|  |
| --- |
| Problem 6. Cellular Automata (Conway's Game of Life) |

# Brief

This programming problem is the first tile-based and animated graphics work from the selection of problems. The tile based solution implemented is to follow Conways rules for the game of life. The program needs to be single threaded and allow basic world adjustments such as speed, pause and size.

# Tools & techniques

The application I created is a single threaded tile based GUI application using JComponents. The setup of my GUI in my constructor method includes setting the window title to display the problem name and creating a BorderLayout so I can add my world to the center. I have created a separate private class called menuBar that creates a JMenuBar for the buttons to be displayed in. As I am using a JMenuBar I have to create a JMenuItem instead of a JButton. Underneath I created my ActionListeners for each menu item but instead of typing the code directly into the actionPerformed method I called a different method from the World class as shown in Figure ?. This keeps the method condensed and is easier to find the correct method later on for editing.

|  |
| --- |
| @Override  public void actionPerformed(ActionEvent ae)  {  // Calls the startButton method from the world class.  world.startButton();  } |

My main method is also kept short by calling the constructor method ConwayGOF on start up.

The World class creates two version of the cell life array calling one cells and one copy. I also included Booleans, start and reset, to help with the button methods. The startButton() method uses the timer object created from the Timer package imported to start the application. The method updateWorld() is called here to run through the array of rows and columns for the copy array. The copy is then updated to the original cells array shown in Figure ?.

|  |
| --- |
| copy[i][j] = cells[i][j]; |

The resetButton() method checks the timer object and cancels it if necessary before calling the method makeWorld() to create a new version of the world (cells array). The pause and the speed button methods incorporate previously used code.

As Conways game has certain rules that need to be followed a method of determining which cells are alive and dead need to be created.

Graphics2D was used along with the BasicStroke layout to define a basic set of rendering attributes for the outlines of graphic primitives. The grid lines were created using Line2D for both the horizontal and vertical lines as demonstrated in Figure ?.

|  |
| --- |
| // Import Used  import java.awt.geom.Line2D;    Line2D hline = new Line2D.Double(0,i\*gridSize,590,i\*gridSize);  Line2D vline = new Line2D.Double(i\*gridSize,0,i\*gridSize,590);  gr2.draw(hline);  gr2.draw(vline); |

A private class, Animation, that extends the TimerTask import package was created. This deals with the timer during run time and checks the Boolean states to see which step is to be executed.

The Life class deals with the cells life state and gets the current cell we are looking at. The draw method uses Graphics2D like previously but this sets X and Y as the row and columns for the world along with the colour of the alive tiles. There is an empty method repaint() at the bottom of my class. This is method repaints the look of the component and is called repeatedly throughout my code to update the look of the component.

My previous work with Conways Game of Life in my Artificial Intelligence module gave me some ideas to think about for my application. It helped me know how to implement the rules effectively so that the game would function correctly.

# Results

When running my application the GUI window appears straight way with a random grid layout of alive and dead cells. The game will continue to run until the cycles either all die or it get stuck in a loop of bringing the same cells to life and killing them straight away. I have tested each of the buttons functionality as shown below in my decision table testing.

## Testing Results

### Junit Tests

I have created Junit tests for every class in my application however I did not get time to fully implement all the tests necessary for the application. This means that a lot of my tests are failing at the minute. If I had more time to improve my application I would get these fully functional and all passing.

### Decision Table

|  |  |  |
| --- | --- | --- |
| **Feature** | **Expected Result** | **Actual Result** |
| Start Button | The application will start showing a grid with cells, alive or dead, and operating according to the game rules. | The application starts and the game begins. |
| Pause Button | After pressing the button the game will pause at its current step | The game pauses until the start button is pressed to resume. |
| Speed Up Button | The game will increase in speed every time the button is pressed | The game speed increases after every click of the button. |
| Reset Button | After pressing the Reset button the game will begin to reset. After pressing start the grid will show the new layout. | The game ends and waits until the start button is pressed to present the new layout on the grid before starting again.  Before: After: |
| Close Button | The application will close down. | The application closes down. |
| Pressing the Reset button after speeding the game up | The game should pause, reset and wait for the start button to be pressed. | The game pauses, resets and waits for the start button to be pressed so the new layout can be displayed |
| Pressing the Pause button after speeding up the game | The game should pause and wait for the start button to be pressed | The game pauses and waits for the start button to be pressed. |

# Critique

I believe the choice to implement a menu bar for my buttons is a great addition to my application. This feature allows my buttons to be at the top of the GI without the usual JButton format. It has also cut down on the amount of code that is in my menuBar() method when implementing the ActionListeners because I am calling methods created in my World class.

