuration, a client must create his own rules class (which extends the RulesAbstr abstract class). We have used the Template Method here. The client, if he wishes to just fills in the logic to figure out what the next state of a cell is. This decision was made so that the client has the freedom to declare his own rules for Cellular Automaton, but can choose the default configuration (Game Of Life).

The client must also instantiate a separate EntryScreen object, which displays a screen with a list of options that the client can choose from(step, run, etc.) if he wants a graphic interface. This is an example of a Decorator Pattern. This is because, in this case, additional behavior (in this case, the GUI) is added to an existing object dynamically. The FrameworkImpl is now a field of the EntryScreen object, thus demonstrating use of the Decorator class. The rationale behind using this class is that the GUI object EntryScreen now has its own functionality plus the functionality of the FrameworkImpl object.

The client has the ability to run his cellular automata by either stepping through the stages one by one or by advancing an arbitrary amount of steps in the GUI. The way this is implemented is through buttons that are attached with Action Listeners. By using the Observer Pattern, we allow the client to see the progress of the cellular automata at his own pace.

Plugin Use & Documentation:

In our design, we have 4 extension points that the client can plugin to. Our framework gives users the ability to define their own game (other than the default Game of Life) and initial positions. It

also provides him with two visualization plugin points, where the user can decide which colors he wants to be mapped to the states he provided, or which images he wants to be mapped to those states.
The four functions in the FrameworkImpl class that act as extension points that the client can plugin to:-

1) Init(Grid world) – This is a function defined in the Framework interface that allows the client to set custom values for the world. A Grid is basically a two dimensional array of cells, for which a client can specify number of states and initial configuration of the world.
2) setRules(Rule rule) – This is a function defined in the Framework interface that allows the client to set custom rules or the world. The user must make create his own Rules class and fill in the getNextState() logic for this class. If no argument is provided, default (Game of Life) rules are used
3) setColor(Color[] c) – This a function defined in the Framework interface that allows the client to specify which colors he wants to be mapped to the states he has declared. The client should make sure that he provides the same number of colors and states of the game.
4) setImage(Icon[] i) – This a function defined in the Framework interface that allows client to specify which images he wants to be mapped to states he has declared. The client should make sure that he provides the same number of images and states of the game.

Thus, if a user wishes to see a simulation of Brian's Brain configuration instead of the Default Game of Life, he can declare a brianBrainRule class that extends the AbstractRules class. In that class, the user must merely override the getNextState() method to what he wants the rules to be.

A sample Client Plugin would be:-

```
Framework f = new FrameworkImpl();
//Client initializes world to predefined values.
Grid grid = createGrid(8, 8);
f.init(grid);
//Client initializes default colors
Color[] colors = new Color[3];
colors[0] = Color.blue;
colors[1] = Color.black;
colors[2] = Color.green;
f.setColor(colors);
//MyRule contains the logic for whatever game you want to implement
Rule myRule = new MyRule();
f.setRule(myRule);
EntryScreen screen = new EntryScreen(f);
screen.display();
```

By providing null (or no arguments) user can use default values (Game Of Life, 8X8 grid).