Custom Feature Specification

In addition to the required features that are mentioned in the homework pdf, we added several custom features for the cellular automata.

Variable Dimensions

Built into the framework is support for 1D and 2D cellular automata. The user can also add support for more dimensions.

When implementing the required plugins, the user specifies the dimension by implementing the parameterized interface with the desired Point subclass. For example, they might the implement the Rule<T extends Point> interface for the Game of Life using the Point2D type parameter to specify that it is two dimensional like so: GameOfLifeRule implements Rule<Point2D>. Both the Rule and Viewport plugin interfaces require this point type parameter.

The user will then implement the methods of the interfaces according to the dimension they have chosen. For example, in the Rule interface, the stepCell method takes as a parameter a neighborhood of type Neighborhood<T> . This gives the all the neighboring cells given the dimension they have chosen, which the user can use to calculate the next state of a particular cell after one step.

The framework will use the dimension selected by the user to draw the cellular automata appropriately. For 1D, each step of the cellular automata will be drawn as line below the previous state in a grid. For 2D, each step will update the entire grid.

To add support for higher dimensions, the user can subclass the Point and Simulation classes. For example, the user might define Point3D, and Simulation3D which extends Simulation<Point 3D>. They would then complete the required abstract methods inherited from Simulation and provide a class to implement Viewport that draws the 3D grid as desired.

Wrap around

The user can specify which dimensions “wrap around” in the cellular automata. This means that cells on the edge of the grid will be treated as adjacent to the cells on the opposite edge when applying the stepCell method defined in the Rule interface.

To specify wrap, the user can call the setWraps(boolean[] wraps) method in State. For example, in a 2D simulation, the user could specify horizontal and vertical wrap around by calling setWraps(new boolean[] {true, true}) on the State object. This can be done when creating the initial state in the getInitialState method, or during runtime by first getting a reference to the state and then calling the setWraps method. Each individual dimension can also be set with the setWrap (int dimension, boolean wrap) method in State. By default, there is no wrap around.

Automata rule can depend on non-adjacent cell states

The user can specify the size of the “neighborhood” of nearby cells which the stepCell method can incorporate into its decision of the new state of a particular cell. That is, the user could make the next state of cell depend on, for example, a cell that is 2 cells north of it by specifying a neighborhood size that is at least 2. In the Rule interface, the user must implement getNeighborhoodSize which returns the size as an int. Then, the stepCell will automatically be called with a neighborhood of that size with which the user can use to decide the next state of the cell. Larger neighborhoods interact in interesting ways with grids that wrap around.

Customizable Viewport for displaying cellular automata

The user can implement the Viewport interface to have complete control over what graphics are drawn for the grid state of the cellular automata. The drawState method in the Viewport interface takes a State<T> object and a Graphics object. It can then draw the state however it likes onto the graphics object, which will be displayed in the GUI inside a ScrollPane.

Play speed

In the GUI, the user can set a time interval at which the simulation will step. A slider bar will provide the user this control over the play speed. A play button and pause button will allow the user to pause and resume the simulation. This functionality is built in to the framework.

Dynamic modification

The user can change the state of the grid, the size of the grid, which dimensions wrap around, the Visualization plugin, and even the Rule plugin after the simulation has started running. This allows for lots of flexibility on the part of the user. These changes can be applied by calling methods in Simulation and State, and then the application will update accordingly.