Deciding to implement Graphics and Graphics2D classes for my paintComponent() method allowed me to provide better control over the coordinate transformations, colour management and text layout as well as the fact that Graphics2D is the fundamental class for 2 dimensional shapes. I believe my code is well presented and commented to the full extent. Each class file is clearly label to show what happens in them and my UI is well laid out with the appropriate functionality.

I would like to improve my reset button feature. At the moment it resets only when the reset button is pressed and then when the start button has been pressed again. Improving my application so that the game does not immediately start up again would be a future development. I would also like to find a different way of implementing my checkCellNeighbour() method due to using many if – else statements one after another. I think this makes my code look less efficient and would look to improving this in the future. Another feature I would like to improve is when the pause button is pressed and then resumed the application’s speed it’s reduced to the starting speed. If I had more time to improve this application I would make it so that the speed would be kept at the same left as before it was paused.

# Other details

# Java code

ConwaysGOF:

|  |
| --- |
| /\*  \* Conways Game Of Life Simulation.  \* The first tile-based and animated graphics work.  \*/  package conwaygof;  import java.awt.\*;  import java.awt.event.\*;  import javax.swing.\*;  /\*\*  \* @author Heidi Portwine (S6110438)  \*/  public class ConwayGOF extends JFrame  {  // Creates a World object  private World world = new World();    /\*\*  \* Setup of GUI.  \*/  public ConwayGOF()  {  // Text displayed on the Window Title.  super("Conways Game of Life");  // Adds the world to a border layout in the center.  add(world, BorderLayout.CENTER);  // Calls the menuBar and the pack methods.  menuBar();  pack();    // Sets up window to be fixed size, visible etc.  setResizable(false);  setDefaultCloseOperation(EXIT\_ON\_CLOSE);  setLocationRelativeTo(null);  setVisible(true);  }  /\*\*  \* Creates the menu at the top of the GUI Window.  \*/  private void menuBar()  {  // Creates a new JMenuBar called menu.  JMenuBar menu = new JMenuBar();  // Creates the menu buttons on the GUI Menu Bar.  final JMenuItem start = new JMenuItem("Start");  final JMenuItem pause = new JMenuItem("Pause");  final JMenuItem speed = new JMenuItem("Speed Up");  final JMenuItem reset = new JMenuItem("Reset");  JMenuItem close = new JMenuItem("Close");  // Adds the buttons to the Menu.  menu.add(start);  menu.add(pause);  menu.add(speed);  menu.add(reset);  menu.add(close);  // Sets the Menu Bar on the GUI to the menu created.  setJMenuBar(menu);    // Calls repaint on the world  world.repaint();  // Action Listener for the Start Button on the Menu.  start.addActionListener(new ActionListener()  {  @Override  public void actionPerformed(ActionEvent ae)  {  // Calls the startButton method from the world class.  world.startButton();  }  });  // Action Listener for the Pause Button on the Menu.  pause.addActionListener(new ActionListener()  {  @Override  public void actionPerformed(ActionEvent ae)  {  // Calls the pause method from the world class.  world.pauseButton();  }  });  // Action Listener for the Speed Button on the Menu.  speed.addActionListener(new ActionListener()  {  @Override  public void actionPerformed(ActionEvent ae)  {  // Calls the speed method from the world class.  world.speedButton();  }  });  // Action Listener for the Reset Button on the Menu.  reset.addActionListener(new ActionListener()  {  @Override  public void actionPerformed(ActionEvent ae)  {  // Calls the resetButton method from the world class.  world.resetButton();  }  });  // Action Listener for the Close Button on the Menu.  close.addActionListener(new ActionListener()  {  @Override  public void actionPerformed(ActionEvent ae)  {  // Closes the GUI.  System.exit(0);  }  });  }  /\*\*  \* Main Method that calls the ConwayGOF method on start up.  \* @param args  \*/  public static void main(String[] args)  {  new ConwayGOF();  }  } |

World:

|  |
| --- |
| /\*  \* Conways Game Of Life World Creation Class.  \*/  package conwaygof;  import java.awt.\*;  import java.awt.geom.Line2D;  import java.util.Timer;  import java.util.TimerTask;  import javax.swing.\*;  /\*\*  \* @author Heidi Portwine (S6110438)  \*/  public class World extends JComponent  {  // Variables and attributes declared.  public static final int gridSize = 20;  private static final int animationDelay = 20;  private static final int lifeUpdateDelay = 33;  public static Life[][] cellsArray;  public static Life[][] copyOfCells;  private Timer timer;  private int rows = Life.size, columns = Life.size;  private int frame;    // Booleans created to decided true or false depending on state.  boolean start;  boolean reset;  /\*\*  \* Constructor for World. Calls makeWorld() method  \* and sets the dimensions of the world.  \*/  public World()  {  super();  this.setPreferredSize(new Dimension(550,550));  makeWorld();  }  /\*\*  \* Creates a new world for the game.  \*/  public void makeWorld()  {  cellsArray = new Life[rows][columns];    for(int i = 0; i < rows; i++)  {  for(int j = 0; j < columns; j++)  {  cellsArray[i][j] = new Life(i,j,(Math.random()<0.1));  }  }  }  /\*\*  \* Start Button method that creates a new timer object and calls Animation().  \* Sets start to true, reset to false and finally calls the updateWorld() method.  \*/  public void startButton()  {  timer = new Timer();  timer.schedule(new Animation(),0,animationDelay);  start = true;  reset = false;  updateWorld();  }    /\*\*  \* Reset Button method that cancels the time if it is not already null.  \* Sets start to false and reset to true. Lastly calls the method makeWorld().  \*/  public void resetButton()  {  if(timer != null)  {  timer.cancel();  }  start = false;  reset = true;  makeWorld();  }    /\*\*  \* Pause Button method stops the timer until the start button is pressed.  \*/  public void pauseButton()  {  if(timer != null)  {  timer.cancel();  }  }    /\*\*  \* Speed Button method that increments the speed of the game  \* every time the button is pressed.  \*/  public void speedButton()  {  timer.schedule(new Animation(),0,animationDelay);  }    /\*\*  \* Updates the world created according to the game rules.  \*/  public void updateWorld()  {  // Creates an array of rows and columns.  copyOfCells = new Life[rows][columns];    for(int i = 0; i < rows; i++)  {  for(int j = 0; j < columns; j++)  {  copyOfCells[i][j] = cellsArray[i][j];  }  }  for(int i = 0; i < rows; i++)  {  for(int j = 0; j < columns; j++)  {  checkGameRules(cellsArray[i][j], copyOfCells[i][j]);  }  }  for(int i = 0; i < rows; i++)  {  for(int j = 0; j < columns; j++)  {  cellsArray[i][j] = copyOfCells[i][j];  }  }  // Calls the repaint() method to update the component.  this.repaint();  }  /\*\*  \* Method that defines the rules of the game.  \* @param cell  \* @param cell1  \*/  public void checkGameRules(Life cell, Life cell1)  {  // Checks the neighbours of the current cell and counts the live ones.  int lNeighbours = checkCellNeighbours(cell);    // Checks if the cell is currently alive.  if(cell.isCellAlive())  {  // If so then it checks if the cell has less than 2 or more than 3 live neighbours.  if(lNeighbours < 2 || lNeighbours > 3)  {  // If the cell has more than 3 live neighbours then the cell dies.  copyOfCells[cell.getRow()][cell.getColumn()].setCellAlive(false);  }  }  // Else if the cell has 3 live neighbours, then the cell becomes alive.  else if(lNeighbours == 3)  {  copyOfCells[cell.getRow()][cell.getColumn()].setCellAlive(true);  }  }  /\*\*  \* Method that is used to check on a cell's neighbours.  \* @param cell  \* @return neighbours  \*/  private int checkCellNeighbours(Life cell)  {  // Initialises variables as integers.  int neighbours = 0;  int topY;  int bottomY;  int rightX;  int leftX;    // Checking for top cell by getting the column number minus 1.  if(cell.getColumn()-1 >= 0)  {  // If its greater than or equal to 0 then its the top cell.  topY = cell.getColumn()-1;  }  else  {  topY = cell.getColumn();  }    // Checking for bottom cell by getting the column number plus 1.    if(cell.getColumn()+1 < Life.size)  {  // If it less than the life size then its the bottom cell.  bottomY = cell.getColumn()+1;  }  else  {  bottomY = cell.getColumn();  }    // Checking for right cell by getting the row number plus 1.  if(cell.getRow()+1 < Life.size)  {  // If its less than the life size then its the right cell.  rightX = cell.getRow()+1;  }  else  {  rightX = cell.getRow();  }    // Checking for left cell by getting the row number minus 1.  if(cell.getRow()-1 >= 0)  {  // If its greater than or equal to 0 then its the left cell.  leftX = cell.getRow()-1;  }  else  {  leftX = cell.getRow();  }  for(int x = leftX; x <= rightX; x++)  {  for(int y = topY; y <= bottomY; y++)  {  if(cellsArray[x][y].isCellAlive() && !(x == cell.getRow() && y == cell.getColumn()))  {  neighbours++;  }  }  }  return neighbours;  }  /\*\*  \* Method that deals with the Graphics of the GUI.  \* Sets the colours of squares and lines.  Puts the cellsArray on the board.  \* @param gr  \*/  @Override  protected void paintComponent(Graphics gr)  {  // Defines a basic set of rendering attributes for the outlines of graphics primitives.  Graphics2D gr2 = (Graphics2D)gr;  gr2.setColor(Color.BLACK);  gr2.setStroke(new BasicStroke(3));    // Drawing the lines on the grid.  for(int i=0; i<=Life.size; i++)  {  Line2D hLine = new Line2D.Double(0,i\*gridSize,590,i\*gridSize);  Line2D vLine = new Line2D.Double(i\*gridSize,0,i\*gridSize,590);  gr2.draw(hLine);  gr2.draw(vLine);  }  for(int i=0; i<rows; i++)  {  for(int j=0; j<columns; j++)  {  if(cellsArray[i][j].isCellAlive())  {  cellsArray[i][j].draw(gr2);  }  }  }  }    /\*\*  \* Animation Class that deals with the timer during run time.  \* It sets the frame to 0 and during run() checks the booleans  \* previously created to see which step to execute.  \*/  private class Animation extends TimerTask  {  /\*\*  \* Sets the frame to 0.  \*/  public Animation()  {  frame = 0;  }    /\*\*  \* Deals with the application while running.  \*/  @Override  public void run()  {  // Check if the boolean reset is true  if(reset)  {  // Sets the timer to null.  timer = null;  // Cancels the current timer.  timer.cancel();  return;  }    // If the boolean reset is false then it excecutes the code below.  // Increaments the tick  frame++;    // Divides the frame number by the lifeUpdateDelay  // and checks if its equal to 0. (% finds the remainder)  if(frame % lifeUpdateDelay == 0)  {  // Calls the update world method  updateWorld();  }  // Calls the repaint() method to update component.  repaint();  }  }  } |

