Week 3

Lingyan Zhou

June 26, 2015

Contents

1 Reflection

My program acheives all the goals in the specification. I tested it on some interesting patterns found on the Internet, and my program behaved as expected.

The primary difficaulty was debugging. Initially, I cannot visualized the intermediate steps. I finnaly changed the file format to png, with the help from the ImageReadWrite java class borrowed from CSC508. I can visualize the universe and thus can find bugs more easily. And with the help from my C++ image processing code in CSC508, implementing border extrapolation and simulation of the universe was not hard. What's hard is to find an interesting rule. I tried to simply change its born/stay-alive rules, say, to B234S234, but most of the time I got a totally random or totally dead universe in the end. However, I did find an interesting rule.

2 Interesting Rules

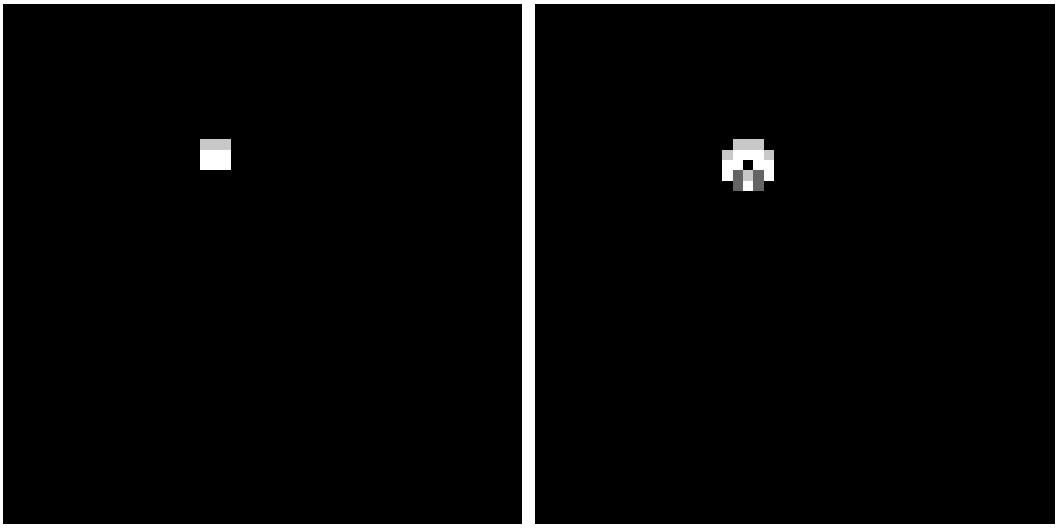
I think an interesting rules makes the final state of the universe interesting. And an interesting universe should display some repetitive patterns. For example, using the original rule set, the universe will not end up in a totally chaotic or empty state. There are patterns like spaceships, sliders, or still lifes. In the universe produced by my ruleset (see Figure 3b), the whole universe is a stable pattern.

3 Testing My Code

I tested my code by observing the output. I prepared some input data like spaceships, oscillators and still lifes. These patterns are predictable. I put some of the patterns on the edges so that I can test the border extrapolation functions. Then I run my program step by step and observe their outcomes. I also wrote the intermediate results to files. But the most used method was to create a simple pattern by clicking on the interface and press the "Step" button to see the result. For example, Figure 1 shows a simple test case and its expected result.

4 My Rule of the Universe

Let me explain the rule I create for the universe. Besides alive and dead state, I added three more states. One is growing, one is dying and the other



(a) Test input (b) Expected output after 2 steps

Figure 1: Testing

don't-care. Dont'-care state is used for borders and walls. A born and stay-alive condition is a predicate on the number of neighboring alive cells. In my current implementation, the one which produced Figure 3b, the born condition is that the 8-connected neighbors have 2 or 3 alive cells. and the stay-alive condition is that the 8-connected neighbors have 2, 3 or 4 alive cells. The rule set has 7 rules:

1. If any born condition is met, a dead cell becomes alive.
2. If any born condition is met, a growing cell becomes alive.
3. If none born condition is met, a growing cell becomes dying.
4. If none stay-alive condition is met, an alive becomes dying.
5. If any stay-alive condition is met, an alive stays alive.
6. If any stay-alive condition is met, a dying cell becomes growing.
7. Otherwise, it will be dead.

The ruleset is illustrated in Figure 2.

For other information, you can refer to the javadoc in the /doc/ folder.

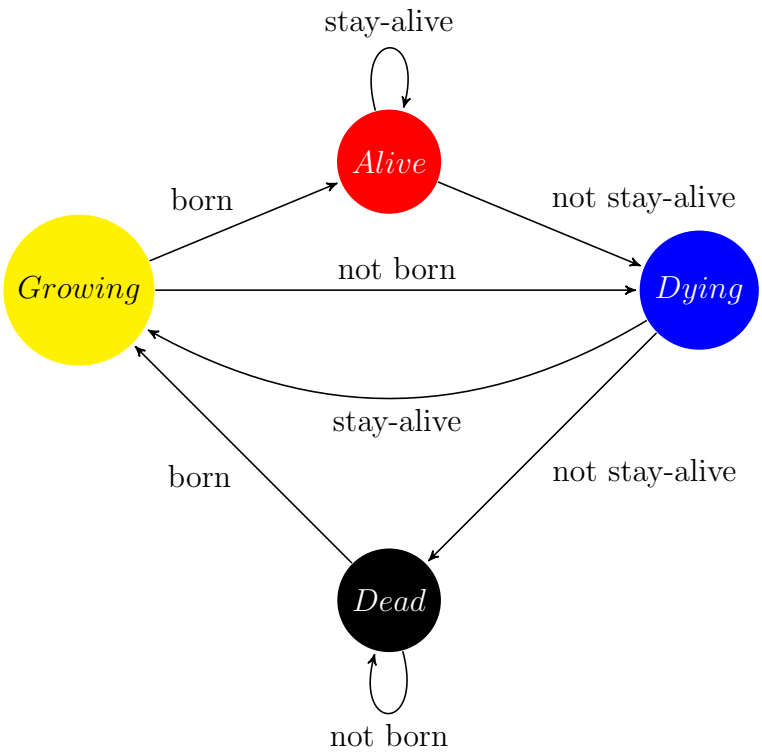
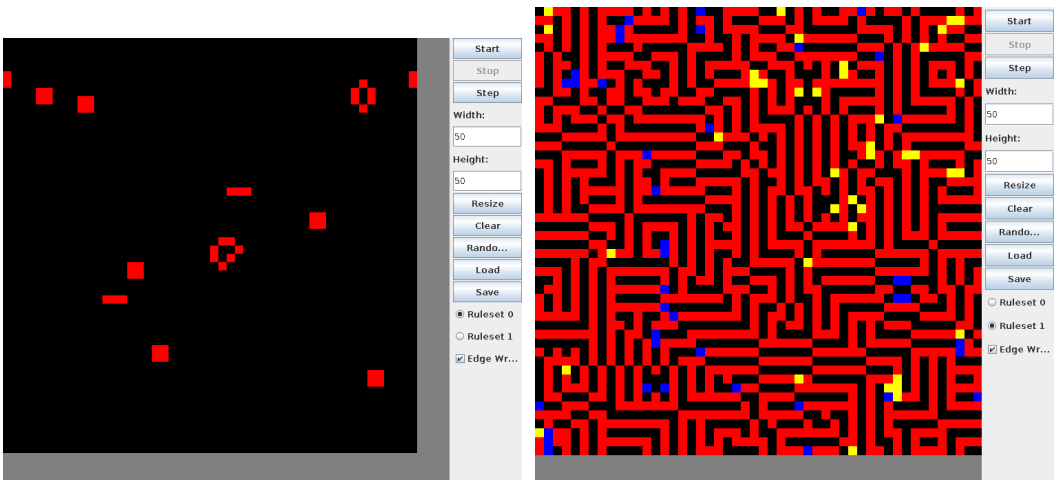


