Übung 06

Arbeitsaufwand insgesamt: 5h

Inhaltsverzeichnis

Teil 1 – Conway's Game of Life

Bei dieser Übung geht es um die Implementierung von Conway's Game of Life mittels JavaFX.

Lösungsidee:

In der Übung wurde bereits der Grundstein für diese Applikation gelegt und die Anzeige des Boards bzw. der Zellen war bereits voll funktionstüchtig.

Grundsätzlich fehlen nur noch die Buttons sowie die Logik zwischen den Generationen.

Folgende Buttons sind notwendig:

- Step: berechnet die nächste Generation von Punkten und stellt diese dar
- Run: Generationen berechnen sich periodisch neu und werden laufend aktualisiert
- Stop: Die automatisierte Berechnung von Generationen wird gestoppt
- Reset: Setzt die Zellen auf den Stand vor dem Betätigen des Run-Buttons zurück
- Clear: Setzt alle Zellen auf den „Tot"-Status
- Save: Speichert den momentanen Zustand der Zellen in eine Datei
- Open: Stellt einen zuvor gespeicherten Zustand aus einer Datei wieder her

Beim Berechnen einer neuen Generation muss für jede Zelle die Anzahl der Nachbarn in der 8er-Nachbarschaft bestimmt werden und daraus folgend ermittelt werden was mit dieser Zelle passieren soll.

Eine lebende Zelle stirbt, wenn sie zu wenige (eine oder weniger) Nachbarn oder zu viele (vier oder mehr) Nachbarn hat. Ansonsten lebt sie weiter.

Eine tote Zelle kann „zum Leben erweckt werden", wenn sie genau drei Nachbarn hat.

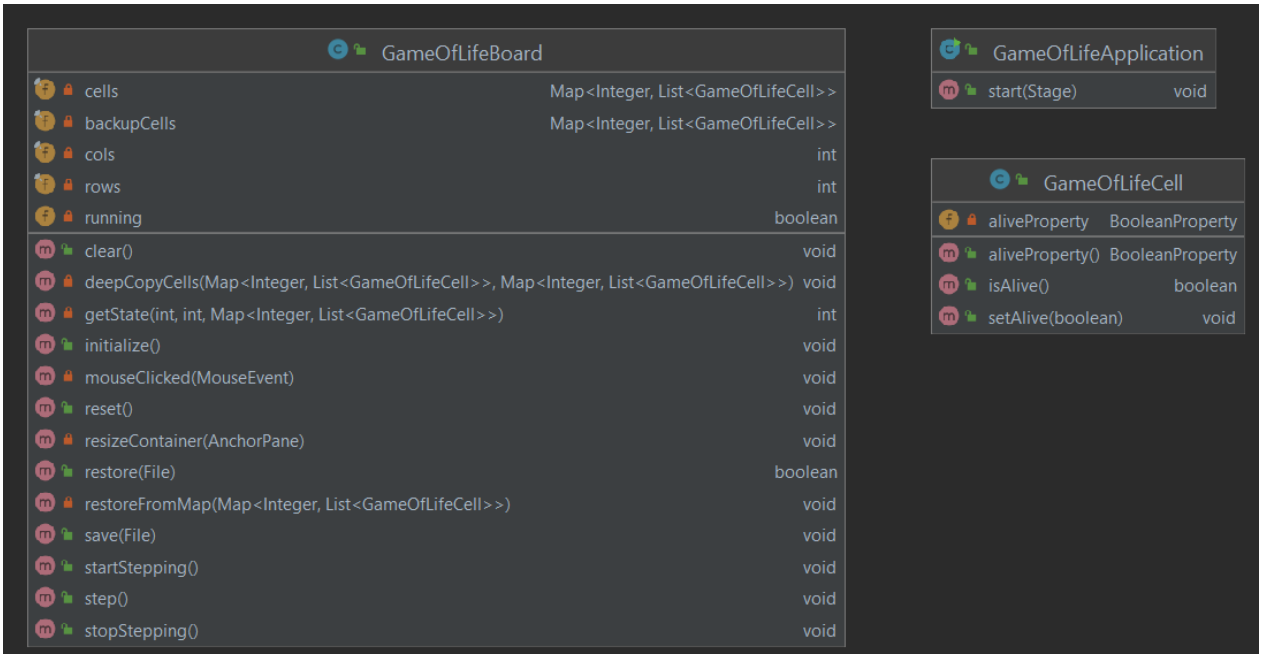
Beim Setzen der neuen Information ist es sehr wichtig, dass das aktuelle Board nicht direkt überschrieben wird, da sonst das Ergebnis beeinflusst wird und nicht mehr stimmt. Deswegen würde ich zuerst eine Kopie der Datenstruktur (Map) anlegen und darauf achten, dass keine Referenzen mitgenommen werden.