Life:

|  |
| --- |
| /\*  \* Conways Game Of Life Cell Life Class.  \*/  package conwaygof;  import java.awt.Color;  import java.awt.Graphics2D;  /\*\*  \* @author Heidi Portwine (S6110438)  \*/  public class Life  {  // Declares variable types.  public static final int size = 50;  boolean isCellAlive;  int X, Y, row, column;  /\*\*  \* Constructor for cell life.  \* @param x  \* @param y  \* @param b  \*/  public Life(int x, int y, boolean b)  {  this.row = x;  this.column = y;  isCellAlive = b;  }  /\*\*  \* Gets the column value.  \* @return column  \*/  public int getColumn()  {  return column;  }    /\*\*  \* Gets the row value.  \* @return row  \*/  public int getRow()  {  return row;  }    /\*\*  \* Returns the boolean for is Alive.  \* @return isCellAlive  \*/  public boolean isCellAlive()  {  return isCellAlive;  }  /\*\*  \* Sets is Alive to boolean.  \* @param b  \*/  public void setCellAlive(boolean b)  {  isCellAlive = b;  }  /\*\*  \* Draw Method to set X and Y as the row and columns for the world (grid) created.  \* Sets the colour of the tiles to black.  \* @param gr2  \*/  public void draw(Graphics2D gr2)  {  X =(int)(row\*World.gridSize + World.gridSize/2d)-9;  Y =(int)(column\*World.gridSize + World.gridSize/2d)-9;  gr2.setColor(Color.BLACK);  gr2.fillRect(X,Y,19,19);  }  /\*\*  \* Repaint Method. Repaints the look of the component.  \*/  public void repaint()  {    }  } |