Figure 2: State transition of ruleset 2

5 Screenshot



(a) Stablized universe on ruleset 1 (b) Stablized universe on ruleset 2

Figure 3: Stablized universe

6 Listings

6.1 GameData.java

```
/**
 *
 * @author zhoulingyan
 *
 */
import java.io.File;
import java.util.Observable;
import java.util.Random;

/**
 * Represents game parameters and states.
 */
public class GameData extends Observable {
  /**
   * The cell is dead.
   */
  static public final byte STATE_DEAD = 0;
  /**
   * The cell is alive.
   */
  static public final byte STATE_ALIVE = (byte) 255;
  /**
   * The cell is not interesting. Used for paddings.
   */
  static public final byte STATE_DONTCARE = 2;
  /**
   * The cell is growing.
   */
  static public final byte STATE_GROWING = (byte) 200;
  /**
   * The cell is dying.
   */
  static public final byte STATE_DYING = 100;

  /**
   * Border extrapolation mode: wrapping aroud the borders.
   */
  static public final int BORDER_WRAP = 1;
  // static public final int BORDER_ALIVE = 2;
  // static public final int BORDER_DEAD = 3;
  /**
   * Border extrapolation mode: use don't-care values for the
    borders.
   */
```

```
  static public final int BORDER_DONTCARE = 4;
45
  /**
47  * Selected rule number.
   */
49 private int m_RuleNum;

51 /**
   * Selected border extrapolation mode
53  */
  private int m_BorderType;
55
  /**
57  * Cells states
   */
59 private byte[][] m_Data;

61 /**
   * Height of the universe
63  */
  private int m_Height;
65
  /**
67  * Width of the universe
   */
69 private int m_Width;

71 /**
   * Current GameRule object.
73  */
  private GameRule m_GameRule;
75
  /**
77  * Constructor.
   *
79  * @param height Height of the universe
   * @param width Width of the universe
81  */
  public GameData(int height, int width) {
83   m_Height = Math.max(1, height);
     m_Width = Math.max(1, width);
85   m_Data = new byte[m_Height][m_Width];
     m_RuleNum = 0;
87   m_GameRule = new GameRule1(GameRule1.BORN_3, GameRule1.
    STAY_ALIVE_2
         | GameRule1.STAY_ALIVE_3);
89   m_BorderType = BORDER_WRAP;
     clear();
91 }
```

```
93    /**
       * Get border extrapolation mode
95     *
       * @return Border extrapolation mode
97     */
      public int getBorderType() {
99      return m_BorderType;
      }
101
      /**
103    * Set border extrapolation mode
       *
105    * @param borderType New border extrapolation mode
       */
107   public void setBorderType(int borderType) {
        m_BorderType = borderType;
109     setChanged();
        notifyObservers();
111   }

113   /**
       * Get current rule number
115    *
       * @return Rule number
117    */
      public int getRuleNum() {
119     return m_RuleNum;
      }
121
      /**
123    * Set rule by rule number.
       * @param ruleNum New Rule number
125    */
      public void setRuleNum(int ruleNum) {
127     if (m_RuleNum != ruleNum) {
          m_RuleNum = ruleNum;
129       if (0 == m_RuleNum) {
            m_GameRule = new GameRule1(GameRule1.BORN_3,
131             GameRule1.STAY_ALIVE_2 | GameRule1.STAY_ALIVE_3);
          } else {
133         m_GameRule = new GameRule2(GameRule1.BORN_2|
       GameRule1.BORN_3,
              GameRule1.STAY_ALIVE_2 | GameRule1.STAY_ALIVE_3 |
        GameRule1.STAY_ALIVE_4
135               );
          }
137       setChanged();
          notifyObservers();
```

```
    }
  }

  /**
   * Get all the universe as 2D byte array
   * @return The universe as 2D byte array
   */
  public byte[][] getData() {
    return m_Data;
  }

  /**
   * Reset the universe.
   */
  public void clear() {
    for (int h = 0; h < m_Height; h++) {
      for (int w = 0; w < m_Width; w++) {
        m_Data[h][w] = 0;
      }
    }
    setChanged();
    notifyObservers();
  }

  /**
   * Go to the next generation.
   */
  public void step() {
    if (BORDER_WRAP == m_BorderType) {
      m_GameRule.process(m_Data, m_Height, m_Width,
          DataPadding.BORDER_WRAP, STATE_DONTCARE);
    } else if (BORDER_DONTCARE == m_BorderType) {
      m_GameRule.process(m_Data, m_Height, m_Width,
          DataPadding.BORDER_CONSTANT, STATE_DONTCARE);
    }
    setChanged();
    notifyObservers();
  }

  /**
   * Resize the dimension of the universe.
   *
   * @param height Height of the universe
   * @param width Width of the universe
   */
  public void resize(int height, int width) {
    if (height == m_Height && width == m_Width) {
      clear();
      setChanged();
```

```
      notifyObservers();
    } else {
      m_Height = Math.max(1, height);
      m_Width = Math.max(1, width);
      m_Data = new byte[m_Height][m_Width];
      clear();
      setChanged();
      notifyObservers();
    }
  }

  /**
   * Load the universe from a file.
   *
   * @param file The file
   */
  public void load(File file) {
    byte[][] tmp = ImageReadWrite.ImageRead(file);
    m_Height = tmp.length;
    m_Width = tmp[0].length;
    m_Data = tmp;
    setChanged();
    notifyObservers();
  }

  /**
   * Save the universe to a file.
   *
   * @param file The file
   */
  public void save(File outfile) {
    ImageReadWrite.ImageWrite(m_Data, outfile);
    setChanged();
    notifyObservers();
  }

  /**
   * Randomize the universe.
   */
  public void randomize() {
    Random rand = new Random();
    for (int h = 0; h < m_Height; h++) {
      for (int w = 0; w < m_Width; w++) {
        if (rand.nextBoolean()) {
          m_Data[h][w] = STATE_ALIVE;
        } else {
          m_Data[h][w] = STATE_DEAD;
        }
      }
```

```
237       }
          setChanged();
239       notifyObservers();
        }
241
        /**
243      * Flip the state at the specified cell.
         *
245      * @param y The position along the height.
         * @param x The position along the width.
247      */
        public void flipStateAt(int y, int x) {
249       if (x >= 0 && x < m_Width && y >= 0 && y < m_Height) {
            if (STATE_DONTCARE == m_Data[y][x]) {
251           m_Data[y][x] = STATE_DEAD;
            } else if (STATE_DEAD == m_Data[y][x]){
253           m_Data[y][x] = STATE_GROWING;
            } else if (STATE_GROWING == m_Data[y][x]){
255           m_Data[y][x] = STATE_ALIVE;
            } else if (STATE_ALIVE == m_Data[y][x]){
257           m_Data[y][x] = STATE_DYING;
            } else if (STATE_DYING == m_Data[y][x]){
259           m_Data[y][x] = STATE_DONTCARE;
            }
261         setChanged();
            notifyObservers();
263       }
        }
265
        /**
267      * Get the height of the universe.
         *
269      * @return The height of the universe.
         */
271     public int getHeight() {
          return m_Height;
273     }

275     /**
         * Get the width of the universe.
277      *
         * @return The width of the universe.
279      */
        public int getWidth() {
281       return m_Width;
        }
283   }
```

../src/GameData.java

6.2 DataPadding.java

```java
/**
 *
 * @author zhoulingyan
 *
 */
/**
 * Class to handle paddings.
 */
public class DataPadding {
  /**
   * Border wrap mode
   */
  static public final int BORDER_WRAP = 1;

  /**
   * Border constant mode
   */
  static public final int BORDER_CONSTANT = 2;

  /**
   * Find a source index of an array when given a position
   out of the range.
   * @param p The position.
   * @param len The length of the array.
   * @param borderType Border extrapolation mode.
   * @return The valid array index.
   */
  static public int borderInterpolate(int p, int len, int
   borderType) {
    if (borderType != BORDER_WRAP && borderType !=
   BORDER_CONSTANT) {
      throw new IllegalArgumentException("Unknown border type
   .");
    }
    if (len < 1) {
      throw new IllegalArgumentException("len<1.");
    }

    if (borderType == DataPadding.BORDER_WRAP) {
      if (p < 0) {
        return len + p % len;
      } else {
        return p % len;
      }
    } else if (borderType == DataPadding.BORDER_CONSTANT) {
      return -1;
    }
```

```
    return -1;
  }

  /**
   * Padding an array.
   *
   * @param src The source array
   * @param sHeight Array height
   * @param sWidth Array width
   * @param top Top border size
   * @param bottom Bottom border size
   * @param left Left border size
   * @param right Right border size
   * @param borderType The border extrapolation mode
   * @param value The border value when constant border mode
   is used.
   * @return The padded array.
   */
  static public byte[][] copyMakeBorder(byte[][] src, int
   sHeight,
      int sWidth, int top, int bottom, int left, int right,
      int borderType, byte value) {
    if (borderType != BORDER_WRAP && borderType !=
   BORDER_CONSTANT) {
      throw new IllegalArgumentException("Unknown border type
   .");
    }
    if (sHeight < 1) {
      throw new IllegalArgumentException("len<1.");
    }
    if (sWidth < 1) {
      throw new IllegalArgumentException("sWidth<1.");
    }
    if (top < 0) {
      throw new IllegalArgumentException("top<0.");
    }
    if (bottom < 0) {
      throw new IllegalArgumentException("bottom<0.");
    }
    if (left < 0) {
      throw new IllegalArgumentException("left<0.");
    }
    if (right < 0) {
      throw new IllegalArgumentException("right<0.");
    }

    byte[][] dst = new byte[sHeight + top + bottom][sWidth +
   left + right];
    for (int h = 0; h < sHeight; ++h) {
```

```
      for (int w = 0; w < sWidth; ++w) {
        dst[h + top][w + left] = src[h][w];
      }
    }

    if (borderType == BORDER_CONSTANT) {
      for (int h = 0; h < top; ++h) {
        for (int w = 0; w < sWidth + left + right; ++w) {
          dst[h][w] = value;
        }
      }
      for (int h = sHeight + top; h < sHeight + top + bottom;
    ++h) {
        for (int w = 0; w < sWidth + left + right; ++w) {
          dst[h][w] = value;
        }
      }
      for (int h = top; h < sHeight + top; ++h) {
        for (int w = 0; w < left; ++w) {
          dst[h][w] = value;
        }
      }
      for (int h = top; h < sHeight + top; ++h) {
        for (int w = left + sWidth; w < sWidth + left + right
    ; ++w) {
          dst[h][w] = value;
        }
      }
    } else if (borderType == BORDER_WRAP) {
      for (int h = top; h < sHeight + top; ++h) {
        for (int w = 0; w < left; ++w) {
          dst[h][w] = src[h - top][borderInterpolate(w - left
    ,
              sWidth, borderType)];
        }
      }
      for (int h = top; h < sHeight + top; ++h) {
        for (int w = left + sWidth; w < sWidth + left + right
    ; ++w) {
          dst[h][w] = src[h - top][borderInterpolate(w - left
    ,
              sWidth, borderType)];
        }
      }
      for (int h = 0; h < top; ++h) {
        for (int w = 0; w < sWidth + left + right; ++w) {
          dst[h][w] = dst[top
              + borderInterpolate(h - top, sHeight,
    borderType)][w];
```

```
131         }
        }
133       for (int h = sHeight + top; h < sHeight + top + bottom;
      ++h) {
          for (int w = 0; w < sWidth + left + right; ++w) {
135         dst[h][w] = dst[top
              + borderInterpolate(h - top, sHeight,
    borderType)][w];
          }
137       }
      }
139     return dst;
    }
141 }
```

../src/DataPadding.java

6.3 GameRule.java

```
/**
2  * GameRule base class
   * @author zhoulingyan
4  *
   */
6 public abstract class GameRule {
    /**
8    * Simulate one generation.
     * @param data The universe.
10   * @param height The height of the universe.
     * @param width The width of the universe.
12   * @param borderType The border extrapolation mode.
     * @param value Used as the padding value when using the
     constant border mode.
14   */
    abstract public void process(byte[][] data, int height, int
      width,
16      int borderType, byte value);
}
```

../src/GameRule.java

6.4 GameRule1.java

```
1 /**
   *  @author zhoulingyan
3  */
```

```
/**
 * Game rule 1 class.
 *
 * 3 rules:
 * <ol>
 *   <li>If any born condition is met, a dead cell becomes alive.</li>
 *   <li>If any stay-alive condition is met, an alive cell stays alive.</li>
 *   <li>Otherwise, it will be dead.</li>
 * </ol>
 */
public class GameRule1 extends GameRule {

  /**
   * Border top.
   */
  static private final int BORDER_TOP = 1;
  /**
   * Border bottom.
   */
  static private final int BORDER_BOTTOM = 1;
  /**
   * Border left.
   */
  static private final int BORDER_LEFT = 1;
  /**
   * Border right.
   */
  static private final int BORDER_RIGHT = 1;


  /**
   * If neighbors have exactly 0 alive cell(s), a dead cell will be born.
   */
  static public final long BORN_0 = 0x1L;
  /**
   * If neighbors have exactly 1 alive cell(s), a dead cell will be born.
   */
  static public final long BORN_1 = 0x10L;
  /**
   * If neighbors have exactly 2 alive cell(s), a dead cell will be born.
   */
  static public final long BORN_2 = 0x100L;
  /**
```

```
47      * If neighbors have exactly 3 alive cell(s), a dead cell
         will be born.
         */
49    static public final long BORN_3 = 0x1000L;
      /**
51      * If neighbors have exactly 4 alive cell(s), a dead cell
         will be born.
         */
53    static public final long BORN_4 = 0x10000L;
      /**
55      * If neighbors have exactly 5 alive cell(s), a dead cell
         will be born.
         */
57    static public final long BORN_5 = 0x100000L;
      /**
59      * If neighbors have exactly 6 alive cell(s), a dead cell
         will be born.
         */
61    static public final long BORN_6 = 0x1000000L;
      /**
63      * If neighbors have exactly 7 alive cell(s), a dead cell
         will be born.
         */
65    static public final long BORN_7 = 0x10000000L;
      /**
67      * If neighbors have exactly 8 alive cell(s), a dead cell
         will be born.
         */
69    static public final long BORN_8 = 0x100000000L;
      /**
71      * If neighbors have exactly 0 alive cell(s), a n alive
         cell stays alive.
         */
73    static public final long STAY_ALIVE_0 = 0x1L;
      /**
75      * If neighbors have exactly 1 alive cell(s), a n alive
         cell stays alive.
         */
77    static public final long STAY_ALIVE_1 = 0x10L;
      /**
79      * If neighbors have exactly 2 alive cell(s), a n alive
         cell stays alive.
         */
81    static public final long STAY_ALIVE_2 = 0x100L;
      /**
83      * If neighbors have exactly 3 alive cell(s), a n alive
         cell stays alive.
         */
85    static public final long STAY_ALIVE_3 = 0x1000L;
```

```
    /**
     * If neighbors have exactly 4 alive cell(s), a n alive
     cell stays alive.
     */
    static public final long STAY_ALIVE_4 = 0x10000L;
    /**
     * If neighbors have exactly 5 alive cell(s), a n alive
     cell stays alive.
     */
    static public final long STAY_ALIVE_5 = 0x100000L;
    /**
     * If neighbors have exactly 6 alive cell(s), a n alive
     cell stays alive.
     */
    static public final long STAY_ALIVE_6 = 0x1000000L;
    /**
     * If neighbors have exactly 7 alive cell(s), a n alive
     cell stays alive.
     */
    static public final long STAY_ALIVE_7 = 0x10000000L;
    /**
     * If neighbors have exactly 8 alive cell(s), a n alive
     cell stays alive.
     */
    static public final long STAY_ALIVE_8 = 0x100000000L;

    /**
     * Cell born conditions
     */
    private int[] m_BornRule;

    /**
     * Cell stay-alive condition
     */
    private int[] m_StayAliveRule;

    /**
     * Constructor
     * @param bornCond Cell born conditions.
     * @param stayAliveCond Cell stay-alive conditions.
     */
    public GameRule1(long bornCond, long stayAliveCond) {
      boolean[] born = new boolean[9];
      boolean[] stay = new boolean[9];
      born[0] = (BORN_0 & bornCond) != 0 ? true : false;
      born[1] = (BORN_1 & bornCond) != 0 ? true : false;
      born[2] = (BORN_2 & bornCond) != 0 ? true : false;
      born[3] = (BORN_3 & bornCond) != 0 ? true : false;
      born[4] = (BORN_4 & bornCond) != 0 ? true : false;
```

```
    born[5] = (BORN_5 & bornCond) != 0 ? true : false;
    born[6] = (BORN_6 & bornCond) != 0 ? true : false;
    born[7] = (BORN_7 & bornCond) != 0 ? true : false;
    born[8] = (BORN_8 & bornCond) != 0 ? true : false;
    stay[0] = (STAY_ALIVE_0 & stayAliveCond) != 0 ? true :
  false;
    stay[1] = (STAY_ALIVE_1 & stayAliveCond) != 0 ? true :
  false;
    stay[2] = (STAY_ALIVE_2 & stayAliveCond) != 0 ? true :
  false;
    stay[3] = (STAY_ALIVE_3 & stayAliveCond) != 0 ? true :
  false;
    stay[4] = (STAY_ALIVE_4 & stayAliveCond) != 0 ? true :
  false;
    stay[5] = (STAY_ALIVE_5 & stayAliveCond) != 0 ? true :
  false;
    stay[6] = (STAY_ALIVE_6 & stayAliveCond) != 0 ? true :
  false;
    stay[7] = (STAY_ALIVE_7 & stayAliveCond) != 0 ? true :
  false;
    stay[8] = (STAY_ALIVE_8 & stayAliveCond) != 0 ? true :
  false;

    int bornRuleNum = 0;
    int stayRuleNum = 0;
    for (int i = 0; i < 9; ++i) {
      if (born[i]) {
        ++bornRuleNum;
      }
      if (stay[i]) {
        ++stayRuleNum;
      }
    }

    m_BornRule = new int[bornRuleNum];
    m_StayAliveRule = new int[stayRuleNum];

    int curBornRuleIndex = 0;
    int curStayAliveRuleIndex = 0;
    for (int i = 0; i < 9; ++i) {
      if (born[i]) {
        m_BornRule[curBornRuleIndex] = i;
        ++curBornRuleIndex;
      }
      if (stay[i]) {
        m_StayAliveRule[curStayAliveRuleIndex] = i;
        ++curStayAliveRuleIndex;
      }
    }
```

```
  }
171
173   @Override
      public void process(byte[][] data, int height, int width,
       int borderType,
175       byte value) {
       byte[][] tmp = DataPadding.copyMakeBorder(data, height,
       width,
177         BORDER_TOP, BORDER_BOTTOM, BORDER_LEFT, BORDER_RIGHT,
            borderType, value);
179     for (int h = 0; h < height; ++h) {
          for (int w = 0; w < width; ++w) {
181         byte curCellState = tmp[h + BORDER_TOP][w +
       BORDER_LEFT];
            int aliveCount = 0;
183         for (int kh = -BORDER_TOP; kh <= BORDER_BOTTOM; ++kh)
       {
              for (int kw = -BORDER_LEFT; kw <= BORDER_RIGHT; ++
       kw) {
185             if (kh == 0 && kw == 0) {
                  continue;
187             }
                if (GameData.STATE_ALIVE == tmp[h + BORDER_TOP +
       kh][w
189                   + BORDER_LEFT + kw]) {
                  ++aliveCount;
191             }
              }
193         }
            if (GameData.STATE_ALIVE == curCellState) {
195           boolean stayAlive = false;
              for (int i = 0; i < m_StayAliveRule.length; ++i) {
197             if (aliveCount == m_StayAliveRule[i]) {
                  stayAlive = true;
199               break;
                }
201           }
              if (!stayAlive) {
203             data[h][w] = GameData.STATE_DEAD;
              }
205         } else if (GameData.STATE_DEAD == curCellState) {
              boolean born = false;
207           for (int i = 0; i < m_BornRule.length; ++i) {
                if (aliveCount == m_BornRule[i]) {
209               born = true;
                  break;
211             }
              }
```

```
213             if (born) {
                  data[h][w] = GameData.STATE_ALIVE;
215             }
            } else if (GameData.STATE_DYING == curCellState) {
              boolean born = false;
              for (int i = 0; i < m_BornRule.length; ++i) {
219             if (aliveCount == m_BornRule[i]) {
                  born = true;
221               break;
                }
223           }
              if (born) {
225             data[h][w] = GameData.STATE_ALIVE;
              }
227         } else if (GameData.STATE_GROWING == curCellState) {
              boolean born = false;
              for (int i = 0; i < m_BornRule.length; ++i) {
229             if (aliveCount == m_BornRule[i]) {
                  born = true;
231               break;
                }
233           }
              if (born) {
235             data[h][w] = GameData.STATE_ALIVE;
              }
237         }
          }
239     }
    }
241 }
}
```

