3-State Automata

Giorgi Tevdoradze

Contents:

1. Overview ---------- 1
2. Main Execution Loop ---------- 1
3. Classes ---------- 2
   3.1. SimBoard ---------- 2
   3.2. DrawBoard ---------- 3
   3.3. BoardWindow ---------- 5
4. Widget Diagram ---------- 7

Overview:

The 3-State Automata program runs in one window and exits when this window is closed. The window is divided in 2 parts, the left and the right side.

The left side contains a visual grid of pixels, which display the state of the generated board using three colors: green, yellow and black. These colors are later defined in code as variables with RGB codes 010, 110, and 000 respectfully.

The right side contains the control information displayed as several buttons, one label and two three-choice radio buttons.

Both sides of the board can be interacted with, and have their own corresponding functions.

The program code is stored in two .cs files. The file “automata.cs” contains the main logic, classes and execution of the program. “rules.cs” contains one class called “Rules”.

Main Execution Loop:

The main execution loop of the program is contained in the class “MainLoop”. This class has only 1 method, Main(), which contains the main execution loop.

The Main() method starts by initializing GTK using the GTK method Init(). After this, the method creates two variables **board** and **main**. **board** is of the class SimBoard and **main** is of the class BoardWindow, both of which are custom classes which will be discussed below. **board** represents the number matrix, which stores individual states (0, 1 or 2) and **main** represents the main window. After this the method calls **main** to draw the current board state

(which is all 0s so all greens) using the Draw() method and then uses the built-in GTK methods ShowAll() and Run() to finally start the window of the application.

The execution loop ends here, however since it is window-based it does not exit, until the user closes the window and finishes the interactions.

Classes:

- SimBoard

The SimBoard class is made to store the state of the simulation board in an integer matrix of 0s, 1s and 2s. It has 3 class variables: **n** of type int, which stores the size of 1 side of the square board. **numBoard** of type int[,], which stores the actual integer grid and the numbers based on their coordinates and **rules,** which is of the type "Rules", which is a custom class that stores the rule dictionaries, which will be discussed below.

Methods:

SimBoard() – Initializer method, takes in an integer as an input and creates a 2d integer array of the specified size filled with 0s. Sets **n** to the same size and callse the initializer method of **rules**.

IntFill() – Fills the entirety of the **numBoard** with an integer that it takes as input.

ApplyForestRule() – Calls ApplyRule() using the forest rule dictionaries.

ApplyScrollRule() – Calls ApplyRule() using the scroll rule dictionaries.

ApplyStairRule() – Calls ApplyRule() using the stair rule dictionaries.

ApplyRule() – Takes two dictionaries as input. Creates a new 2d integer board called **newBoardH** and fills all coordinates x, y of the board based on the number x, y of **numBoard** and the first input dictionary. For every x, y it inputs the number at x, y-1 number at x, y and number at x, y + 1 into the dictionary and gets the corresponding integer output. If y-1 or y+1 is out of bounds they "warp around" to the other side. After all the coordinates are done it reassigns **numBoard** to be **newBoardH** and creates another new 2d integer board called **newBoardV**. Similarly, it goes through all coordinates and fills them based on **numBoard** and the second input dictionary, however this time the input for x, y consists of the number at x+1, y number at x, y and the number at x-1, y. After the **newBoardV** is filled the method reassigns **numBoard** to be **newBoardV**.

- DrawBoard

The DrawBoard class is the one that represents the square area of the grid in the final window. It is the left side of the main window. It inherits from the DrawingArea class of the Cairo visuals library. It has 3 color variables **green**, **yellow** and **black** each of the Cairo class Color. It also has **numMatrix** of the type SimBoard, which is the integer board the class represents. It has **currColor** and **currRule** which are enums and keep track of the user choice for color and rule. It also has two doubles **xCord** and **yCord**, both store where the user clicked last in pixel measurements. Most importantly it has a variable **board** of the type ImageSurface from Cairo, which stores the visual Image state which needs to be displayed.

Methods:

DrawBoard() – Initializer method. Creates the Cairo ImageSurface **board** and sets the **numMatrix** as the input SimBoard variables. It also sets the starting color to be green and the

starting rule to be the first rule. It also paints the entire board green and uses the Cairo AddEvents() method to add OnButtonPressEvent using ButtonPressMask and ButtonReleaseMask.