Bei der automatischen Berechnung wird in periodischen Abständen einfach die Methode „Step" ausgeführt. Wichtig dabei ist, dass dies in einem eigenen Thread gekapselt wird, sodass die UI nicht blockiert wird. Bevor der Thread gestartet wird, sollte zudem noch für „Reset" das momentane Feld gespeichert werden.

Beim Speichern des Zustands bietet es sich an das CSV Format zu verwenden, wobei die lebenden Felder als 1 und die toten Felder als 0 gespeichert werden. Beim Einlesen des Files würde ich auf gleiches Format bzw. Zellenanzahl überprüfen, sodass nur valide Zustände wiederhergestellt werden können. Andernfalls kann ein Error angezeigt werden.

Zusätzlich ist bei Aktionen, die das UI verändern, darauf zu achten, dass diese über „PLatform.runLater" ausgeführt werden.

UML


GameOfLifeBoard
cells Map<Integer, List<GameOfLifeCell>>
backupCells Map<Integer, List<GameOfLifeCell>>
cols int
rows int
running boolean
clear() void
deepCopyCells(Map<Integer, List<GameOfLifeCell>>, Map<Integer, List<GameOfLifeCell>>) void
getState(int, int, Map<Integer, List<GameOfLifeCell>>) int
initialize() void
mouseClicked(MouseEvent) void
reset() void
resizeContainer(AnchorPane) void
restore(File) boolean
restoreFromMap(Map<Integer, List<GameOfLifeCell>>) void
save(File) void
startStepping() void
step() void
stopStepping() void
GameOfLifeApplication
start(Stage) void
GameOfLifeCell
aliveProperty BooleanProperty
aliveProperty() BooleanProperty
isAlive() boolean
setAlive(boolean) void