../src/GameRule1.java

6.5 GameRule2.java

```
/**
2 * @author zhoulingyan
 */
4 /**
 * Game rule 2 class.
6 *
 * 7 rules:
8 * <ol>
 *   <li>If any born condition is met, a dead cell becomes
   growing.</li>
10 *   <li>If any born condition is met, a growing cell becomes
    alive.</li>
```

```
 *   <li>If none born condition is met, a growing cell
    becomes dying.</li>
 *   <li>If none stay-alive condition is met, an alive
    becomes dying.</li>
 *   <li>If any stay-alive condition is met, an alive stays
    alive.</li>
 *   <li>If any stay-alive condition is met, a dying cell
    becomes growing.</li>
 *   <li>Otherwise, it will be dead.</li>
 * </ol>
 */
public class GameRule2 extends GameRule {

  static private final int BORDER_TOP = 1;
  static private final int BORDER_BOTTOM = 1;
  static private final int BORDER_LEFT = 1;
  static private final int BORDER_RIGHT = 1;

  static public final long BORN_0 = 0x1L;
  static public final long BORN_1 = 0x10L;
  static public final long BORN_2 = 0x100L;
  static public final long BORN_3 = 0x1000L;
  static public final long BORN_4 = 0x10000L;
  static public final long BORN_5 = 0x100000L;
  static public final long BORN_6 = 0x1000000L;
  static public final long BORN_7 = 0x10000000L;
  static public final long BORN_8 = 0x100000000L;
  static public final long STAY_ALIVE_0 = 0x1L;
  static public final long STAY_ALIVE_1 = 0x10L;
  static public final long STAY_ALIVE_2 = 0x100L;
  static public final long STAY_ALIVE_3 = 0x1000L;
  static public final long STAY_ALIVE_4 = 0x10000L;
  static public final long STAY_ALIVE_5 = 0x100000L;
  static public final long STAY_ALIVE_6 = 0x1000000L;
  static public final long STAY_ALIVE_7 = 0x10000000L;
  static public final long STAY_ALIVE_8 = 0x100000000L;

  private int[] m_BornRule;
  private int[] m_StayAliveRule;

  /**
   * Constructor
   * @param bornCond Cell born conditions.
   * @param stayAliveCond Cell stay-alive conditions.
   */
  public GameRule2(long bornCond, long stayAliveCond) {
    boolean[] born = new boolean[9];
    boolean[] stay = new boolean[9];
    born[0] = (BORN_0 & bornCond) != 0 ? true : false;
```

```
56      born[1] = (BORN_1 & bornCond) != 0 ? true : false;
        born[2] = (BORN_2 & bornCond) != 0 ? true : false;
58      born[3] = (BORN_3 & bornCond) != 0 ? true : false;
        born[4] = (BORN_4 & bornCond) != 0 ? true : false;
60      born[5] = (BORN_5 & bornCond) != 0 ? true : false;
        born[6] = (BORN_6 & bornCond) != 0 ? true : false;
62      born[7] = (BORN_7 & bornCond) != 0 ? true : false;
        born[8] = (BORN_8 & bornCond) != 0 ? true : false;
64      stay[0] = (STAY_ALIVE_0 & stayAliveCond) != 0 ? true :
      false;
        stay[1] = (STAY_ALIVE_1 & stayAliveCond) != 0 ? true :
      false;
66      stay[2] = (STAY_ALIVE_2 & stayAliveCond) != 0 ? true :
      false;
        stay[3] = (STAY_ALIVE_3 & stayAliveCond) != 0 ? true :
      false;
68      stay[4] = (STAY_ALIVE_4 & stayAliveCond) != 0 ? true :
      false;
        stay[5] = (STAY_ALIVE_5 & stayAliveCond) != 0 ? true :
      false;
70      stay[6] = (STAY_ALIVE_6 & stayAliveCond) != 0 ? true :
      false;
        stay[7] = (STAY_ALIVE_7 & stayAliveCond) != 0 ? true :
      false;
72      stay[8] = (STAY_ALIVE_8 & stayAliveCond) != 0 ? true :
      false;

74      int bornRuleNum = 0;
        int stayRuleNum = 0;
76      for (int i = 0; i < 9; ++i) {
          if (born[i]) {
78          ++bornRuleNum;
          }
80        if (stay[i]) {
            ++stayRuleNum;
82        }
        }
84
        m_BornRule = new int[bornRuleNum];
86      m_StayAliveRule = new int[stayRuleNum];

88      int curBornRuleIndex = 0;
        int curStayAliveRuleIndex = 0;
90      for (int i = 0; i < 9; ++i) {
          if (born[i]) {
92          m_BornRule[curBornRuleIndex] = i;
            ++curBornRuleIndex;
94        }
          if (stay[i]) {
```

```
96          m_StayAliveRule[curStayAliveRuleIndex] = i;
            ++curStayAliveRuleIndex;
98        }
      }
100   }

102   @Override
      public void process(byte[][] data, int height, int width,
       int borderType,
104       byte value) {
       byte[][] tmp = DataPadding.copyMakeBorder(data, height,
       width,
106         BORDER_TOP, BORDER_BOTTOM, BORDER_LEFT, BORDER_RIGHT,
            borderType, value);
108    for (int h = 0; h < height; ++h) {
         for (int w = 0; w < width; ++w) {
110        byte curCellState = tmp[h + BORDER_TOP][w +
       BORDER_LEFT];
           int aliveCount = 0;
112        for (int kh = -BORDER_TOP; kh <= BORDER_BOTTOM; ++kh)
        {
             for (int kw = -BORDER_LEFT; kw <= BORDER_RIGHT; ++
       kw) {
114            if (kh == 0 && kw == 0) {
                 continue;
116            }
               if (GameData.STATE_ALIVE == tmp[h + BORDER_TOP +
       kh][w
118                 + BORDER_LEFT + kw]) {
                 ++aliveCount;
120            }
             }
122        }
           if (GameData.STATE_ALIVE == curCellState) {
124          boolean stayAlive = false;
             for (int i = 0; i < m_StayAliveRule.length; ++i) {
126            if (aliveCount == m_StayAliveRule[i]) {
                 stayAlive = true;
128              break;
               }
130          }
             if (!stayAlive) {
132            data[h][w] = GameData.STATE_DYING;
             }
134        } else if (GameData.STATE_GROWING == curCellState) {
             boolean grow = false;
136          for (int i = 0; i < m_BornRule.length; ++i) {
               if (aliveCount == m_BornRule[i]) {
138              grow = true;
```

```
              break;
            }
          }
          if (grow) {
            data[h][w] = GameData.STATE_ALIVE;
          } else {
            data[h][w] = GameData.STATE_DYING;
          }
        } else if (GameData.STATE_DYING == curCellState) {
          boolean stayAlive = false;
          for (int i = 0; i < m_StayAliveRule.length; ++i) {
            if (aliveCount == m_StayAliveRule[i]) {
              stayAlive = true;
              break;
            }
          }
          if (!stayAlive) {
            data[h][w] = GameData.STATE_DEAD;
          } else {
            data[h][w] = GameData.STATE_GROWING;
          }
        } else if (GameData.STATE_DEAD == curCellState) {
          boolean born = false;
          for (int i = 0; i < m_BornRule.length; ++i) {
            if (aliveCount == m_BornRule[i]) {
              born = true;
              break;
            }
          }
          if (born) {
            data[h][w] = GameData.STATE_GROWING;
          }
        }
      }
    }
  }
}
```

../src/GameRule2.java

6.6 ControlPanel.java

```
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileNotFoundException;
```

```
7  import java.io.PrintWriter;
   import java.util.Observable;
9  import java.util.Observer;
   import java.util.Scanner;
11
   import javax.swing.ButtonGroup;
13 import javax.swing.JButton;
   import javax.swing.JCheckBox;
15 import javax.swing.JFileChooser;
   import javax.swing.JLabel;
17 import javax.swing.JPanel;
   import javax.swing.JRadioButton;
19 import javax.swing.JTextField;
   import javax.swing.filechooser.FileNameExtensionFilter;
21
   public class ControlPanel extends JPanel implements Observer
      {
23
     private CellularAutomataFrame gf;
25   private GameData m_GameData;
     private JButton m_StartButton;
27   private JButton m_StopButton;
     private JButton m_StepButton;
29   private JTextField m_WidthTextField;
     private JTextField m_HeightTextField;
31   private JButton m_ResizeButton;
     private JButton m_ClearButton;
33   private JButton m_RandomButton;
     private JButton m_LoadButton;
35   private JButton m_SaveButton;
     private JRadioButton m_Ruleset0Button;
37   private JRadioButton m_Ruleset1Button;
     private JCheckBox m_WrapCheckbox;
39
     public ControlPanel(CellularAutomataFrame _gf) {
41     gf = _gf;

43     setLayout(new GridLayout(20, 1, 2, 2));

45     m_StartButton = new JButton("Start");
       m_StartButton.addActionListener(new ActionListener() {
47       public void actionPerformed(ActionEvent arg0) {
           gf.getGraphicPanel().getAnimationTimer().start();
49         m_StartButton.setEnabled(false);
           m_StopButton.setEnabled(true);
51         m_StepButton.setEnabled(false);
           m_WidthTextField.setEnabled(false);
53         m_HeightTextField.setEnabled(false);
           m_ResizeButton.setEnabled(false);
```

```
55          m_ClearButton.setEnabled(false);
            m_RandomButton.setEnabled(false);
57          m_LoadButton.setEnabled(false);
            m_SaveButton.setEnabled(false);
59          m_Ruleset0Button.setEnabled(false);
            m_Ruleset1Button.setEnabled(false);
61          m_WrapCheckbox.setEnabled(false);
            gf.getGraphicPanel().setIsRunning(true);
63        }
        });
65
        m_StopButton = new JButton("Stop");
67      m_StopButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent arg0) {
69          gf.getGraphicPanel().getAnimationTimer().stop();
            m_StartButton.setEnabled(true);
71          m_StopButton.setEnabled(false);
            m_StepButton.setEnabled(true);
73          m_WidthTextField.setEnabled(true);
            m_HeightTextField.setEnabled(true);
75          m_ResizeButton.setEnabled(true);
            m_ClearButton.setEnabled(true);
77          m_RandomButton.setEnabled(true);
            m_LoadButton.setEnabled(true);
79          m_SaveButton.setEnabled(true);
            m_Ruleset0Button.setEnabled(true);
81          m_Ruleset1Button.setEnabled(true);
            m_WrapCheckbox.setEnabled(true);
83          gf.getGraphicPanel().setIsRunning(false);
          }
85      });

87      m_StepButton = new JButton("Step");
        m_StepButton.addActionListener(new ActionListener() {
89        public void actionPerformed(ActionEvent arg0) {
            m_GameData.step();
91        }
        });
93
        m_ClearButton = new JButton("Clear");
95      m_ClearButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent arg0) {
97          m_GameData.clear();
          }
99      });

101     m_RandomButton = new JButton("Randomize");
        m_RandomButton.addActionListener(new ActionListener() {
103       public void actionPerformed(ActionEvent arg0) {
```

```
        m_GameData.randomize();
      }
    });

    m_LoadButton = new JButton("Load");
    m_LoadButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
        String filename = "";
        JFileChooser chooser = new JFileChooser();
        File workingDirectory = new File(System.getProperty("
user.dir"));
        chooser.setCurrentDirectory(workingDirectory);
        FileNameExtensionFilter filter = new
FileNameExtensionFilter(
            "PNG files", "png");
        chooser.setFileFilter(filter);
        int returnVal = chooser.showOpenDialog(null);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
          filename = chooser.getSelectedFile().getName();

          File file = chooser.getSelectedFile();
          m_GameData.load(file);
        }
      }
    });

    m_SaveButton = new JButton("Save");
    m_SaveButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
        String filename = "";
        JFileChooser chooser = new JFileChooser();
        File workingDirectory = new File(System.getProperty("
user.dir"));
        chooser.setCurrentDirectory(workingDirectory);
        FileNameExtensionFilter filter = new
FileNameExtensionFilter(
            "PNG files", "png");
        chooser.setFileFilter(filter);
        int returnVal = chooser.showOpenDialog(null);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
          File outfile = chooser.getSelectedFile();
          m_GameData.save(outfile);
          // if (outfile.exists()) {
          // System.out.println(outfile + " already exists.")
;
          // }
          //
          // // -- PrintWriter is used for writing text files
          // PrintWriter writer = null;
```

```
            // try {
149         // writer = new PrintWriter(outfile);
            // } catch (FileNotFoundException e) {
151         // System.out.println("Cannot wrap " + outfile.
    getName()
            // + " with a PrintWriter");
153         // }

155       }
        }
157   });

159   // -- each radio button needs it's own action listener
      m_Ruleset0Button = new JRadioButton("Ruleset 0", true);
161   m_Ruleset0Button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
163       m_GameData.setRuleNum(0);
        }
165   });
      m_Ruleset1Button = new JRadioButton("Ruleset 1", true);
167   m_Ruleset1Button.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
169       m_GameData.setRuleNum(1);
        }
171   });

173   // -- create a ButtonGroup to make the buttons work as a
    mutually
      // exclusive (only one will
175   // ever be selected) set
      ButtonGroup group = new ButtonGroup();
177   group.add(m_Ruleset0Button);
      group.add(m_Ruleset1Button);
179
      // m_Ruleset0Button.setSelected(true);
181   // m_Ruleset1Button.setSelected(false);

183   // -- boolean variable is the default state of the button
      m_WrapCheckbox = new JCheckBox("Edge Wrap", true);
185   m_WrapCheckbox.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
187       if (m_WrapCheckbox.isSelected()) {
            m_GameData.setBorderType(GameData.BORDER_WRAP);
189       } else {
            m_GameData.setBorderType(GameData.BORDER_DONTCARE);
191       }

193     }
      });
```

```
195
    JLabel widthLabel = new JLabel("Width:");
197 m_WidthTextField = new JTextField(3);
    JLabel heightLabel = new JLabel("Height:");
199 m_HeightTextField = new JTextField(3);
    m_ResizeButton = new JButton("Resize");
201 m_ResizeButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent arg0) {
203     try {
          int width = Integer.parseInt(m_WidthTextField.
    getText());
205       int height = Integer.parseInt(m_HeightTextField.
    getText());
          m_GameData.resize(height, width);
207     } catch (NumberFormatException e) {
        }
209   }
    });
211
    add(m_StartButton);
213 add(m_StopButton);
    add(m_StepButton);
215 add(widthLabel);
    add(m_WidthTextField);
217 add(heightLabel);
    add(m_HeightTextField);
219 add(m_ResizeButton);
    add(m_ClearButton);
221 add(m_RandomButton);
    add(m_LoadButton);
223 add(m_SaveButton);
    add(m_Ruleset0Button);
225 add(m_Ruleset1Button);
    add(m_WrapCheckbox);
227
    m_StartButton.setEnabled(true);
229 m_StopButton.setEnabled(false);
    m_StepButton.setEnabled(true);
231 m_WidthTextField.setEnabled(true);
    m_HeightTextField.setEnabled(true);
233 m_ResizeButton.setEnabled(true);
    m_ClearButton.setEnabled(true);
235 m_RandomButton.setEnabled(true);
    m_LoadButton.setEnabled(true);
237 m_SaveButton.setEnabled(true);
    m_Ruleset0Button.setEnabled(true);
239 m_Ruleset1Button.setEnabled(true);
    m_WrapCheckbox.setEnabled(true);
241 gf.getGraphicPanel().setIsRunning(false);
```

```
  }

  public Dimension getPreferredSize() {
    return new Dimension(100, 500);
  }

  public void setGameData(GameData gameData) {
    m_GameData = gameData;
    m_GameData.addObserver(this);
    update(m_GameData, null);
  }

  @Override
  public void update(Observable o, Object arg) {
    if (m_GameData.getRuleNum() == 0) {
      m_Ruleset0Button.setSelected(true);
    } else if (m_GameData.getRuleNum() == 1) {
      m_Ruleset1Button.setSelected(true);
    }
    if (m_GameData.getBorderType() == GameData.BORDER_WRAP) {
      m_WrapCheckbox.setSelected(true);
    } else if (m_GameData.getBorderType() == GameData.
   BORDER_DONTCARE) {
      m_WrapCheckbox.setSelected(false);
    }
    m_WidthTextField.setText(Integer.toString(m_GameData.
   getWidth()));
    m_HeightTextField.setText(Integer.toString(m_GameData.
   getHeight()));
  }

}
```

../src/ControlPanel.java

6.7 GraphicPanel.java

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
```

```
12 import java.util.Observable;
   import java.util.Observer;
14
   import javax.swing.JPanel;
16 import javax.swing.Timer;

18 // -- need to extend JPanel so that we can override some of
   //    the default methods -- JPanel inherits from AWT
      Container
20 //    (can hold Components) which inherits from AWT Component
   //    (can be displayed on a screen)
22 public class GraphicPanel extends JPanel implements Observer
      {

24   private static final int MIN_RECT_WIDTH = 1;
     private CellularAutomataFrame gf;
26   private GameData m_GameData;
     private float m_RectWidth;
28   private float m_RectHeight;
     private boolean m_IsRunning;
30
     private Timer animationTimer = null;
32
     public Timer getAnimationTimer() {
34     return animationTimer;
     }
36
     public boolean isRunning() {
38     return m_IsRunning;
     }
40
     public void setIsRunning(boolean isRunning) {
42     this.m_IsRunning = isRunning;
     }
44
     public GraphicPanel(CellularAutomataFrame _gf) {
46     super();

48     gf = _gf;

50     this.setBackground(Color.gray);

52     // -- The JPanel can have a mouse listener if desired
       this.addMouseListener(new MouseListener() {
54
         public void mouseClicked(MouseEvent arg0) {
56         if (!m_IsRunning) {
             m_GameData.flipStateAt((int)(arg0.getY() /
      m_RectHeight),
```

```
58                (int)(arg0.getX() / m_RectWidth));
          }
60        }

62        public void mouseEntered(MouseEvent arg0) {
            // Do not handle
64        }

66        public void mouseExited(MouseEvent arg0) {
            // Do not handle
68        }

70        public void mousePressed(MouseEvent arg0) {
            // Do not handle
72        }

74        public void mouseReleased(MouseEvent arg0) {
            // Do not handle
76        }

78      });
        // -- The JPanel can have a mouse listener if desired
80      this.addMouseMotionListener(new MouseMotionListener() {

82        public void mouseDragged(MouseEvent arg0) {
            if (!m_IsRunning) {
84            m_GameData.flipStateAt((int)(arg0.getY() /
    m_RectHeight),
                (int)(arg0.getX() / m_RectWidth));
86          }
          }
88
          public void mouseMoved(MouseEvent e) {
90        }

92      });

94      // -- Timer will generate an event every 10mSec once it
    is started
        // First parameter is the delay in mSec, second is the
    ActionListener
96      animationTimer = new Timer(10,
        // -- ActionListener for the timer event
98          new ActionListener() {
              public void actionPerformed(ActionEvent arg0) {
100             m_GameData.step();
              }
102         }
```

```
104       );
106   }
108   // -- this override sets the desired size of the JPanel
       which is
      // used by some layout managers -- default desired size is
       0,0
110   // which is, in general, not good -- will pull from layout
       manager
      public Dimension getPreferredSize() {
112     return new Dimension(50, 50);
      }
114
      // -- this override is where all the painting should be
       done.
116   // DO NOT call it directly. Rather, call repaint() and let
       the
      // event handling system decide when to call it
118   // DO NOT put graphics function call elsewhere in the code,
       although
      // legal, it's bad practice and could destroy the integrity
       of the
120   // display
      public void paint(Graphics g) {
122     // -- the base class paintComponent(g) method ensures
        // the drawing area will be cleared properly. Do not
124     // modify any attributes of g prior to sending it to
        // the base class
126     super.paintComponent(g);

128     // -- for legacy reasons the parameter comes in as type
       Graphics
        // but it is really a Graphics2D object. Cast it up since
         the
130     // Graphics2D class is more capable
        Graphics2D g2d = (Graphics2D) g;
132
        int h = this.getHeight();
134     int w = this.getWidth();
        m_RectWidth = Math.max(MIN_RECT_WIDTH, (float)w / (float)
       m_GameData.getWidth());
136     m_RectHeight = Math.max(MIN_RECT_WIDTH, (float)h / (float
       )m_GameData.getHeight());

138     g2d.setColor(Color.gray);
        g2d.fill(new Rectangle(0, 0, w, h));
140     byte[][] data = m_GameData.getData();
        for (int i = 0; i < m_GameData.getHeight(); ++i) {
```

```
142         final int startY = (int)Math.round(i * m_RectHeight);
            final int height = (int)Math.round((i+1) * m_RectHeight
    )-startY;
144         for (int j = 0; j < m_GameData.getWidth(); ++j) {
              final int startX = (int)Math.round(j * m_RectWidth);
146           final int width = (int)Math.round((j+1) * m_RectWidth
    )-startX;

148           if (GameData.STATE_ALIVE == data[i][j]) {
                g2d.setColor(Color.red);
150           } else if (GameData.STATE_DYING == data[i][j]) {
                g2d.setColor(Color.blue);
152           } else if (GameData.STATE_DEAD == data[i][j]) {
                g2d.setColor(Color.black);
154           } else if (GameData.STATE_GROWING == data[i][j]) {
                g2d.setColor(Color.yellow);
156           } else if (GameData.STATE_DONTCARE == data[i][j]) {
                g2d.setColor(Color.white);
158           }

160           g2d.fill(new Rectangle(startX, startY, width, height)
    );
            }
162       }