OnDrawn() – Default Cairo method for all DrawingArea objects, which is called when QueueDraw() is called. In this instance it is changed to repaint the board from its previous state to the current one by setting the new generated **board** variable as the SetSourceSufrace() method variable.

OnButtonPressEvent() – Method which is called when the user presses the board in any location. The method gets the pixel location of the press using the EventButton **e** and if it is out of bounds returns out. If not, it divides the pixels by the grid ratio and determines which coordinate was pressed. After this, it fills the coordinate using the current choice of color in the **numMatrix** and calls the FillBoard() and QueueDraw() methods. FillBoard() makes the changes to the **board** and QueueDraw() updates the visuals.

fillCube() – Takes coordinates, context and color as input and fills the specified coordinate box in the current **board** using the Cairo Rectangle() and Fill() methods.

FillBorad() – Takes a SimBoard variable **InpBoard** as an input and for each of its entries, it colors the corresponding coordinates on the **board** using the numbers in the **InpBoard**'s 2d integer array. It does this by calling fillCube() on every coordinate.

ColorFill() – Fills the entire **board** with one color and calls the IntFill() method for its **numMatrix**.

SetColor() – Sets **currColor** as the input enum.

SetRule() – Sets **currRule** as the input enum.

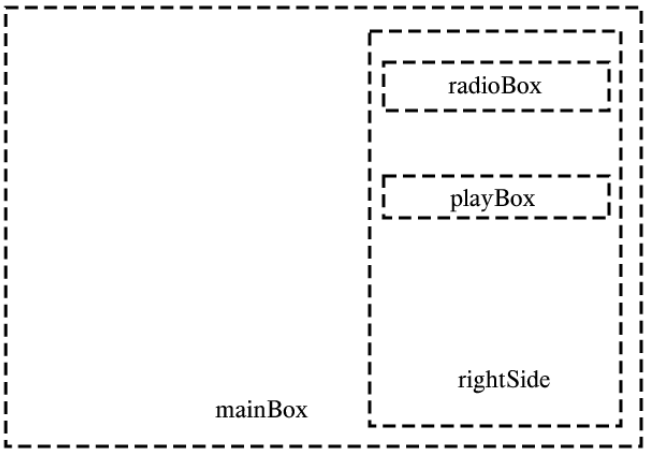
ApplyStep() – Based on the current rule stored in **currRule** the method calls the corresponding SimBoard methods of its **numMatrix** variable and then updates the **board** with FillBoard() and calls QueueDraw() to display it.

- BoardWindow

The BoardWindow class is the one that represents the main window of the application and contains the simulated visual board along with the controls. It has a DrawBoard variable **mainBoard** and the Boolean variable **playing**, which is false by default. Variable **playing** represents whether the window should be automatically applying steps.

Methods:

BoardWindow() – Initializer method. It resizes the window to a dimension of 950x750 pixels and initializes the **mainBoard** variable using the input. Then it creates GTK Box class variables to store visual boxes. These variables are **mainBox**, **rightSide**, **radioBox** and **playBar** and the layout looks something like this:



Apart from other boxes, the mainBox contains the **mainBoard** variable, which represents the left side. The **rightSide** box contains all the controls which are mostly GTK buttons such as **fill** and **applyStep** and also three radio buttons for rule selection **ruleOne**, **ruleTwo** and **ruleThree**. The **radioBox** contains three radio buttons **g**, **y** and **b** for color selection and the **playBar** contains **play** and **pause** buttons for animation control. This method also adds a GTK Timout function which executes the method OnTimeout() every 100ms.

OnDeleteEvent() – Quits the program if the window is closed.

Draw() – Calls the FillBoard() method of the **mainBoard**.

OnGreen(), OnYellow(), OnBlack() – Calls the SetColor() method of the **mainBoard** with the according color.

OnStep() – Calls the ApplyStep() method of the **mainBoard**.

OnForest(), OnScroll(), OnStair() – Calls the SetRule() method of the **mainBoard** with the according rule.

OnTimout() – This method is executed every 100ms, and if **playing** is true is applies one simulation step to the **mainBoard**.

OnPlay() – Sets **playing** to true.

OnPause() – Sets **playing** to false.

OnFill() – Calls the ColorFill() method of the **mainBoard**.

Here is a diagram of widgets and their assigned methods: