Ryan Fish and Arshia Surti

5/4/15

Game of Life - Software Requirement Specification

1. Introduction
   1. Purpose
      1. This document describes the software requirements for a modified version of Conway’s Game of Life.  Outlined below are specifications for Classes, their Variables, Methods and Constructors with a structural assumption of Java being used for implementation.
   2. System Overview
      1. The system will simulate Conway’s Game of Life. In addition to the base state machine, various agents will introduce disturbances to the pattern in characteristic ways.
2. Simulator Class
   1. Variables
      1. sec: The Simulator shall contain an integer sec that is in the range [0,100].
      2. msec: The Simulator shall contain an integer msec that is in the range [0,1000].
      3. dt: The Simulator shall contain an integer dt in the range [0, 1000].
         1. dt will specify the time in milliseconds between each step in the Game of Life.
   2. Constructor
      1. Simulator (): The constructor shall be empty.
   3. Methods
      1. getSec(): The method shall return the number of seconds elapsed since the simulation began.
      2. getMsec(): The method shall return the number of milliseconds elapsed since the last second.
      3. advanceClock()
         1. The method shall increase the variable msec by dt.
         2. If msec exceeds the range specified in 2.1.2, msec shall be decremented by 1000, and sec shall be incremented by 1.
      4. run()
         1. The method shall set the maximum time span of the simulation as specified in 2.1.1.
         2. The method shall call advanceClock() every dt milliseconds.
         3. The method shall update the grid every dt milliseconds.
      5. main()
         1. The method shall instantiate the Grid class.
         2. The method shall instantiate Agent subclasses.
         3. The method shall begin the simulation.
3. Grid Class
   1. Variables
      1. maxX: The Grid shall contain a positive integer maxX that represents the number of columns.
      2. maxY: The Grid shall contain a positive integer maxY that represents the number of rows.
      3. tileList: The Grid shall contain an array that contains references to each Tile instance in the grid.
   2. Constructor: Grid (String csvFile)
      1. The constructor shall take as input a reference to a csv file.
      2. The constructor shall call the initGrid() method.
   3. Methods
      1. initGrid(String csvFile)
         1. The method shall parse the csv file.
         2. The csv file shall have a rectangular number of cells.
         3. Each cell in the csv file shall contain a 0 or a 1.
         4. A 0 in a cell shall represent a white tile in the grid.
         5. A 1 in a cell shall represent a black tile in the grid.
         6. For each cell in the csv file, the method shall create an instance of the Tile class.
      2. repaint(): The method shall update the graphical representation of the grid since the last Simulation time step.
4. Tile Class
   1. Variables
      1. x: The Tile shall contain an integer x in the range [0, Grid.maxX-1].
      2. y: The Tile shall contain an integer y in the range [0, Grid.maxY-1].
      3. ID
         1. The Tile shall contain an integer ID equal to y\*Grid.maxX + x.
         2. Integer ID shall be in the range [0, (Grid.maxY\*Grid.maxX) - 1].
      4. color: The Tile shall contain a reference to color, an object of type Color.
      5. onOff: The Tile shall contain a boolean variable onOff that specifies whether a tile is alive or dead.
         1. If onOff is true, the tile is alive.
         2. If onOff is false, the tile is dead.
      6. decay: The Tile shall contain an integer decay in the range [0, 5].
         1. A decay of 0 represents full color.
         2. A decay of 5 represents black and white.
   2. Constructor
      1. Tile (Simulator s, Grid g, int x, int y)
         1. The constructor shall take as inputs a Simulator s, a Grid g, an integer x, and an integer y.
         2. If s or g are null, a null pointer exception shall be thrown.
         3. If x or y are not within the bounds specified in 4.1.1 and 4.1.2, an illegal argument exception shall be thrown.
      2. Tile (Simulator s, Grid g, int id)
         1. The constructor shall take as inputs a Simulator s, a Grid g, and an integer id.
         2. If s or g are null, a null pointer exception shall be thrown.
         3. If id is not within the bounds specified in 4.1.3, an illegal argument exception shall be thrown.
   3. Methods
      1. getID(): The method shall return the variable ID.
      2. getCoordinates(): The method shall return an array containing variables x and y.
      3. getDecay(): The method shall return the variable decay.
      4. incremDecay(): The method shall increment the value of decay within the bounds specified in 4.1.6.
      5. flip(): The method shall change the value of onOff.
      6. getOnOff(): The method shall return the value of onOff.
      7. getNeighbors(): The method shall return an array of pointers to all adjacent Tiles.
5. Color Class
   1. Variables
      1. tile: Color shall contain a reference to an object of type Tile.
      2. hue: Color shall contain an integer hue, representing the hue in the HSL model, in the range [0, 360).
      3. sat: Color shall contain an integer sat, representing the saturation in the HSL model, set to the integer 1.
      4. light: Color shall contain an integer light, representing the lightness in the HSL model, in the range [0,1].
      5. colorName: Color shall contain an enum colorName representing the name of the color mapping to the hue. Possible values are “red,” “green,” “blue,” “yellow,” “orange,” “purple,” and “pink.”
   2. Constructor: Color (Tile t, enum name, int decay)
      1. The constructor shall take as input a Tile t, an enum name and an integer decay.
      2. If t is null, a null pointer exception shall be thrown.
      3. The constructor shall set tile to t.
      4. The constructor shall call setColor (enum name, int decay).
   3. Methods
      1. getHSL(): The method shall return an array containing hue, sat, and light.
      2. getColor(): The method shall return colorName.
      3. updateColor()
         1. The method shall check the status of tile.getOnOff().
         2. If the status has changed since the last update, the method shall call setColor (colorName, tile.getDecay()) with the current color and decay value provided as input.
         3. If the status of onOff has not changed:
            1. If onOff is true, the method shall decrement the lightness by 0.05.
            2. If onOff is false, the method shall increment the lightness by 0.05.
      4. setColor(enum name, int decay)
         1. The method shall take as input an enum colorName and an integer decay.
         2. If decay is not within the bounds specified in 4.1.6, an illegal argument exception shall be thrown.
         3. The method shall set colorName to name.
         4. The method shall compute the hsl values associated with the name and the decay value.
         5. The method shall set the hue to the constant associated with the enum.
         6. The method shall set light to 0.5 if decay is 10.
         7. If onOff is true, the method shall set light to 0.25 - decay/20.
         8. If onOff is false, the method shall set light to 0.75 + decay/20.
6. Agent Superclass: The agent superclass shall be made abstract, and non-instantiable.
   1. Variables
      1. grid
         1. Each implementing class of Agent shall hold a reference to an object of type Grid, which contains the current state of the game board.
         2. The reference shall point to the same object held in the current Simulator instance.
      2. buffer
         1. Each implementing class of Agent shall allocate space in which to store deep copies of Tiles relevant to that Agent’s operation.
         2. These copies shall contain no reference back to the original object.
      3. runOnce
         1. Each implementing class of Agent shall store a boolean variable, whose value indicates whether the Agent shall continue running after one cycle, or halt and kill it’s thread.
      4. chance
         1. Each implementing class of Agent shall store a decimal number in the range (0,1], indicating the probability of that Agent becoming active during the current cycle.
      5. lastTime
         1. Each implementing class of Agent shall store the previous time of the cycle it last ran, for use in determining whether it should run in the current cycle.
      6. color
         1. Each implementing class of Agent shall store a color enum, representing a hue.
      7. buffX
         1. Each implementing class of Agent shall store the width of a row to be stored in buffer as a positive integer.
      8. buffY
         1. Each implementing class of Agent shall store the height of a column to be stored as a positive integer.
   2. Constructor
      1. Agent(Grid g, boolean runOnce, double chance): The constructor shall take as input:
         1. An instance of Grid, to be stored in grid
         2. A boolean, to be stored in runOnce
         3. A decimal value in the range (0,1], to be stored in chance
      2. The constructor shall initialize the time value in lastTime to the current time in Simulator.
      3. The constructor shall initialize a Set or Matrix to store Tiles as described in the buffer variable, using sizes specific to the subclass.
   3. Methods
      1. runCheck()
         1. The method shall select a value in the range (0,1] and compare it to the value stored in chance.  It shall return true if chance is greater than the random value, else it shall return false.
      2. bufferSize()
         1. The method shall return the intended size of the variable buffer.
      3. update()
         1. The method shall be abstract, and return nothing.
7. Blinker Class: The Blinker Class shall be a subclass of the Agent Superclass.
   1. Variables
      1. pattern: The class shall store a representation of the blinker pattern it applies to the game board.
   2. Constructor
      1. Blinker(Grid g, boolean runOnce, double chance): The constructor shall take as input:
         1. An instance of Grid, and passed to the super constructor
         2. A boolean, and passed to the super constructor
         3. A decimal value in the range (0,1], and passed to the super constructor
      2. The constructor shall set the values of buffX and buffY accordant to their ranges
   3. Methods
      1. update()
         1. The method shall check whether a change to the board is needed with runCheck(); if yes, continue, if no, set time and exit
         2. The method shall acquire locks on all tiles that will be read from.
         3. The method shall copy tile state information to the buffer
         4. The method shall set tiles not in their proper state, as given by the pattern variable, to the color designated in the color variable
         5. The method shall then release any locks it holds and return
8. Default Class: The Default Class shall be a subclass of the Agent Superclass.
   1. Variables
   2. Constructor
      1. Default(Grid g):  The constructor shall take as input:
         1. A reference to an object G of type Grid
      2. The constructor shall also set:
         1. runOnce to false
         2. chance to 1
      3. The constructor shall set the values of buffX and buffY to the corresponding dimensions of the grid
   3. Methods
      1. update()
         1. The method shall acquire locks on all tiles.
         2. The method shall copy tile state information to the buffer.
         3. The method shall set tiles according to the transition scheme defined in Conway’s Game of Life, using the color designated in the color variable.
         4. The method shall then release any locks it holds and return.