Java 程式設計
Conway’s Game of Life

1102065
游竣捷

目錄

程式說明

此程式使用 Java 覆現康威生命遊戲，並透過 Canvas 視覺化程式。
生命遊戲中，對於任意細胞，規則如下：

- 每個細胞有兩種狀態 - 存活或死亡，每個細胞與以自身為中心的周圍八格細胞產生互動（如圖，黑色為存活，白色為死亡）
- 當前細胞為存活狀態時，當周圍的存活細胞低於 2 個時（不包含 2 個），該細胞變成死亡狀態。（模擬生命數量稀少）
- 當前細胞為存活狀態時，當周圍有 2 個或 3 個存活細胞時，該細胞保持原樣。
- 當前細胞為存活狀態時，當周圍有超過 3 個存活細胞時，該細胞變成死亡狀態。（模擬生命數量過多）
- 當前細胞為死亡狀態時，當周圍有 3 個存活細胞時，該細胞變成存活狀態。（模擬繁殖）

開發環境

- **Java-IDE**: IntelliJ IDEA 2023.3.1 (Ultimate Edition)
- **JDK**: openjdk-21.0.1
- **OS**: Windows 11.0

操作方式

1. 於程式左上方輸入盤面大小。
2. 下方操作按鈕依序為 Start, Next, Stop, Reset, Random
3. Start －細胞開始連續演變 (演變週期，程式設定為 333ms)
4. Next － 細胞演變一個週期 (注意若只想查看下一個週期的結果，需先使用 Stop 停止細胞連續演變)。
5. Stop － 於連續演變時，停止演變。
6. Reset － 將目前盤面上的所有細胞清空，回復成乾淨的盤面。
7. Random － 於目前盤面上散佈細胞，細胞位置為隨機生成。
8. 於右上角顯示目前演變的次數。

- 架構說明

使用以下元件進行程式操控、運作。

- JButton 操控程式開始、下一步、暫停、重置、隨機生成。
- JTextField 供使用者輸入數值，設置盤面大小。
- JLabel 顯示目前演變的次數。
- JPanel 用於各元件之排版。
- Cavas 繪製盤面及細胞演變過程。

- 函式庫引用

- import javax.swing.*;
- import javax.swing.event.DocumentEvent;
- import javax.swing.event.DocumentListener;
- import java.awt.Dimension;
- import java.awt.FlowLayout;
- import java.awt.BorderLayout;
- import java.awt.Font;
- import java.awt.Graphics;
- import java.awt.Color;
- import java.awt.event.MouseAdapter;
- import java.awt.event.MouseEvent;
- import java.awt.event.WindowAdapter;
- import java.awt.event.WindowEvent;

- 系統分析

- **盤面繪製(Canvas Paint)**

根據目前 Canvas 畫布的寬度除以欲繪製的格數，得出每格(Cell Size)大小。再透過每格大小，與 Canvas 畫布的高度，推算出可以畫多高。
Board Width = 使用者輸入的格數
Cell Size = Canvas 寬度除以 Board Width
Board Height = Canvas 高度除以 Cell Size

假如使用者輸入 < 10 的格數或不是數字，將 Board Width 設為 10。

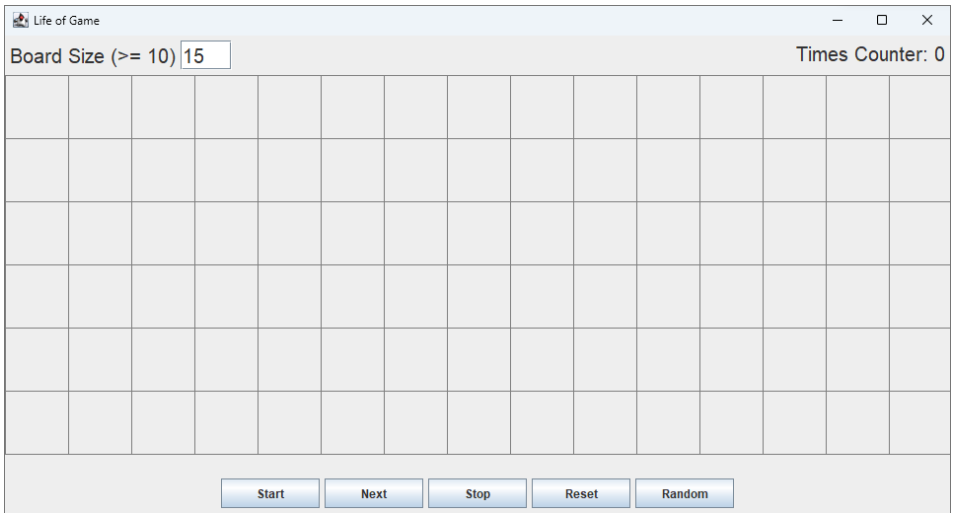
使用迴圈繪製直線，迴圈次數 i 為 0 到 Board Width，如圖(一)(a)

g.drawLine(i * CellSize, 0, i * CellSize, BoardHeight * CellSize);

圖(一)(a)

同理以繪製直線的方式，再將橫線繪製出。
迴圈次數 i 為 0 到 Board Height，如圖(一)(b)

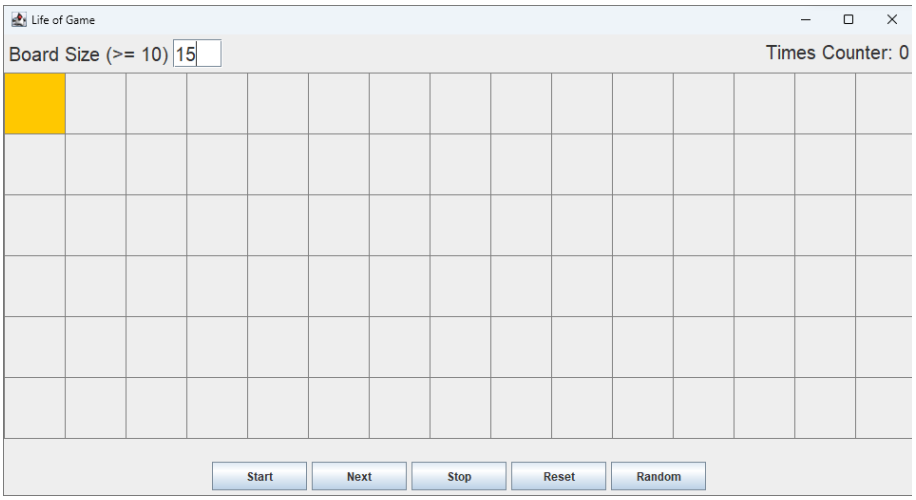
g.drawLine(0, i * CellSize, BoardWidth * CellSize, i * CellSize);



圖(一)(b)

使用布林二維陣列(Borad)儲存盤面狀態，陣列大小依 Board Width 和 Board Height 而定，false 為該方格無細胞、true 則有，
透過巢狀迴圈讀取此二維陣列，假如該方格為 true 則把那一格依照 Cell Size 大小，使用顏色塗滿方格。如圖(二)，Board[0][0]為 true，就將該方格，塗滿橘色，

```
g.fillRect(i * CellSize, j * CellSize, CellSize, CellSize);
```



圖(二)

註: Cavas 於繪製時會先繪畫出在 Board 中為 true 的方格顏色，再繪畫出格線，避免格線被覆蓋過去。

■ 盤面方格滑鼠點擊(Cell Click Listener)

對 Canvas 增加滑鼠監聽事件，取得 Cavas 在被滑鼠點擊時的 x 座標、y 座標，被分別除以 Cell Size，即可得出在二維陣列 Board 中，是哪一格被點擊，並將那一格的狀態設為與之相反的狀態。
如為 false 則設成 true。

最後透過 Canvas.repaint()重新繪製出新的盤面盤面。

■ 盤面狀態演變(Update Borad)

為避免修改原資料(Board)造成判斷錯誤，所以需要一個新的相同大小的盤面(new Board)來儲存修改資訊後的盤面。
透過巢狀迴圈讀取 Board 資料，並計算每一格的周圍其他細胞的數量。

如果 Board 該格為 true，且周圍細胞數量為 2 或為 3(小於 2 孤單死，大於 3 擁擠死)，則該格在 new Board 為 true。

如果 Board 該格為 false，且周圍細胞數量為 3 時會復活，則該格在 new Board 為 true。

最後將 new Board 覆蓋至 Board 並將目前演變的總次數加一。

■ 細胞鄰居數量計算(Count Neighbors)

計算以該格座標(x, y)為中心的九宮格的細胞鄰居數量。
透過巢狀迴圈計算，如圖(三) ，周圍座標於 Board 中為 true 的數量有多少即代表鄰居數量。

(x-1,y-1)	(x-0,y-1)	(x+1,y-1)
(x-1,y-0)	(x, y)	(x+1,y-0)
(x-1,y+1)	(x-0,y+1)	(x+1,y+1)

圖(三)

■ 連續執行盤面演變(Start Simulation)

透過 Timer 函式庫，建立 simulationTimer 物件，週期為 333ms，並開始執行。
重複執行以下兩個步驟。

1. 盤面狀態演變
2. 重新繪製盤面

■ 停止盤面狀態演變(Stop Simulation)

當 simulationTimer 物件不為空(null)且為正在執行中(is Running)，
則呼叫 simulationTimer 的停止函式。

■ 執行一次盤面演變(Once Simulation)

執行以下三個步驟

1. 停止盤面狀態演變
2. 盤面狀態演變
3. 重新繪製盤面

相似於連續執行盤面演變(Start Simulation)，差別在於沒有透過 Timer 去執行因此只會執行一次。

■ 重置盤面狀態(Reset Board)

執行以下四個步驟

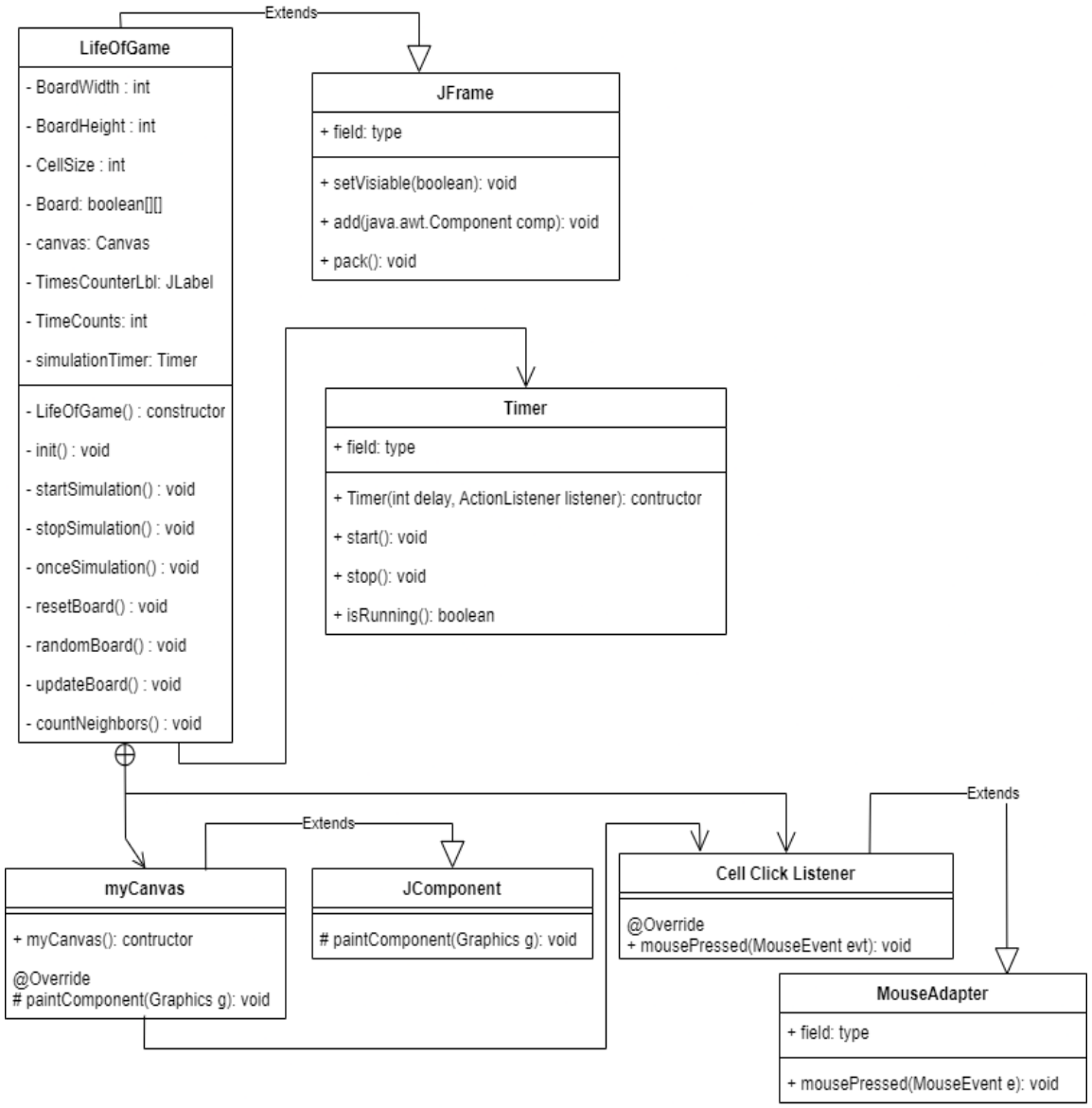
1. 停止盤面狀態演變
2. 目前演變的總次數歸零
3. 將目前 Board 內所有的值皆設為 false
4. 重新繪製盤面

■ 隨機生成盤面(Random Board)

執行以下四個步驟

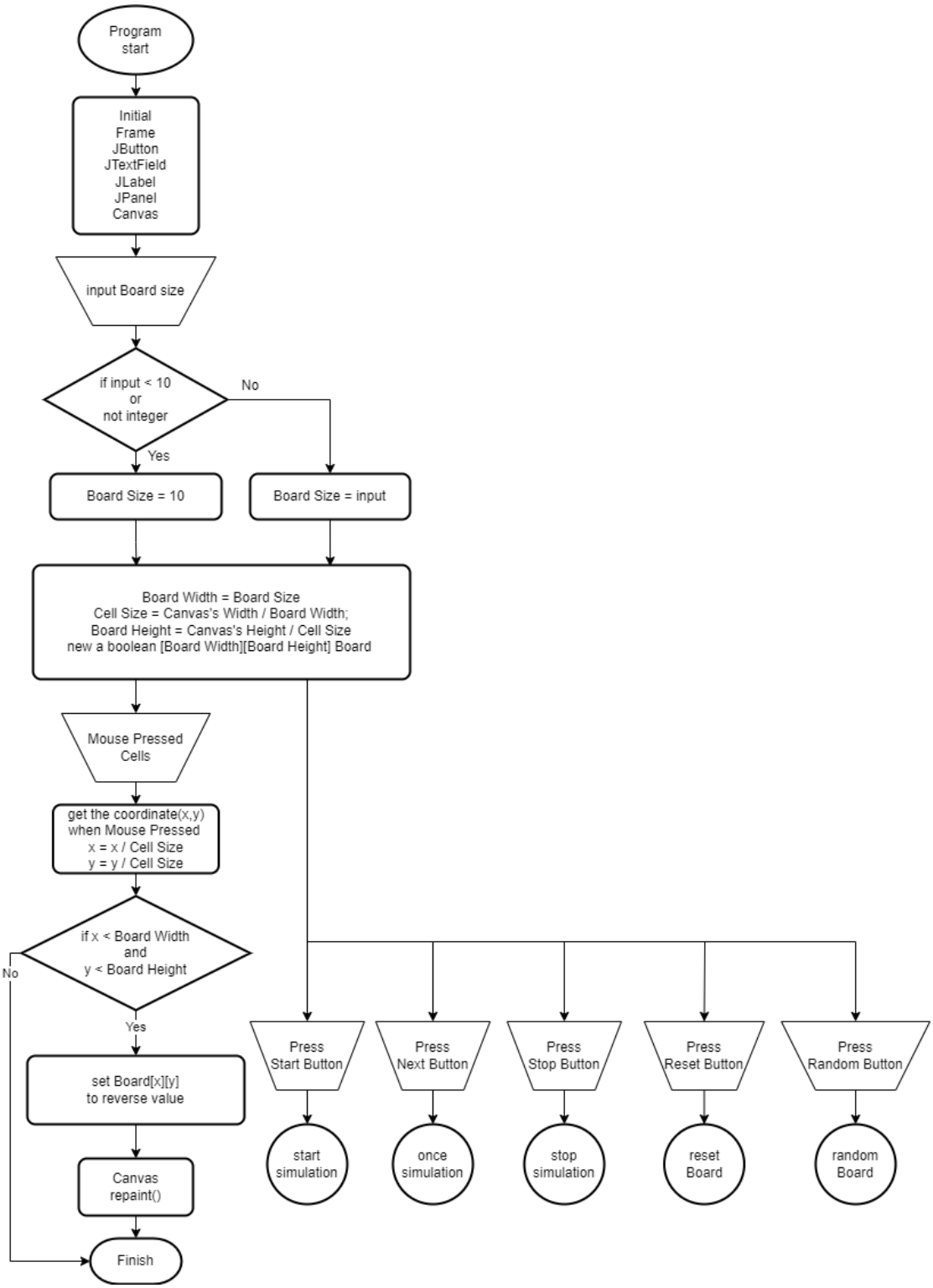
1. 停止盤面狀態演變
2. 重置盤面狀態
3. 於重置後的 Board 中，將所有值，亂數設定為 true
4. 重新繪製盤面

- 程式類別圖

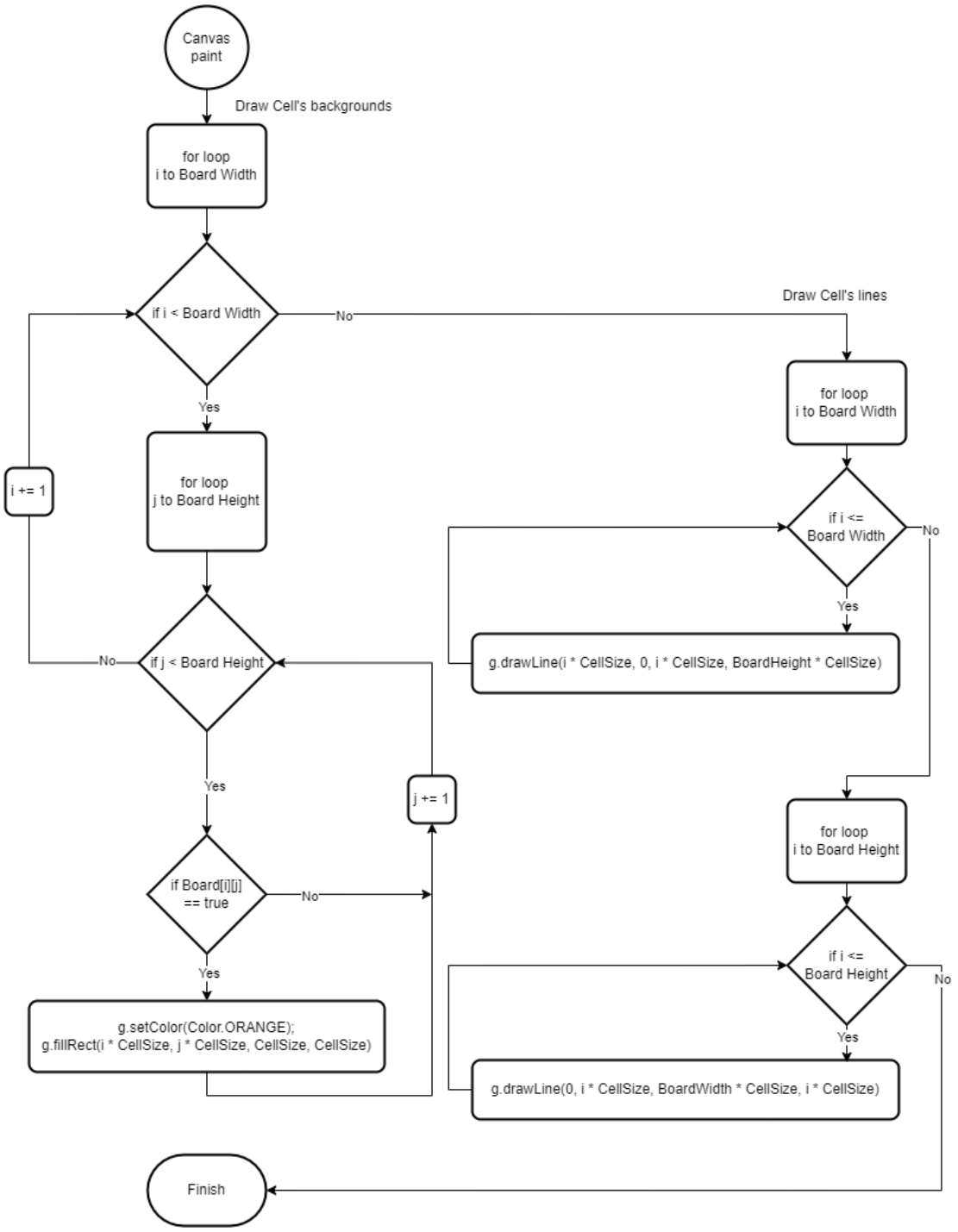

Extends
LifeOfGame
- BoardWidth : int
- BoardHeight : int
- CellSize : int
- Board: boolean[][]
- canvas: Canvas
- TimesCounterLbl: JLabel
- TimeCounts: int
- simulationTimer: Timer
- LifeOfGame() : constructor
- init() : void
- startSimulation() : void
- stopSimulation() : void
- onceSimulation() : void
- resetBoard() : void
- randomBoard() : void
- updateBoard() : void
- countNeighbors() : void
JFrame
+ field: type
+ setVisiable(boolean): void
+ add(java.awt.Component comp): void
+ pack(): void
Timer
+ field: type
+ Timer(int delay, ActionListener listener): contructor
+ start(): void
+ stop(): void
+ isRunning(): boolean
Extends
myCanvas
+ myCanvas(): contructor
@Override
paintComponent(Graphics g): void
JComponent
paintComponent(Graphics g): void
Cell Click Listener
@Override
+ mousePressed(MouseEvent evt): void
Extends
MouseAdapter
+ field: type
+ mousePressed(MouseEvent e): void

圖(四)

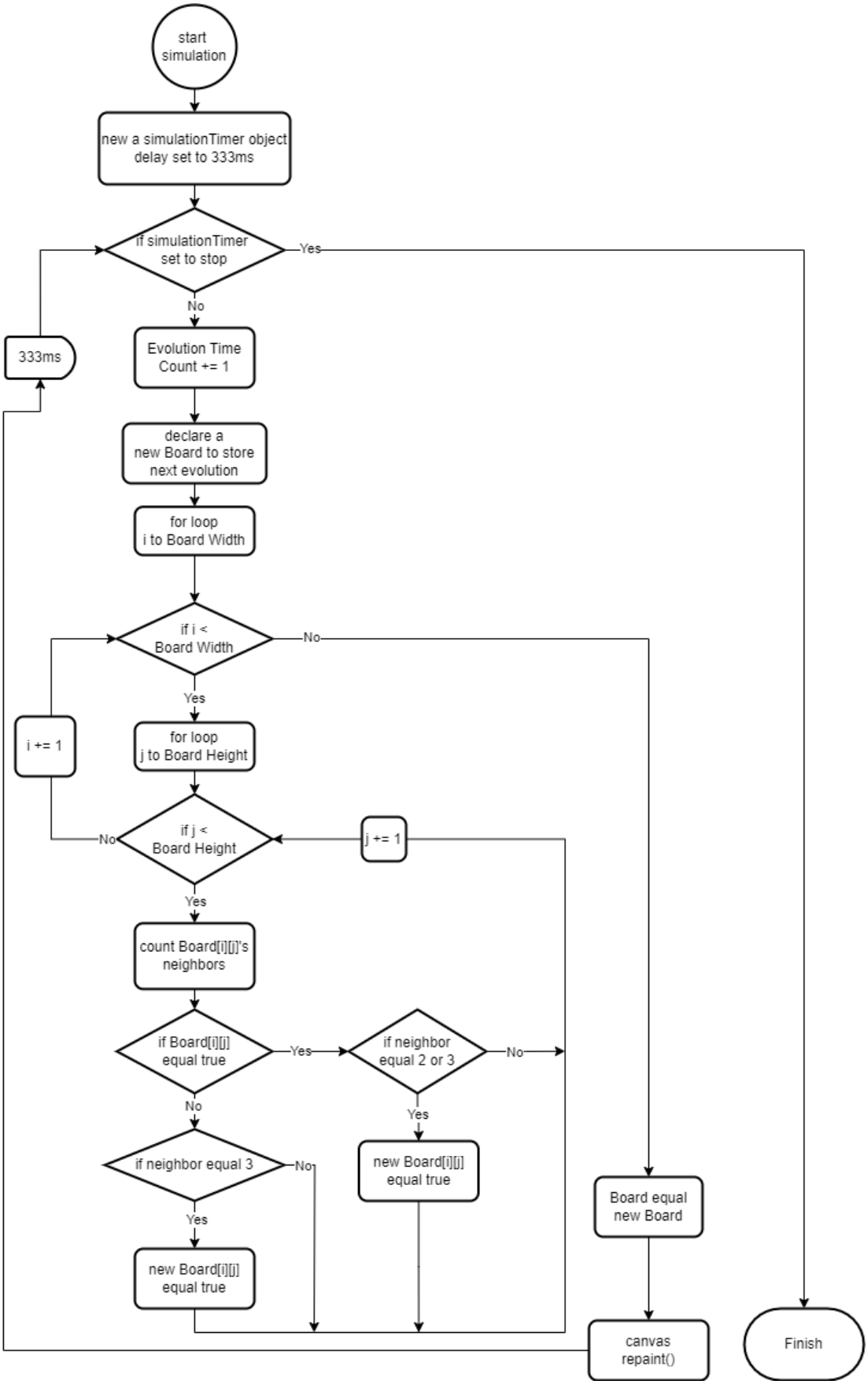
- 執行流程圖


Program start
Initial
Frame
JButton
JTextField
JLabel
JPanel
Canvas
input Board size
if input < 10
or
not integer
No
Yes
Board Size = 10
Board Size = input
Board Width = Board Size
Cell Size = Canvas's Width / Board Width;
Board Height = Canvas's Height / Cell Size
new a boolean [Board Width][Board Height] Board
Mouse Pressed
Cells
get the coordinate(x,y)
when Mouse Pressed
x = x / Cell Size
y = y / Cell Size
if x < Board Width
and
y < Board Height
No
Yes
set Board[x][y]
to reverse value
Canvas
repaint()
Finish
Press
Start Button
Press
Next Button
Press
Stop Button
Press
Reset Button
Press
Random Button
start
simulation
once
simulation
stop
simulation
reset
Board
random
Board

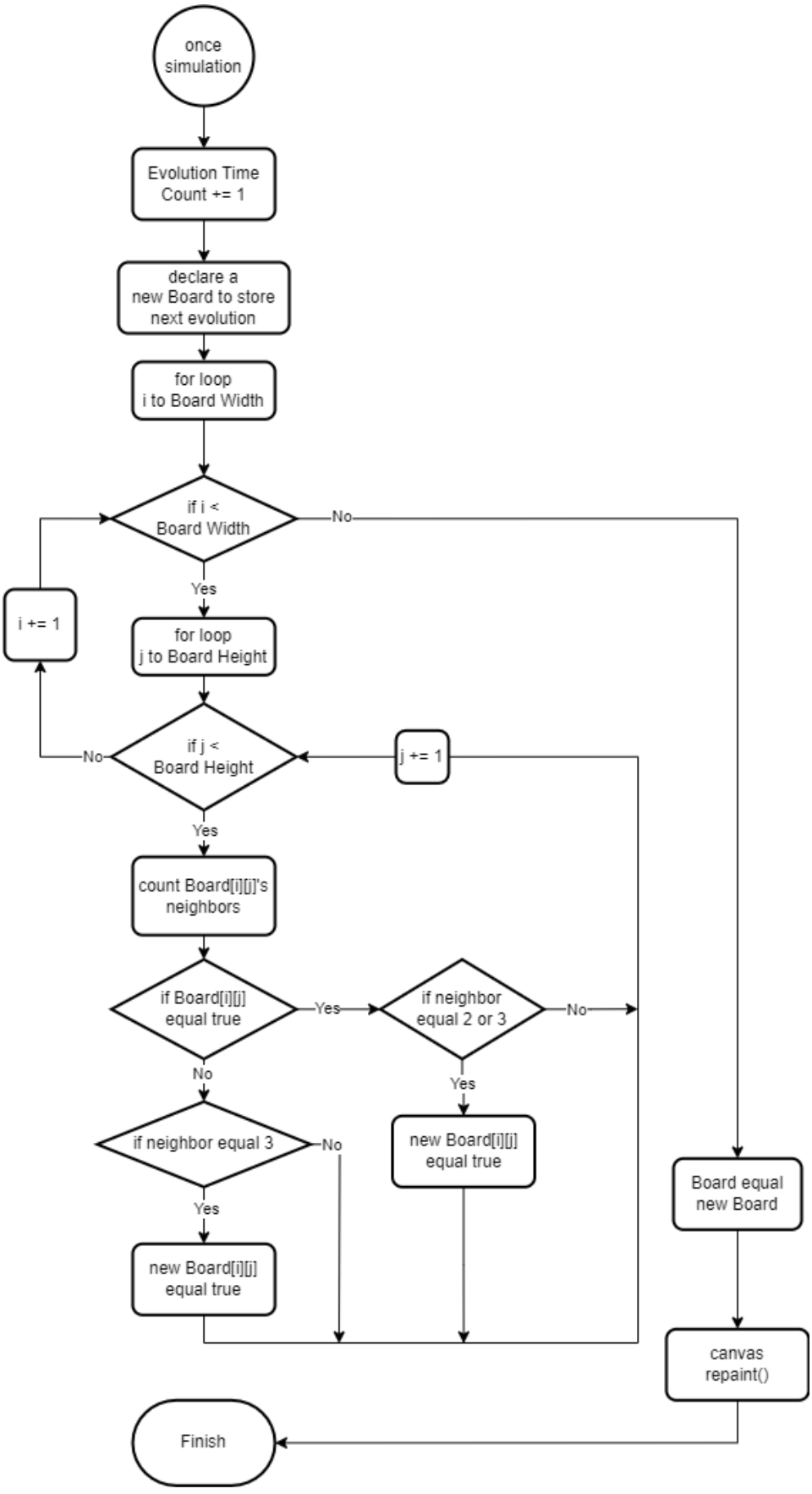
圖(五)(a)


Canvas paint
Draw Cell's backgrounds
for loop i to Board Width
if i < Board Width
No
Yes
for loop j to Board Height
i += 1
if j < Board Height
No
Yes
if Board[i][j] == true
No
Yes
g.setColor(Color.ORANGE);
g.fillRect(i * CellSize, j * CellSize, CellSize, CellSize)
j += 1
Draw Cell's lines
for loop i to Board Width
if i <= Board Width
No
Yes
g.drawLine(i * CellSize, 0, i * CellSize, BoardHeight * CellSize)
for loop i to Board Height
if i <= Board Height
No
Yes
g.drawLine(0, i * CellSize, BoardWidth * CellSize, i * CellSize)
Finish

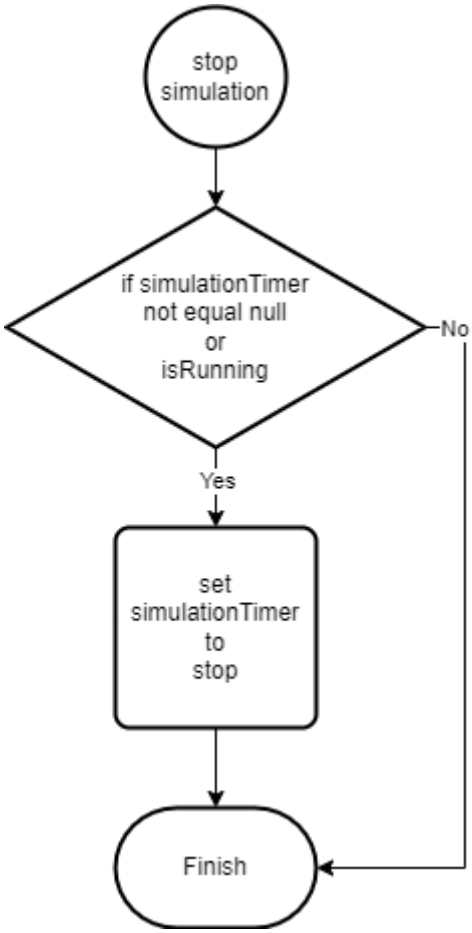
圖(五)(b)


start simulation
new a simulationTimer object delay set to 333ms
if simulationTimer set to stop
Yes
No
333ms
Evolution Time Count += 1
declare a new Board to store next evolution
for loop i to Board Width
if i < Board Width
No
Yes
i += 1
for loop j to Board Height
if j < Board Height
No
j += 1
Yes
count Board[i][j]'s neighbors
if Board[i][j] equal true
Yes
if neighbor equal 2 or 3
No
Yes
No
if neighbor equal 3
No
Yes
new Board[i][j] equal true
new Board[i][j] equal true
Board equal new Board
canvas repaint()
Finish

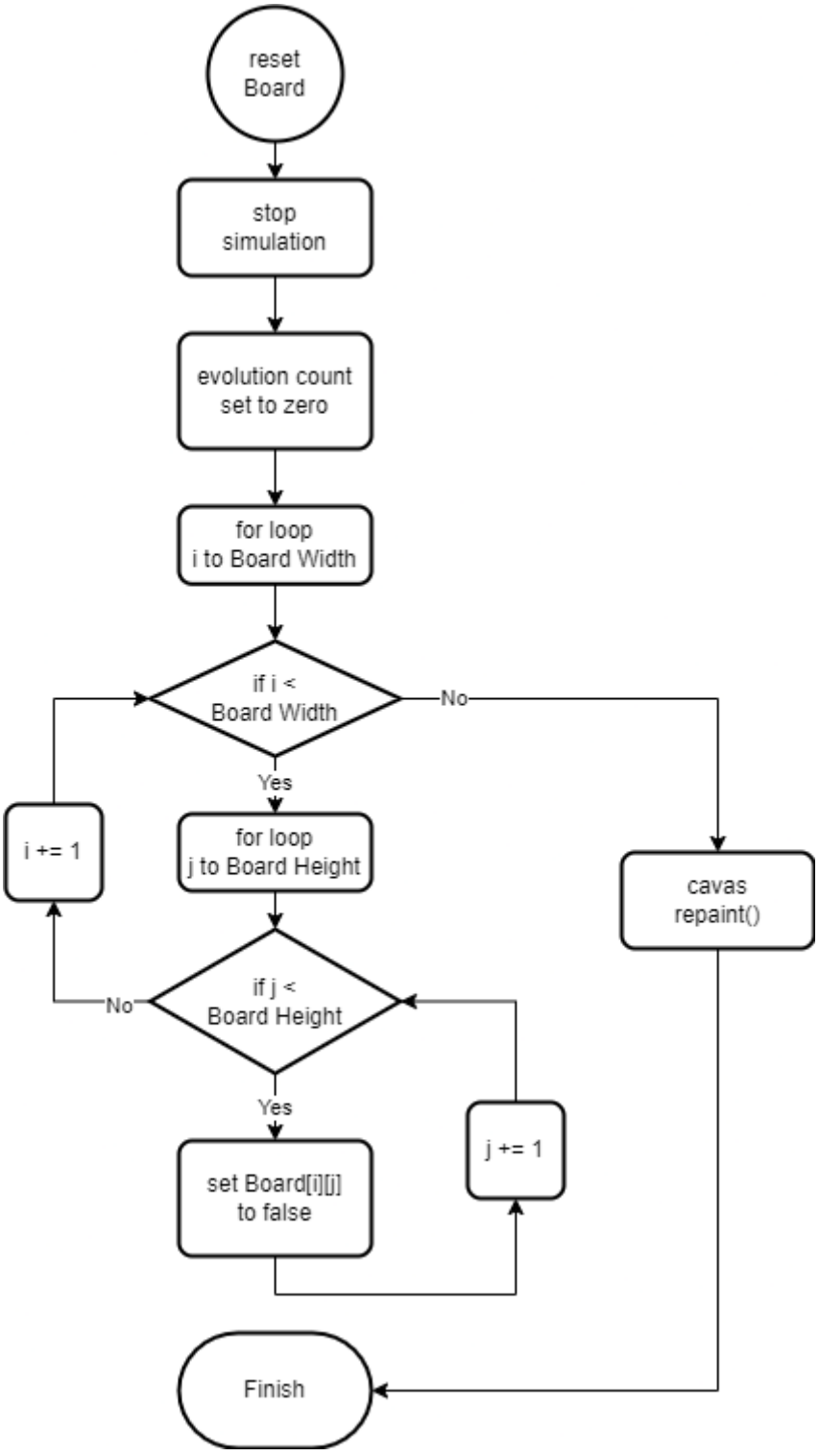
圖(五)(c)


once simulation
Evolution Time Count += 1
declare a new Board to store next evolution
for loop i to Board Width
if i < Board Width
No
Yes
i += 1
for loop j to Board Height
if j < Board Height
No
j += 1
Yes
count Board[i][j]'s neighbors
if Board[i][j] equal true
Yes
if neighbor equal 2 or 3
No
Yes
No
if neighbor equal 3
No
Yes
new Board[i][j] equal true
new Board[i][j] equal true
Board equal new Board
canvas repaint()
Finish

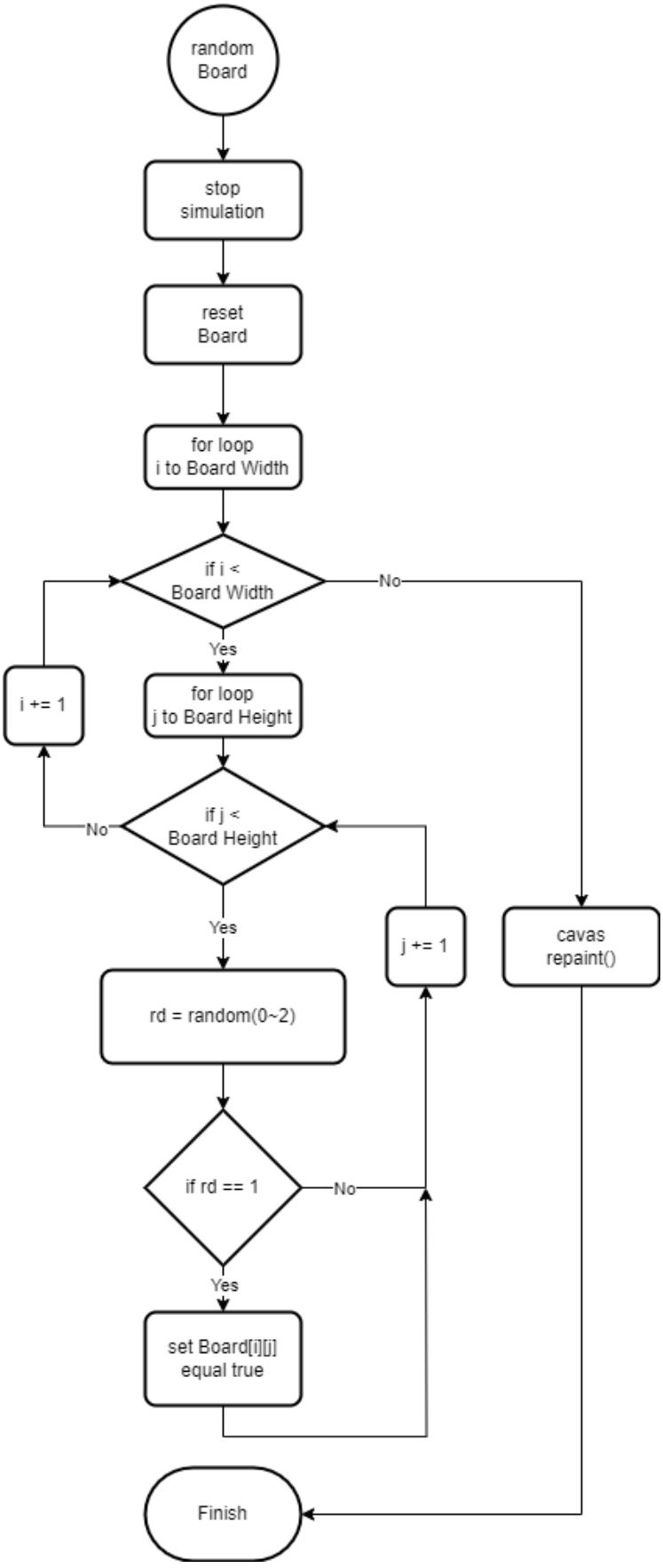
圖(五)(d)


stop
simulation
if simulationTimer
not equal null
or
isRunning
No
Yes
set
simulationTimer
to
stop
Finish

圖(五)(e)


reset
Board
stop
simulation
evolution count
set to zero
for loop
i to Board Width
if i <
Board Width
No
Yes
i += 1
for loop
j to Board Height
cavas
repaint()
if j <
Board Height
No
Yes
j += 1
set Board[i][j]
to false
Finish

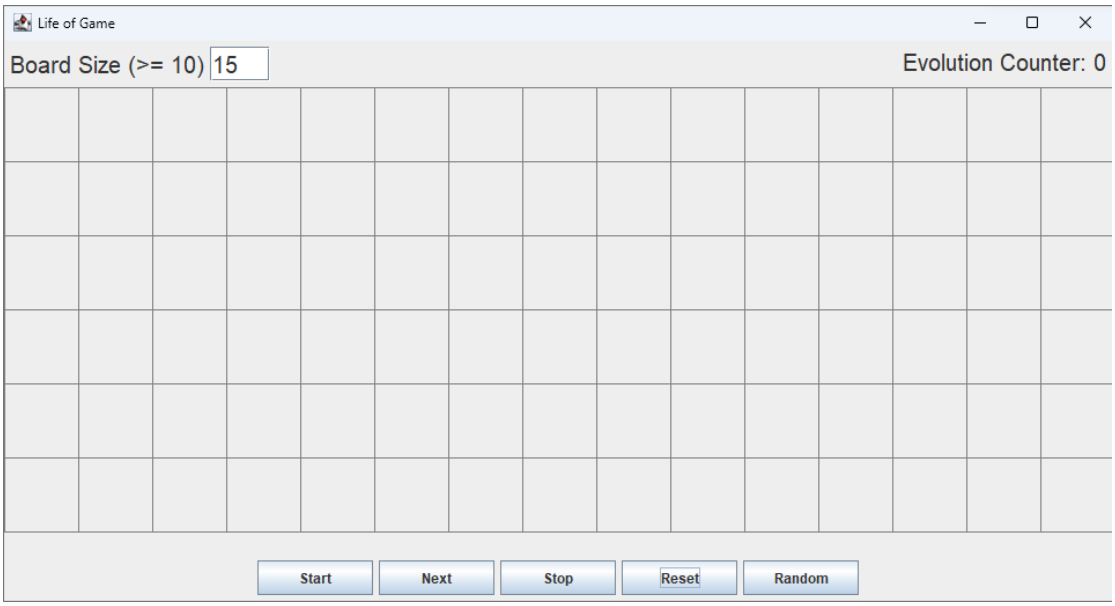
圖(五)(f)


random Board
stop simulation
reset Board
for loop i to Board Width
if i < Board Width
No
Yes
i += 1
for loop j to Board Height
if j < Board Height
No
Yes
j += 1
cavas repaint()
rd = random(0~2)
if rd == 1
No
Yes
set Board[i][j] equal true
Finish

圖(五)(g)

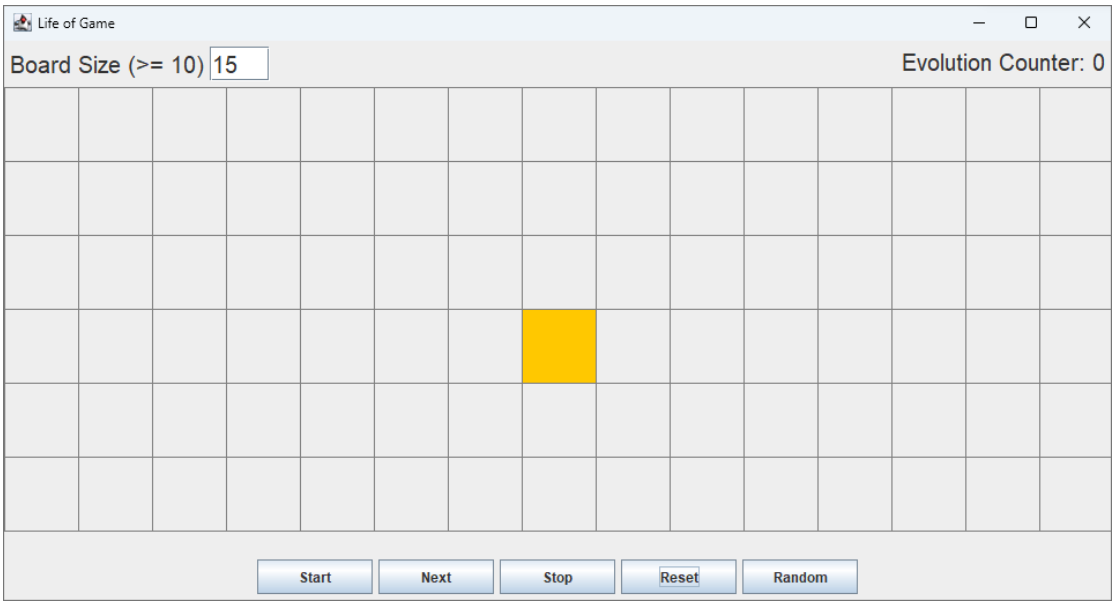
- **運行結果**

設置盤面大小，輸入 Board Size 為 15，繪製出寬度為 15 的盤面，如圖(六)(a)



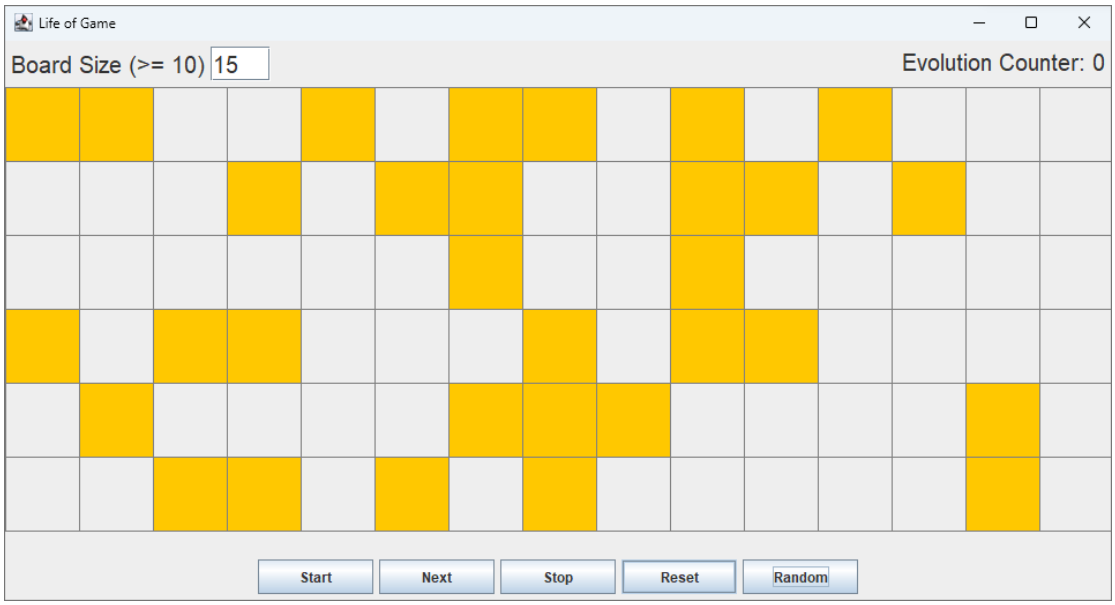
圖(六)(a)

滑鼠於盤面(8,4)位置點擊後，該方格被激活，如圖(六)(b)



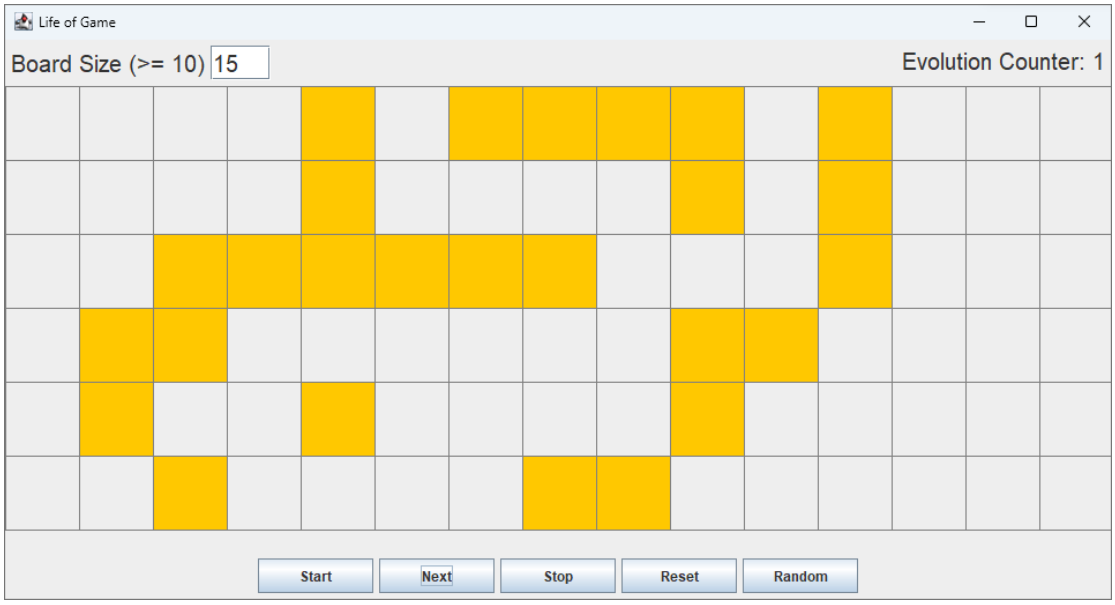
圖(六)(b)

按下 Random 按鈕，隨機生成盤面，如圖(六)(c)



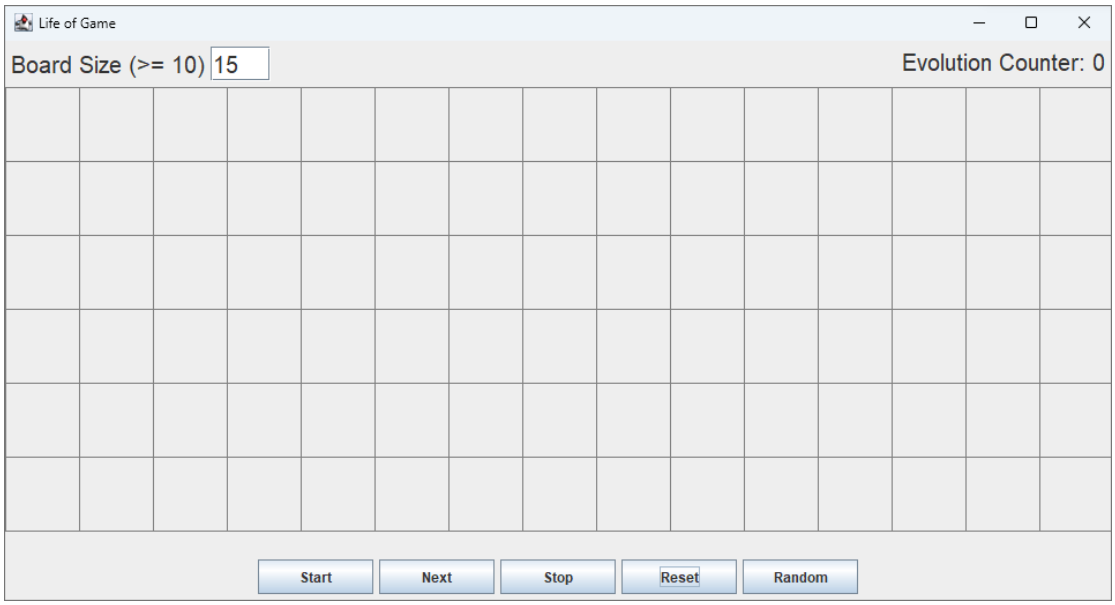
圖(六)(c)

按下 Next，執行演變一次，如圖(六)(d)



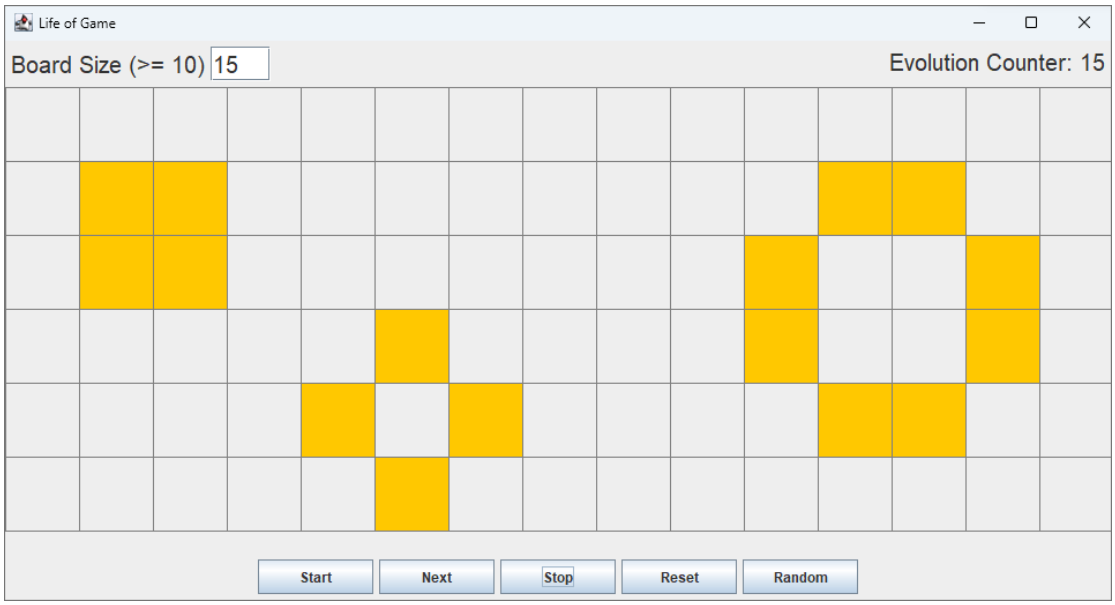
圖(六)(d)

按下 Reset 後，重置盤面，Evolution Counter 歸零。如圖(六)(e)



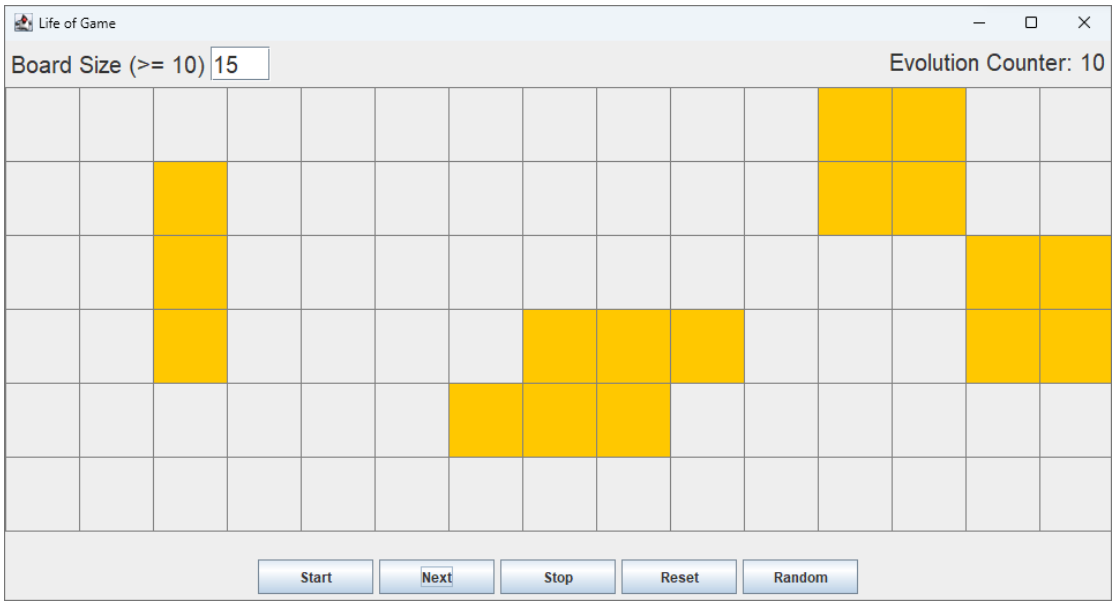
圖(六)(e)

穩定狀態，如圖(六)(f)

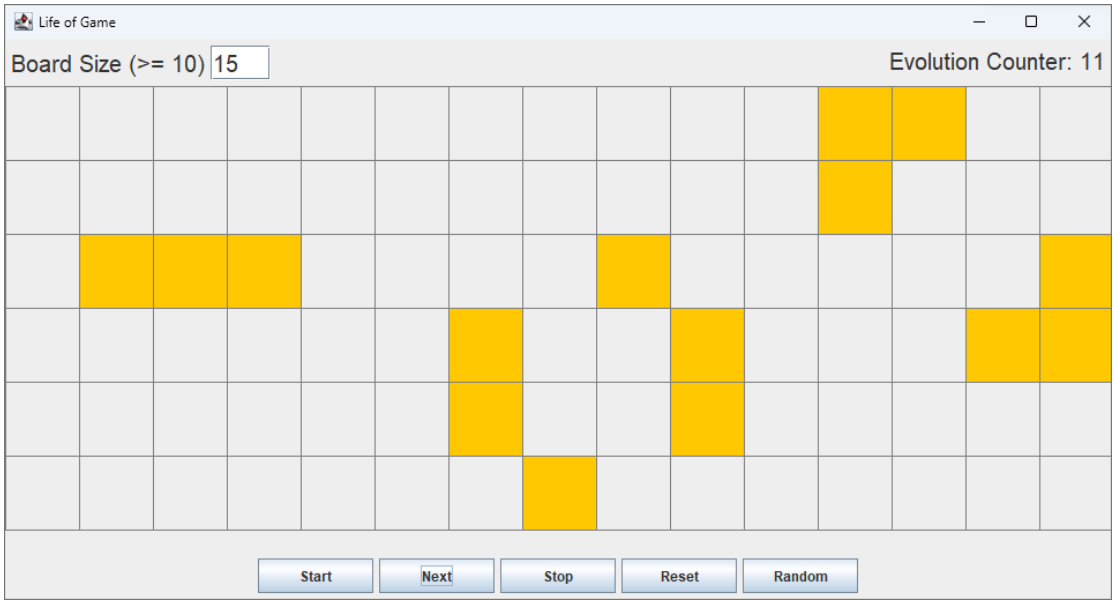


圖(六)(f)

震盪狀態，如圖(六)(g)，圖(六)(h)

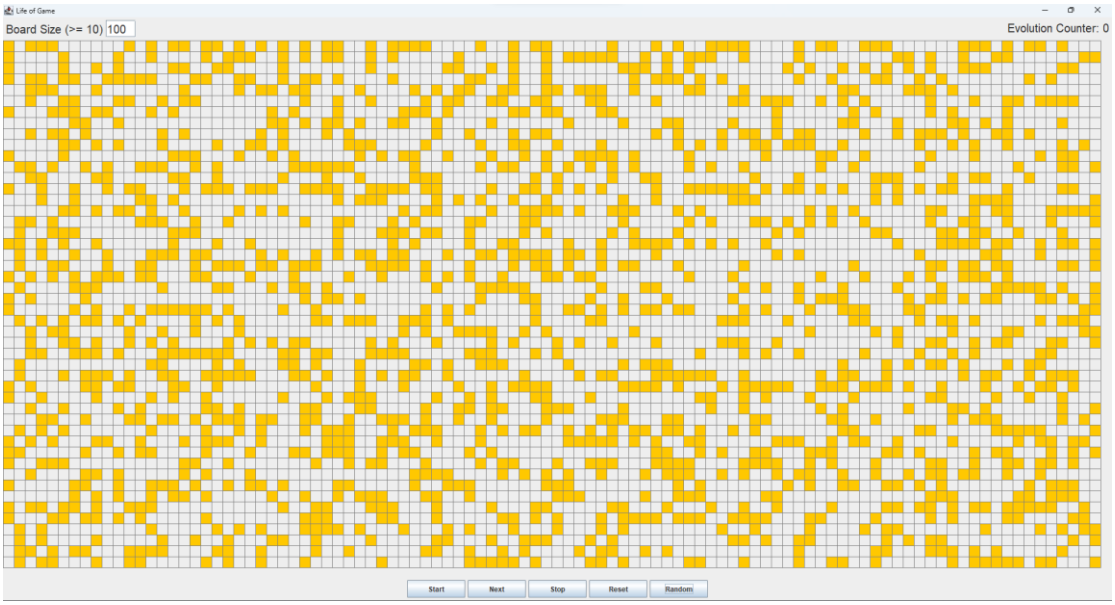


圖(六)(g)

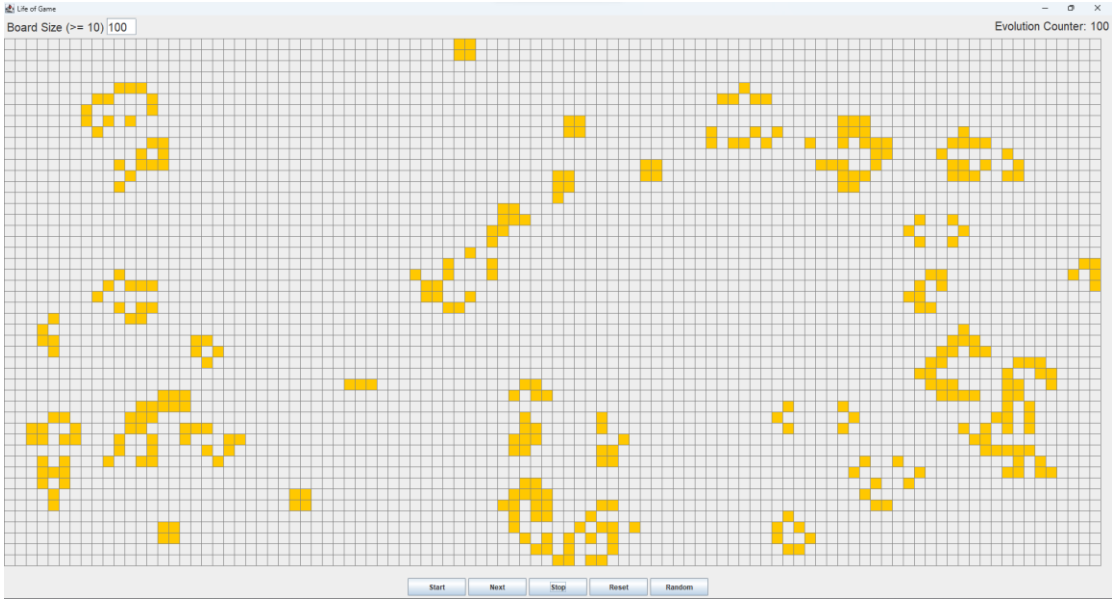


圖(六)(h)

盤面連續演變 100 次，起始盤面如圖(七)(a)，結果為圖(七)(b)



圖(七)(a)



圖(七)(b)

- 原始程式碼

```
import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import static java.lang.Integer.parseInt;

public class LifeOfGame extends JFrame {
    int BoardSize;
    int BoardHeight;
    int BoardWidth;
    int CellSize;
    boolean[][] Board;
    myCanvas canvas;
    JLabel evolutionCounterLbl;
    int evolutionCounts = 0;
    Timer simulationTimer;

    public static void main(String[] args) {
        new LifeOfGame().setVisible(true);
    }

    LifeOfGame() {
        init();
    }

    void init() {
```

```
        this.setTitle("Life of Game");
        this.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                super.windowClosing(e);
                System.exit(0);
            }
        });

        JButton startButton = new JButton("Start");
        startButton.setPreferredSize(new Dimension(100,
30));
        startButton.addActionListener(e ->
startSimulation());

        JButton nextButton = new JButton("Next");
        nextButton.setPreferredSize(new Dimension(100,
30));
        nextButton.addActionListener(e ->
onceSimulation());

        JButton stopButton = new JButton("Stop");
        stopButton.setPreferredSize(new Dimension(100,
30));
        stopButton.addActionListener(e ->
stopSimulation());

        JButton resetButton = new JButton("Reset");
        resetButton.setPreferredSize(new Dimension(100,
30));
        resetButton.addActionListener(e ->
resetBoard());

        JButton randomButton = new JButton("Random");
        randomButton.setPreferredSize(new
Dimension(100, 30));
        randomButton.addActionListener(e ->
randomBoard());
```

```
        JTextField BoardSizeTF = new JTextField();
        BoardSizeTF.setColumns(3);
        BoardSizeTF.setFont(new Font("Ubuntu",
Font.PLAIN, 20));

BoardSizeTF.getDocument().addDocumentListener(new
DocumentListener() {
            @Override
            public void insertUpdate(DocumentEvent e) {
                resetBoard();
                updateBoardSize();
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                resetBoard();
                updateBoardSize();
            }

            @Override
            public void changedUpdate(DocumentEvent e) {
                resetBoard();
                updateBoardSize();
            }

            private void updateBoardSize() {
                try {
                    BoardSize =
parseInt(BoardSizeTF.getText());
                    if (BoardSize <= 10) {
                        BoardSize = 10;
                    }



                } catch (Exception ex) {
                    BoardSize = 10;
```

```
                }
                finally {
                    BoardWidth = BoardSize;
                    CellSize =
(int)Math.round((double)(canvas.getWidth()) /
(double)(BoardWidth));
                    BoardHeight = canvas.getHeight() /
CellSize;
                    Board = new
boolean[BoardWidth][BoardHeight];
                }
            }
        });

        JLabel BoardSizeLbl = new JLabel("Board Size
(>= 10)");
        BoardSizeLbl.setFont(new Font("Ubuntu",
Font.PLAIN, 20));

        evolutionCounterLbl = new JLabel("Evolution
Counter: " + evolutionCounts);
        evolutionCounterLbl.setFont(new Font("Ubuntu",
Font.PLAIN, 20));

        JPanel buttonsPanel = new JPanel();
        buttonsPanel.add(startButton);
        buttonsPanel.add(nextButton);
        buttonsPanel.add(stopButton);
        buttonsPanel.add(resetButton);
        buttonsPanel.add(randomButton);

        JPanel settingPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
        settingPanel.add(BoardSizeLbl);
        settingPanel.add(BoardSizeTF);

        JPanel evolutionCounterPanel = new JPanel(new
FlowLayout(FlowLayout.LEFT));
```

```
        evolutionCounterPanel.add(evolutionCounterLbl);

        JPanel HeaderPanel = new JPanel(new
BorderLayout());
        HeaderPanel.add(settingPanel,
BorderLayout.WEST);
        HeaderPanel.add(evolutionCounterPanel,
BorderLayout.EAST);

        canvas = new myCanvas();
        canvas.setPreferredSize(new Dimension(800,
800));

        this.add(HeaderPanel, BorderLayout.NORTH);
        this.add(buttonsPanel, BorderLayout.SOUTH);
        this.add(canvas);

        this.pack();
    }

    private class myCanvas extends JComponent {
        public myCanvas() {
            addMouseListener(new CellClickListener());
        }

        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);

            for (int i = 0; i < BoardWidth; i++) {
                for (int j = 0; j < BoardHeight; j++) {
                    if (Board[i][j]) {
                        g.setColor(Color.ORANGE);
                        g.fillRect(i * CellSize, j *
CellSize, CellSize, CellSize);
                    }
                }
            }
```

```
            g.setColor(Color.GRAY);
            for (int i = 0; i <= BoardWidth; i++) {
                g.drawLine(i * CellSize, 0, i * CellSize,
BoardHeight * CellSize);
            }
            for(int i = 0; i <= BoardHeight; i++){
                g.drawLine(0, i * CellSize, BoardWidth *
CellSize, i * CellSize);
            }
        }
    }

    private class CellClickListener extends
MouseAdapter {
        @Override
        public void mousePressed(MouseEvent evt) {
            int x = evt.getX() / CellSize;
            int y = evt.getY() / CellSize;

            if(x<BoardWidth && y < BoardHeight){
                Board[x][y] = !Board[x][y];
                canvas.repaint();
            }
        }
    }

    private void startSimulation() {

        if(simulationTimer == null
|| !simulationTimer.isRunning()){
            simulationTimer = new Timer(333, e -> {
                updateBoard();
                canvas.repaint();
            });
            simulationTimer.start();
        }
```

```
    }

    private void stopSimulation() {
        if (simulationTimer != null &&
simulationTimer.isRunning()) {
            simulationTimer.stop();
        }
    }
    private void onceSimulation() {
        stopSimulation();
        updateBoard();
        canvas.repaint();
    }
    private void resetBoard() {
        stopSimulation();
        evolutionCounts = 0;
        evolutionCounterLbl.setText("Evolution Counter:
" + evolutionCounts);
        for (int i = 0; i < BoardWidth; i++) {
            for (int j = 0; j < BoardHeight; j++) {
                Board[i][j] = false;
            }
        }
        canvas.repaint();
    }

    private void randomBoard() {
        stopSimulation();
        resetBoard();
        for (int i = 0; i < BoardWidth; i++) {
            for (int j = 0; j < BoardHeight; j++) {
                int rd = (int) (Math.random() * 3);
                if (rd == 1)
                    Board[i][j] = true;
            }
        }
        canvas.repaint();
    }
```

```
    private void updateBoard() {
        evolutionCounts += 1;
        evolutionCounterLbl.setText("Evolution Counter:
" + evolutionCounts);
        boolean[][] newBoard = new
boolean[BoardWidth][BoardHeight];

        for (int i = 0; i < BoardWidth; i++) {
            for (int j = 0; j < BoardHeight; j++) {
                int neighbors = countNeighbors(i, j);

                if (Board[i][j]) {
                    if (neighbors == 2 || neighbors == 3)
                        newBoard[i][j] = true;
                } else {
                    if (neighbors == 3)
                        newBoard[i][j] = true;
                }
            }
        }

        Board = newBoard;
    }

    private int countNeighbors(int x, int y) {
        int count = 0;

        for (int i = -1; i <= 1; i++)
            for (int j = -1; j <= 1; j++)
                if (!(i == 0 && j == 0)) {
                    int otherX = x + i;
                    int otherY = y + j;

                    if (otherX >= 0 && otherX <
BoardWidth && otherY >= 0 && otherY < BoardHeight) {
                        if (Board[otherX][otherY]) {
                            count++;
```

```
                        }
                    }
                }
            return count;
        }
    }
```