Conway’s Game of Life

Computer Science 1710:
Object-Oriented Programming I

Department of Computer Science
Memorial University of Newfoundland

July 30, 2007
Conway’s game of life

The rules

Implementation
This “game” is not so much a game, as it is a simulation. Ideally, the game is carried out on an infinite two-dimensional array of cells.

The cells are considered either alive or dead. At each step of the game a cell can either stay alive (or stay dead), die, or be reborn. What occurs depends upon the number of living neighbours of the cell during the previous step.

The cell’s neighbours are other cells that are either vertically, horizontally, or diagonally adjacent.
The rules

At each step, we count the number of living neighbours of each cell:

- A live cell with $\leq 1$ neighbours dies (of loneliness)
- A live cell with $\geq 4$ neighbours dies (of overcrowding)
- A dead cell with exactly 3 neighbours becomes alive

If none of the above apply, the cell maintains its current state.

These simple rules lead to very interesting and complex behaviour that has attracted the attention of mathematicians, physicists, biologists, economists, and (of course) computer scientists...
Implementation

**The board:**

We cannot represent the game board using an infinite array. Instead we use an array of some fixed size (e.g. $100 \times 100$).

The borders of the array present a challenge. We handle them simply by not updating any cell that borders on the edge of the array.

**The classes:**

The `GameGUI` class from the stick man game example has been modified to incorporate mouse input and provide a start/stop button. The new class is called `LifeGUI`.

Again, we have not gone far enough in COMP 1710 to fully understand this code. The following slide gives the documentation for the class and its constructor.
/**
A graphical user interface for Conway's game of life (see Life.java). Provides a JFrame for graphical display and the ability to receive mouse input from the user when the JFrame has the focus.

An instance of LifeGUI will call the 'step' method of the Life object every DELAY milliseconds. The 'step' method must be defined as follows:

```java
public void step(Graphics g, int width, int height,
boolean mouseEvent, int x, int y, boolean run);
```

where g is the graphics context of the GameGUI's JComponent, width and height are the dimensions of the JComponent, mouseEvent is true if the user has just pressed the left mouse button with (x, y) being the coordinates of the mouse event in pixels. Final, run indicates that the simulation should proceed.
*/
public class LifeGUI {

    /**
     Constructor.
     @param inLife the instance of Life whose 'step' method will be called every DELAY milliseconds
     @param width the width of the JFrame in pixels
     @param height the height of the JFrame in pixels
     @param title the text to appear in the title bar of the JFrame
     */
    public LifeGUI(Life inLife, int width, int height, String title) {
        ...
    }
}
The Life class creates the LifeGUI and Board objects and manages communication between the two. It also contains the main method (so the program is run by executing java Life.

The following shows the code for Life (import statements and comments removed):

```java
public class Life {
    public Life() {
        board = new Board();
        lifeGUI = new LifeGUI(this, WIDTH, HEIGHT, "The Game of Life");
    }
}
```
Here is the code for the very important `step` method which is repeatedly called by `LifeGUI` and includes information about mouse input (`mouseEvent, x, y`) and indicates whether the rules of the game should be applied (`run`).

```java
public void step(Graphics g, int width, int height,
    boolean mouseEvent, int x, int y, boolean run)
{
    Graphics2D g2d = (Graphics2D) g;

    if (mouseEvent)
        board.setByScreenPos(width, height, x, y, true);

    if (run)
        board.update();

    board.draw(g2d, width, height);
}
```
The remainder of the class consists of the `main` method and the class’s instance fields:

```java
public static void main(String[] args)
{
    new Life();
}

private LifeGUI lifeGUI;
private Board board;
private static final int WIDTH = 600;
private static final int HEIGHT = 600;
```
Finally, we have the `Board` class. The job of this class is to store the contents of the board as a 2-D array.

`Board` has the following fields:

```java
private static final int ROWS = 80;
private static final int COLS = 80;
private boolean[][][] array;
private boolean[][][] tempArray;
```

tempArray will be used in the `update` method. It is needed because `update` determines the next state of the array based on the previous state. Two arrays are required so that cells updated for the next state don’t affect the update of other cells.
/**  
   Constructor. Creates the board array and a temporary array.  
*/
public Board()  
{  
   array = new boolean[ROWS][COLS];  
   tempArray = new boolean[ROWS][COLS];  
}
/**
 * Determines the cell selected by a mouse press on position (x, y) when the board itself is drawn on a width x height surface, then sets the value of this cell to the given value.
 */

public void setByScreenPos(int width, int height, int x, int y, boolean value)
{
    int cellWidth = width / COLS;
    int cellHeight = height / ROWS;

    int r = y / cellHeight;
    int c = x / cellWidth;

    if (r < ROWS && c < COLS)
    {
        array[r][c] = value;
        tempArray[r][c] = value;
    }
}
/**
   Applies the rules of Conway’s Game of Life.
*/
public void update()
{
    // Update all cells not adjacent to the boundaries
    for (int i = 1; i < ROWS−1; i++)
    {
        for (int j = 1; j < COLS−1; j++)
        {
            int liveCount = 0;
            if (array[i−1][j−1]) liveCount++;
            if (array[i−1][j]) liveCount++;
            if (array[i−1][j+1]) liveCount++;
            if (array[i][j−1]) liveCount++;
            if (array[i][j+1]) liveCount++;
            if (array[i+1][j−1]) liveCount++;
            if (array[i+1][j]) liveCount++;
            if (array[i+1][j+1]) liveCount++;
            ...
        }
    }
}
if (array[i][j])
{
    if (liveCount <= 1 || liveCount >= 4)
    // Existing cell: dies unless it is
    // surrounded by exactly 2 or 3 living
    // neighbours.
        tempArray[i][j] = false;
    else
        tempArray[i][j] = true;
} else
{
    if (liveCount == 3)
        // Dead cell: reborn if surrounded by
        // exactly 3 living neighbours.
            tempArray[i][j] = true;
    else
        tempArray[i][j] = false;
}
... big loop ...

// Could now copy tempArray into array. But this is expensive. We swap their references instead.

boolean[][][] tempRef = array;
array = tempArray;
tempArray = tempRef;
}
/**
 * Draw the game board as a grid.
 * @param g2 the graphics context
 * @param width the width of the drawing component
 * @param height the height of the drawing component
 */
public void draw(Graphics2D g2, int width, int height)
{
    int cellWidth = width / COLS;
    int cellHeight = height / ROWS;

    for (int i = 0; i < ROWS; i++)
    {
        for (int j = 0; j < COLS; j++)
        {
            if (array[i][j])
                g2.setColor(Color.BLACK);
            else
                g2.setColor(Color.WHITE);
            Rectangle rect = new Rectangle(j*cellWidth, i*cellHeight,
                                            cellWidth, cellHeight);
            g2.fill(rect);
        }
    }
}
Interesting facts:

- The Game of Life was invented by mathematician John Conway in 1970.
- Initially, the game was run on paper.
- The game is the most well-known instance of a **cellular automaton** — a grid of cells whose states evolve through time using simple rules.
- The game can be set up as general-purpose computer that is able to compute anything that a regular computer can.