Akash Kulkarni (aakulkar)
Gautham Nagaraju (gnagaraj)
Victor Dong (vdong)
Prakhar Dubey (pdubey)

Design Rationale:

In our design, we have a main FrameworkImpl Class that is more or less the skeleton. In this class, we have methods for the client to set various aspects of the framework such as which cellular automata you want to use, the colors you want to see, and how you want the initial world to look like.

The client can either choose one of the default configurations (Conway's Game of Life, Brian's Brain, Rule 101), or can make their own. To make a custom configuration, a client must create a RulesImpl 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 common ones. He must also create a Cell class, that specifies the number of states you want your automaton to take, and any possible pictures associated with it.

The client can view the GUI by calling the display() method. However, the actual GUI is taken care of by DisplayGrid class, which manages the layout, panels, and buttons of the GUI. This decoupling of implementor and abstraction (**Bridge Pattern**) is used so that we could have the freedom of changing one without it affecting the other.

The overall design also employs **Façade Pattern.** The FrameworkImpl consists of various subsystems (Grid of Cells, GUI, RuleImpl (the class that implements the next state logic). However, the client only interacts with the Framework interface, preventing the user from accessing the subsystems. The framework interface provides methods for initializing the framework, starting it, displaying the GUI and getters so that the framework user can analyze the data (grid).