164   }

166   public void setGameData(GameData gameData) {
        this.m_GameData = gameData;
168     m_GameData.addObserver(this);
        update(m_GameData, null);
170   }

172   @Override
      public void update(Observable o, Object arg) {
174     repaint();
      }
176
}
```

../src/GraphicPanel.java

6.8 ImageReadWrite.java

```
1 import java.awt.Graphics2D;
  import java.awt.image.BufferedImage;
3 import java.io.File;
  import java.io.IOException;
```

```
5
import javax.imageio.ImageIO;
7
public class ImageReadWrite {
9
  public static byte[][] ImageRead(String filename) {
11  // -- read input image
    File infile = new File(filename);
13  return ImageRead(infile);
  }
15
  public static byte[][] ImageRead(File infile) {
17
    try {
19    BufferedImage bi = ImageIO.read(infile);

21    // -- separate image into RGB components
      byte red[][] = new byte[bi.getHeight()][bi.getWidth()];
23    byte grn[][] = new byte[bi.getHeight()][bi.getWidth()];
      byte blu[][] = new byte[bi.getHeight()][bi.getWidth()];
25    for (int i = 0; i < red.length; ++i) {
        for (int j = 0; j < red[i].length; ++j) {
27        red[i][j] = (byte) (bi.getRGB(j, i) >> 16 & 0xFF);
          grn[i][j] = (byte) (bi.getRGB(j, i) >> 8 & 0xFF);
29        blu[i][j] = (byte) (bi.getRGB(j, i) & 0xFF);
        }
31    }

33    return grn;

35  } catch (IOException e) {
      System.out.println("image I/O error");
37    return null;
    }
39  }

41  public static byte[][][] ImageReadC(String filename) {

43    try {

45      // -- read input image
        File infile = new File(filename);
47      BufferedImage bi = ImageIO.read(infile);

49      // -- separate image into RGB components
        byte img[][][] = new byte[3][bi.getHeight()][bi.
   getWidth()];
51      for (int i = 0; i < img[0].length; ++i) {
          for (int j = 0; j < img[0][i].length; ++j) {
```

```
53            img[0][i][j] = (byte) (bi.getRGB(j, i) >> 16 & 0xFF
    );
              img[1][i][j] = (byte) (bi.getRGB(j, i) >> 8 & 0xFF)
    ;
55            img[2][i][j] = (byte) (bi.getRGB(j, i) & 0xFF);
          }
57      }

59      return img;

61    } catch (IOException e) {
        System.out.println("image I/O error");
63      return null;
      }
65  }

67  public static void renderImage(byte img[][][], Graphics2D
     g2d, int x,
        int y, int w, int h) {
69     BufferedImage bi = new BufferedImage(img[0][0].length,
     img[0].length,
          BufferedImage.TYPE_INT_RGB);
71
      // -- prepare output image
73    for (int i = 0; i < bi.getHeight(); ++i) {
        for (int j = 0; j < bi.getWidth(); ++j) {
75        int pixel = ((int) img[0][i][j] << 16)
              | ((int) img[1][i][j] << 8) | ((int) img[2][i][j
     ]);
77        bi.setRGB(j, i, pixel);
        }
79    }
      g2d.drawImage(bi, x, y, Math.min(w, bi.getWidth()),
81        Math.min(h, bi.getHeight()), 0, 0, bi.getWidth(),
          bi.getHeight(), null);
83
  }
85
  public static void ImageWrite(byte img[][], String filename
     ) {
87    try {
        BufferedImage bi = new BufferedImage(img[0].length, img
     .length,
89           BufferedImage.TYPE_INT_RGB);

91      // -- prepare output image
        for (int i = 0; i < bi.getHeight(); ++i) {
93        for (int j = 0; j < bi.getWidth(); ++j) {
            int val = img[i][j] & 0xff;
```

```
95            int pixel = (val << 16) | (val << 8) | (val);
              bi.setRGB(j, i, pixel);
97          }
          }
99
          // -- write output image
101       File outputfile = new File(filename);
          ImageIO.write(bi, "png", outputfile);
103     } catch (IOException e) {

105     }
    }
107
    public static void ImageWrite(byte img[][], File outputfile
     ) {
109
      try {
111     BufferedImage bi = new BufferedImage(img[0].length, img
    .length,
              BufferedImage.TYPE_INT_RGB);
113
          // -- prepare output image
115       for (int i = 0; i < bi.getHeight(); ++i) {
            for (int j = 0; j < bi.getWidth(); ++j) {
117           int val = img[i][j] & 0xff;
              int pixel = (val << 16) | (val << 8) | (val);
119           bi.setRGB(j, i, pixel);
            }
121       }

123       // -- write output image

125       ImageIO.write(bi, "png", outputfile);
      } catch (Exception e) {
127   }
    }
129
    public static void ImageWriteC(byte img[][][], String
     filename) {
131   try {
        BufferedImage bi = new BufferedImage(img[0][0].length,
133         img[0].length, BufferedImage.TYPE_INT_RGB);

135     // -- prepare output image
        for (int i = 0; i < bi.getHeight(); ++i) {
137       for (int j = 0; j < bi.getWidth(); ++j) {
            int pixel = ((int) img[0][i][j] << 16)
139             | ((int) img[1][i][j] << 8) | ((int) img[2][i][
    j]);
```

```
            bi.setRGB(j, i, pixel);
          }
        }

        // -- write output image
        File outputfile = new File(filename);
        ImageIO.write(bi, "png", outputfile);
      } catch (IOException e) {

      }
    }
}
```

../src/ImageReadWrite.java

6.9 CellularAutomataFrame.java

```

import java.awt.BorderLayout;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JLabel;

import java.util.Observable;
import java.util.Observer;

public class CellularAutomataFrame extends JFrame implements
    Observer {

  private GraphicPanel gp;
  private ControlPanel cp;
  private GameData gameData;

  GraphicPanel getGraphicPanel() {
    return gp;
  }

  public CellularAutomataFrame(int height, int width,
    GameData gameData) {

    setTitle("Cellular Automata");
    // -- add some items to the content pane of the frame
    // JButton okButton = new JButton("OK");
    // frame.add(okButton);
```

```
31      // -- size of the frame: width, height
        setSize(width, height);
33
        // -- center the frame on the screen
35      setLocationRelativeTo(null);

37      // -- shut down the entire application when the frame is
        closed
        // if you don't include this the application will
        continue to
39      // run in the background and you'll have to kill it by
        pressing
        // the red square in eclipse
41      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

43      // -- set the layout manager and add items
        // 5, 5 is the border around the edges of the areas
45      setLayout(new BorderLayout(5, 5));

47      gp = new GraphicPanel(this);
        this.add(gp, BorderLayout.CENTER);
49
        cp = new ControlPanel(this);
51      this.add(cp, BorderLayout.EAST);

53      // -- show the frame on the screen
        setVisible(true);
55      this.gameData = gameData;
        gameData.addObserver(this);
57      gp.setGameData(gameData);
        cp.setGameData(gameData);
59    }

61    public static void main(String[] args) {
        CellularAutomataFrame gf = new CellularAutomataFrame(768,
        1024,
63          new GameData(50, 50));
      }
65
      @Override
67    public void update(Observable arg0, Object arg1) {
      }
69  }
```

../src/CellularAutomataFrame.java