Source-Code

GameOfLifeApplication.java

```
package swp4.ue06.part1.gol;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.ButtonType;
import javafx.scene.layout.FlowPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.Priority;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;


public class GameOfLifeApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        // create new vbox as root
        VBox vbox = new VBox();

        // create gameOfLIfeBoard with 100x100 grid
        GameOfLifeBoard board = new GameOfLifeBoard(50,50);
        // set min-dimensions to allow shrinking of board on resize
        board.setMinHeight(0);
        board.setMinWidth(0);
        // also allow it to fill the container
        VBox.setVgrow(board, Priority.ALWAYS);
        vbox.getChildren().add(board);

        // Button to make ONE step
        Button btnStep = new Button("Step");
        btnStep.setOnAction(a -> {
            // board::step makes changes to the UI --> Platform.runLater
required!
            Platform.runLater(board::step);
        });

        // Run and stopButton to allow autonomous steps
        Button btnRun = new Button("Run");
        Button btnStop = new Button("Stop");
        btnStop.setDisable(true);
        btnRun.setOnAction(a -> {
            // upon running, start the Thread and disable the run button (and
activate the stop button)
            board.startStepping();
            btnRun.setDisable(true);
            btnStop.setDisable(false);
        });
        btnStop.setOnAction(a -> {
            // stop the thread and reverse button disabling
            board.stopStepping();
```

```
            btnRun.setDisable(false);
            btnStop.setDisable(true);
        });

        // button to reset the state to the point before pressing the run
button
        Button btnReset = new Button("Reset");
        btnReset.setOnAction(a -> {
            Platform.runLater(() -> board.reset());
        });

        // sets all cells to a dead state
        Button btnClear = new Button("Clear");
        btnClear.setOnAction(a -> {
            Platform.runLater(() -> board.clear());
        });

        // saves the current board
        Button btnSave = new Button("Save");
        btnSave.setOnAction(a -> {
            FileChooser fc = new FileChooser();
            FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter("GOL Files (*.gol)","*.gol");
            fc.getExtensionFilters().add(extFilter);
            fc.setTitle("Please save file destination");

            File file = fc.showSaveDialog(primaryStage);
            if(file != null) {
                board.save(file);
            }
        });

        // restores state from file
        Button btnOpen = new Button("Open");
        btnOpen.setOnAction(a -> {
            FileChooser fc = new FileChooser();
            FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter("GOL Files (*.gol)","*.gol");
            fc.getExtensionFilters().add(extFilter);
            fc.setTitle("Please pick savegame");

            File file = fc.showOpenDialog(primaryStage);
            if(file != null) {
                Platform.runLater(() -> {
                    if(!board.restore(file)) {
                        Alert alert = new Alert(Alert.AlertType.ERROR, "File
couldn't be read.\nMake sure the file is well formed.", ButtonType.OK);
                        alert.setHeaderText("Error reading file");
                        alert.showAndWait();
                    }
                });
            }
        });


        // place all buttons in an hbox beneath board
        HBox hbox = new HBox();
        hbox.setSpacing(7);
```

```
        hbox.getChildren().add(btnStep);
        hbox.getChildren().add(btnRun);
        hbox.getChildren().add(btnStop);
        hbox.getChildren().add(btnReset);
        hbox.getChildren().add(btnClear);
        hbox.getChildren().add(btnSave);
        hbox.getChildren().add(btnOpen);
        vbox.getChildren().add(hbox);

        // initialize the scene, board and stage and display the stage
        Scene scene = new Scene(vbox, 500, 500);
        board.initialize();
        primaryStage.setMinHeight(350);
        primaryStage.setMinWidth(350);
        primaryStage.setTitle("Game of Life");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

GameOfLifeBoard.java

```
package swp4.ue06.part1.gol;

import javafx.application.Platform;
import javafx.beans.binding.When;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.paint.Color;


import java.io.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


public class GameOfLifeBoard extends AnchorPane {
    private final Map<Integer, List<GameOfLifeCell>> cells = new HashMap<>();
    private final Map<Integer, List<GameOfLifeCell>> backupCells = new
HashMap<>();
    private final int cols;
    private final int rows;
    private boolean running = false;

    public GameOfLifeBoard(int cols, int rows) {
        this.cols = cols;
        this.rows = rows;
    }

    // deep copy the cells to avoid changing the same cells (references!)
    private void deepCopyCells(Map<Integer, List<GameOfLifeCell>> src,
Map<Integer, List<GameOfLifeCell>> dst) {
        for(int col = 0; col < cols; col++) {
            for (int row = 0; row < rows; row++) {
                // create a new cell to avoid referencing the old one and copy
the alive status
                GameOfLifeCell cell = new GameOfLifeCell();
                if(src.get(row).get(col).isAlive()) {
                    cell.setAlive(true);
                }

                // finally copy the cell to the new map
                dst.computeIfAbsent(row, k -> new ArrayList<>());
                dst.get(row).add(cell);
            }
        }
    }

    public void initialize() {
        // iterate through all positions
        for(int col = 0; col < cols; col++) {
            for(int row = 0; row < rows; row++) {
                // create cell for each position and set its color
                GameOfLifeCell cell = new GameOfLifeCell();
                cell.setFill(Color.LIGHTGRAY);

                // bind the fillProperty to the cells alive status and color
```

```
it accordingly
                cell.fillProperty().bind(new
When(cell.aliveProperty()).then(Color.VIOLET).otherwise(Color.LIGHTGRAY));
                // on mouseclick, call the mouseClicked function
                cell.setOnMouseClicked(this::mouseClicked);

                // set the cells userdata to easily read the coordinates later
                cell.setUserData(new Coordinates(col, row));

                // add the cells to a map and to the board as JavaFX
components
                cells.computeIfAbsent(row, k -> new ArrayList<>());
                cells.get(row).add(cell);
                getChildren().add(cell);
            }
        }
        // initially "resize" the container so the cells width is determined
correctly
        resizeContainer(this);
        // additionally bind a listener to widthproperty, so the resizing the
window triggers the function
        this.widthProperty().addListener( (o, ov, nv) ->
resizeContainer(this));
        this.heightProperty().addListener( (o, ov, nv) ->
resizeContainer(this));
    }

    public void reset() {
        // delegate
        restoreFromMap(backupCells);
    }

    private void restoreFromMap(Map<Integer, List<GameOfLifeCell>>
restoreCells) {
        // restore cells alive status using the given map
        for(int col = 0; col < cols; col++) {
            for(int row = 0; row < rows; row++) {

cells.get(row).get(col).setAlive(restoreCells.get(row).get(col).isAlive());
            }
        }
    }

    public void clear() {
        // iterate over the map and set every cell as dead
        for(int col = 0; col < cols; col++) {
            for(int row = 0; row < rows; row++) {
                cells.get(row).get(col).setAlive(false);
            }
        }
    }

    public void save(File file) {
        // upon saving the current status, iterate over the map
        try (PrintWriter writer = new PrintWriter(file)) {
            for (int row = 0; row < rows; row++) {
                for(int col = 0; col < cols; col++) {
                    if(col > 0) {
```

```
                    // whitespace as delimiter
                    writer.print(" ");
                }
                // write the cells as zeroes and ones
                writer.print(cells.get(row).get(col).isAlive() ? 1 : 0);
            }
            writer.print("\n");
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}

public boolean restore(File file) {
    // keep track of the read fields and their amount
    int readFields = 0;
    Map<Integer, List<GameOfLifeCell>> readCells = new HashMap<>();

    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        // read the entire file line by line
        String s = "";
        int row = 0;
        while((s = br.readLine()) != null) {
            // split the line using whitespaces and create a new list
inside the map
            String[] valList = s.split("\s");
            readCells.put(row, new ArrayList<>());
            // afterward, check if 0 or 1 has been read and accordingly
add dead or alive cells for each value
            for(String val : valList) {
                GameOfLifeCell cell = new GameOfLifeCell();
                if(val.equals("0")) {
                    readCells.get(row).add(cell);
                } else if(val.equals("1")) {
                    cell.setAlive(true);
                    readCells.get(row).add(cell);
                }
            }
            readFields += readCells.get(row).size();
            row++;
        }
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    // the amount of read cells needs to match the space in the grid
    if(readFields == rows*cols) {
        restoreFromMap(readCells);
        return true;
    } else {
        return false;
    }
}

private void mouseClicked(MouseEvent mouseEvent) {
    // upon mouseclick, find the right cell and print its coordinates
```

```
        GameOfLifeCell cell = (GameOfLifeCell) mouseEvent.getSource();
        Coordinates coordinates = (Coordinates)cell.getUserData();
        System.out.println(coordinates);

        // then, toggle its alive status
        Platform.runLater( () -> {
            cell.setAlive(!cell.isAlive()); // ensure that the code is
executed on the ui thread
        });
    }

    private void resizeContainer(AnchorPane pane) {
        for(int col = 0; col < cols; col++) {
            for(int row = 0; row < rows; row++) {
                // calculate new positions and sizes for each cell upon
resizing
                GameOfLifeCell cell = cells.get(row).get(col);
                cell.setX((pane.getWidth() / cols) * col + 1);
                cell.setY((pane.getHeight() / rows) * row + 1);
                cell.setWidth((pane.getWidth() / cols-1) - 1);
                cell.setHeight((pane.getHeight() / rows-1) - 1);
            }
        }
    }

    public void step() {
        // make a copy of the old cells, to avoid affecting the result with
made changes
        Map<Integer, List<GameOfLifeCell>> oldCells = new HashMap<>();
        deepCopyCells(cells,oldCells);

        for(int col = 0; col < cols; col++) {
            for(int row = 0; row < rows; row++) {
                // get the status for each cell. -1 if they die, 0 if they
continue to live, 1 if they come alive.
                // set the property accordingly
                if(getState(row, col, oldCells) == -1) {
                    cells.get(row).get(col).setAlive(false);
                } else if(getState(row, col, oldCells) == 1) {
                    cells.get(row).get(col).setAlive(true);
                }
            }
        }
    }

    private int getState(int row, int col, Map<Integer, List<GameOfLifeCell>>
oldCells) {
        // count all neighbors in N8
        int neighbourCount = 0;
        for(int xOffset=-1; xOffset <= 1; xOffset++) {
            for(int yOffset=-1; yOffset <= 1; yOffset++) {
                // calculate neighbor coordinates
                int nbRow = row + xOffset;
                int nbCol = col + yOffset;
                // check if coordinates are inside bounds and different to the
initial cells
                if(nbCol >= 0 && nbCol < cols && nbRow >= 0 && nbRow < rows &&
!(xOffset==0 && yOffset==0)) {
```

```
                    // if the cell is alive, increase the counter
                    if(oldCells.get(nbRow).get(nbCol).isAlive()) {
                        neighbourCount++;
                    }
                }
            }
        }

        // check if an alive cell has too much or too little neighbors ( = dies)
        if(oldCells.get(row).get(col).isAlive()) {
            if (neighbourCount <= 1 || neighbourCount >= 4) {
                return -1;
            }
        } else { // and check if a dead cell has 3 neighbors (= becomes alive)
            if(neighbourCount == 3) {
                return 1;
            }
        }
        return 0;
    }

    public void startStepping() {
        // make a new thread to avoid blocking the UI thread
        Thread t = new Thread(new Runnable() {
            @Override
            public void run() {
                // first, save the initial state to a map
                backupCells.clear();
                deepCopyCells(cells,backupCells);
                running = true;

                // unless stopped, make one step every 250ms
                while(running) {
                    Platform.runLater(() -> step());
                    try {
                        Thread.sleep(250);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        });
        // set the thread as daemon to close it together with the UI
        t.setDaemon(true);
        t.start();
    }

    public void stopStepping() {
        running = false;
    }

    public class Coordinates {
        int column;
        int row;

        public Coordinates(int column, int row) {
            this.column = column;
```

```
            this.row = row;
        }

        @Override
        public String toString() {
            return "Coordinates{" +
                    "row=" + row +
                    ", column=" + column +
                    '}';
        }
    }
}
```

GameOfLifeCell.java

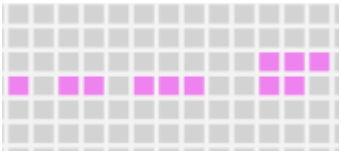
```
package swp4.ue06.part1.gol;

import javafx.beans.property.BooleanProperty;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.scene.shape.Rectangle;

public class GameOfLifeCell extends Rectangle {
    // Property to display the alive status
    private BooleanProperty aliveProperty = new SimpleBooleanProperty();

    // getter for alive property value
    public boolean isAlive() {
        return aliveProperty.get();
    }

    // set the alive property value
    public void setAlive(boolean alive) {
        aliveProperty.set(alive);
    }

    // get the alive property
    public BooleanProperty aliveProperty() {
        return aliveProperty;
    }
}
```

Testfälle

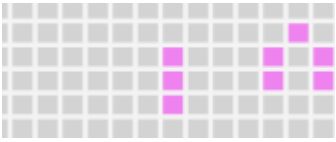
- Richtige Berechnung
- Automatische Berechnung
- Richtige Wiederherstellung
- Richtiges Clearen
- Ordnungsgemäßes Speichern und Wiederherstellen
- Aktionen während der Automatischen Berechnung

Richtige Berechnung

Initial:



Nach Step:

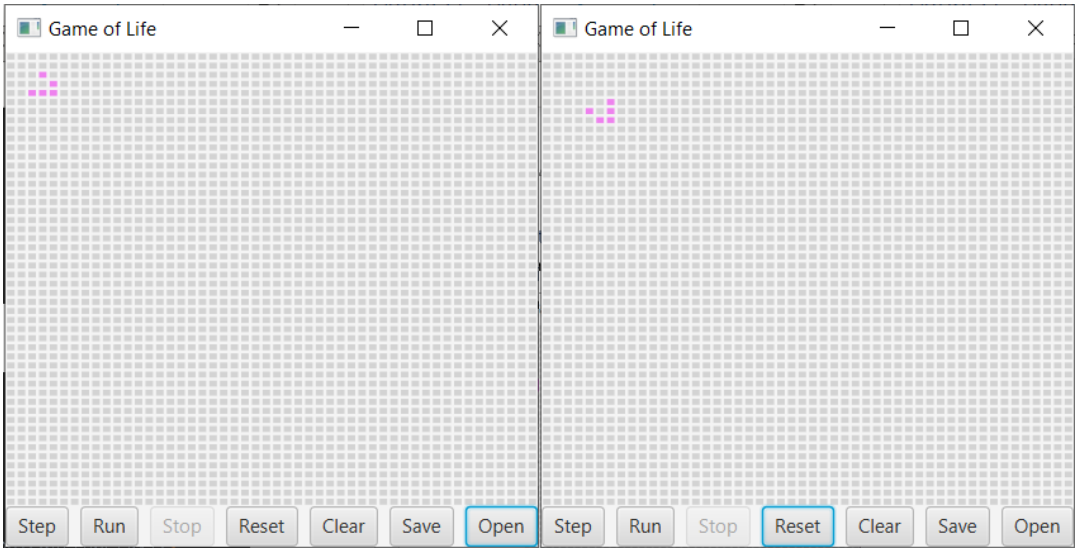


Automatische Berechnung

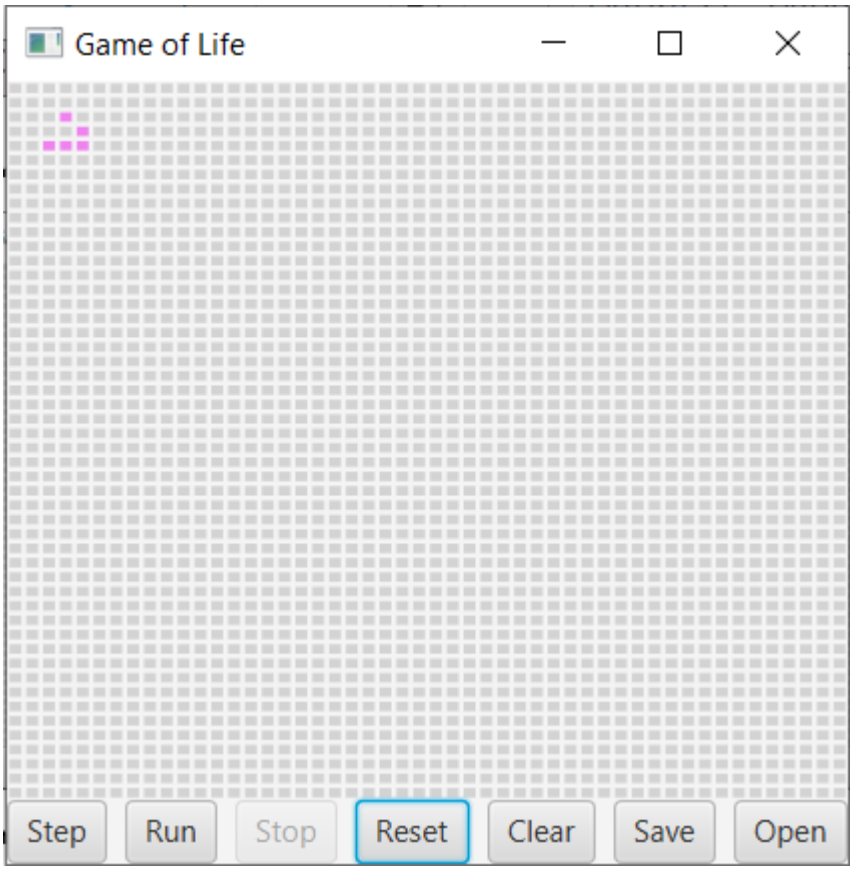
Für diesen Testfall ist eine Dokumentation schwierig bzw. wäre sie als Video erforderlich. Ich hab es jedoch getestet und auch das „Glider“-Beispiel von der Website als File zum öffnen inkludiert. Bei der manuellen Berechnung mehrmals hintereinander und der automatischen Berechnung ergeben sich die gleichen Resultate.

Richtige Wiederherstellung

Nachfolgend wird das Board zu Beginn, nach einem Run, und nach dem Reset gezeigt:

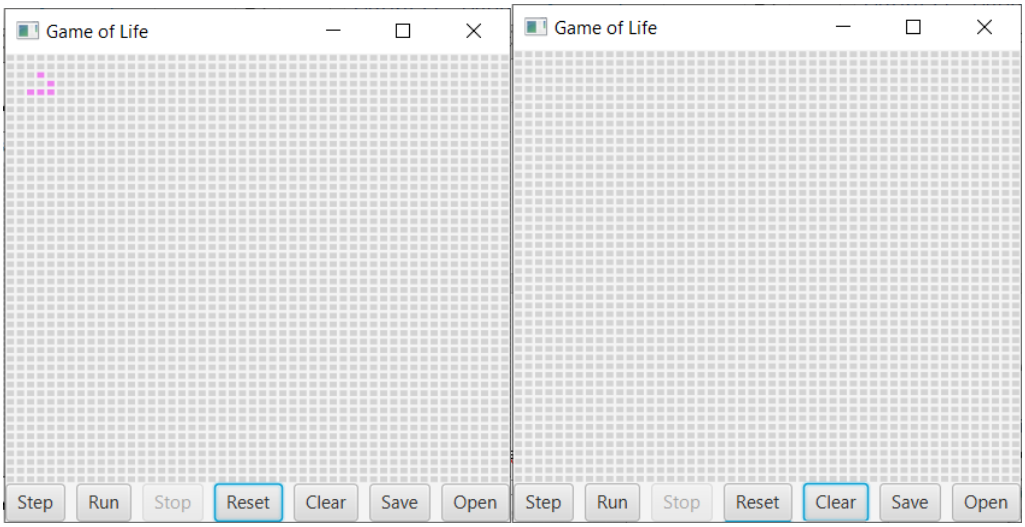


Nach dem Reset:



Richtiges Clearen:

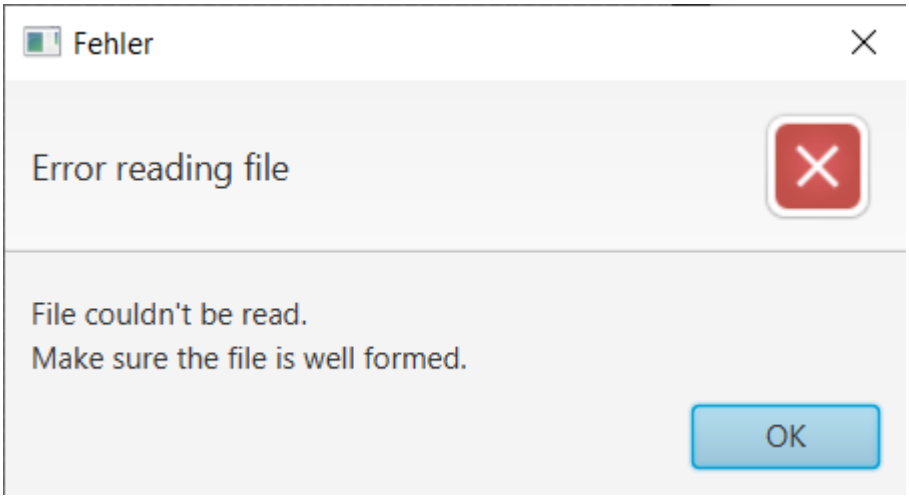
Vorher vs. Nachher:



Ordnungsgemäßes Speichern und Wiederherstellen

Wie bereits gesagt wurde das Speichern und Wiederherstellen mit dem „Glider“ (links oben) gezeigt.

Wird das File im Nachhinein bearbeitet oder entspricht nicht dem Format, so erscheint eine Fehlermeldung:



Gleiches gilt natürlich auch für leere Files.

Aktionen während der automatischen Berechnung

Das ist wiederum ein schwieriger Testfall, weswegen ich wieder erwähnen will, dass getestet wurde wie die Interaktion mit Step, Reset, Clear, Save oder Open während der automatischen Berechnung funktioniert.

Es kam dabei zu keinen Fehlern, Exceptions oder Ungereimtheiten.